Universitatea Transilvania din Brașov
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

Study profile:

Applied Informatics in German

Bachelor thesis

Cloud-Vitae

Web application for creating and sharing a Curriculum Vitae

Author: **Andrei Nicolescu**

Scientific Coordinator: **Lect. Univ. Dr. Vlad Monescu**

Brașov, 2023

Introduction

Shortly describing the theme

This paper presents a comprehensive account of the developmental process involved in creating a web application for the purpose of generating, sharing, and modifying curriculum vitae (CV) documents. The application adopts a client-server architecture and leverages modern technologies, namely Symfony 6.2 (a PHP framework), Doctrine, and Webpack Encore. The application will be deployed using conventional hosting solutions.

The underlying concept of the application revolves around facilitating CV creation for users, either from scratch or by utilizing information from pre-existing documents, without necessitating familiarity with page layout tools like Microsoft Word, or skills in photo editing or graphic design. Users are only required to input their raw personal data, which may also be automatically extracted if feasible, and the application handles the rest. Users are given the option to select their preferred design, upon which the application populates the data accordingly. In addition to prioritizing simplicity, the application endeavors to establish standard practices within the realm of online employment by promoting the adoption of novel approaches. These include transitioning from traditional email-based CV sharing to utilizing cloud storage solutions, replacing conventional paper business cards with QR codes, and facilitating seamless API integration with recruitment company platforms. By embracing these advancements, the application seeks to contribute to the evolution of the internet employment landscape while enhancing efficiency and streamlining processes across various industry platforms.

Reasoning behind this theme choice

The decision to undertake the development of this application is motivated by the need to address prevalent challenges and limitations encountered by individuals when creating and disseminating their curriculum vitae (CV). Traditional methodologies for CV creation often involve intricate page layout tools and design competencies, posing difficulties for users with restricted technical proficiency or access to resources. Furthermore, the reliance on email-based CV transmission and physical business cards can be inefficient and burdensome within the context of today's digital landscape.

The primary objective of constructing this application is to offer users a friendly and accessible solution that empowers them to effortlessly produce professional CVs. Through automated layout and design processes, the application streamlines the CV creation procedure, enabling users to concentrate solely on inputting their personal details. This approach simplifies CV development, eradicating the necessity for specialized software or design expertise. Moreover, the integration of contemporary technologies, including Symfony 6.2, Doctrine, and Webpack Encore, establishes a sturdy and scalable framework for the application. By leveraging cloud storage for CVs, embracing QR code business cards, and providing API integration with recruitment platforms, the application incorporates progressive features aligned with emerging trends in the internet employment domain.

In summary, the rationale behind developing this application derives from the aspiration to revolutionize CV creation and dissemination practices, furnishing users with a simplified and efficient tool while embracing contemporary advancements in the field of internet employment.

Document structure

1. Addressing the topic

The application addresses the complexities and limitations associated with creating, sharing, and editing curriculum vitae (CV) documents. It achieves this by providing a user-friendly interface that automates layout and design, eliminating the need for specialized software or design skills. By simplifying the CV creation process, the application aims to alleviate the challenges faced by individuals with limited technical knowledge or resources.

Moreover, the application embraces modern practices in the internet employment domain, offering solutions to existing problems. It facilitates the transition from conventional email-based CV sending to cloud storage, resolving inefficiencies and obstacles related to file management and sharing. Additionally, the adoption of QR code business cards tackles the shortcomings of traditional paper business cards, enabling easy access to comprehensive CV information in today's digital era.

Furthermore, the application recognizes the need for seamless integration with recruitment company platforms. Through API integration, it bridges the gap between job seekers and recruiters, streamlining interactions and enhancing the efficiency of the recruitment process.

In summary, the application tackles the complexities of CV creation, sharing, and editing, while simultaneously addressing problems in the internet employment landscape. It simplifies the process for users, revolutionizes CV distribution methods, replaces outdated business card practices, and facilitates seamless API integration with recruitment platforms.

1.1 The Purpose of this Thesis

The primary objective of this application is to streamline the multifaceted process of creating, sharing, and editing curriculum vitae (CV) documents. Its core purpose is to simplify these tasks by providing a user-friendly interface that automates layout and design, thereby alleviating the complexities associated with CV creation. In addition to simplification, the application also aims to drive the adoption of contemporary practices in the realm of internet employment.

One such practice is the transition to cloud storage for CVs, which offers numerous advantages over traditional methods of CV sharing and storage. By enabling users to store and access their CVs through cloud-based repositories, the application eliminates the limitations and inefficiencies associated with email-based CV sending and physical storage mediums. This modern approach enhances accessibility, version control, and collaboration among users.

Furthermore, the application introduces the use of QR code business cards as an innovative and efficient method for CV sharing. By integrating CV information into QR codes, users can effortlessly share their comprehensive profiles with others, eliminating the need for traditional paper business cards and enhancing digital networking capabilities.

Additionally, the application facilitates seamless integration with recruitment platforms through the provision of an application programming interface (API). This integration enables smooth data exchange, streamlined communication, and enhanced efficiency between job seekers and recruitment companies. By embracing API integration, the application promotes interoperability and integration with various recruitment platforms, enhancing the overall user experience.

By adopting these practices, the application endeavors to streamline CV sharing procedures, amplify visibility for job seekers, and facilitate seamless interactions with recruitment entities.

In summary, the overarching goal of this application is to simplify the processes of creating, sharing, and editing CVs, while concurrently promoting contemporary practices in the internet employment domain. By offering a user-friendly interface, transitioning to cloud storage, adopting QR code business cards, and facilitating API integration with recruitment platforms, the application aims to enhance the efficiency, accessibility, and effectiveness of CV-related activities in the modern employment landscape.

1.2 Alternative approaches in this field

The concept of web-based solutions for curriculum vitae (CV) management is not new, as there already exist numerous solutions that address specific challenges. However, no comprehensive all-in-one web solution has emerged that encompasses all the stated functionalities. Examples of such existing solutions include online CV builders, cloud-based CV management systems, and CV parsing tools.

Online CV builders have gained popularity due to their user-friendly interfaces and diverse template options, streamlining the CV creation process. These platforms (i.e. Zety, LiveCareer and MyPerfectResume) offer intuitive data entry features, allowing users to input information effortlessly. With customizable layout choices, fonts, and styles, users can tailor their CVs to their preferences. This solution proves beneficial for individuals lacking design skills or seeking a quick and efficient method to generate professional-looking CVs.

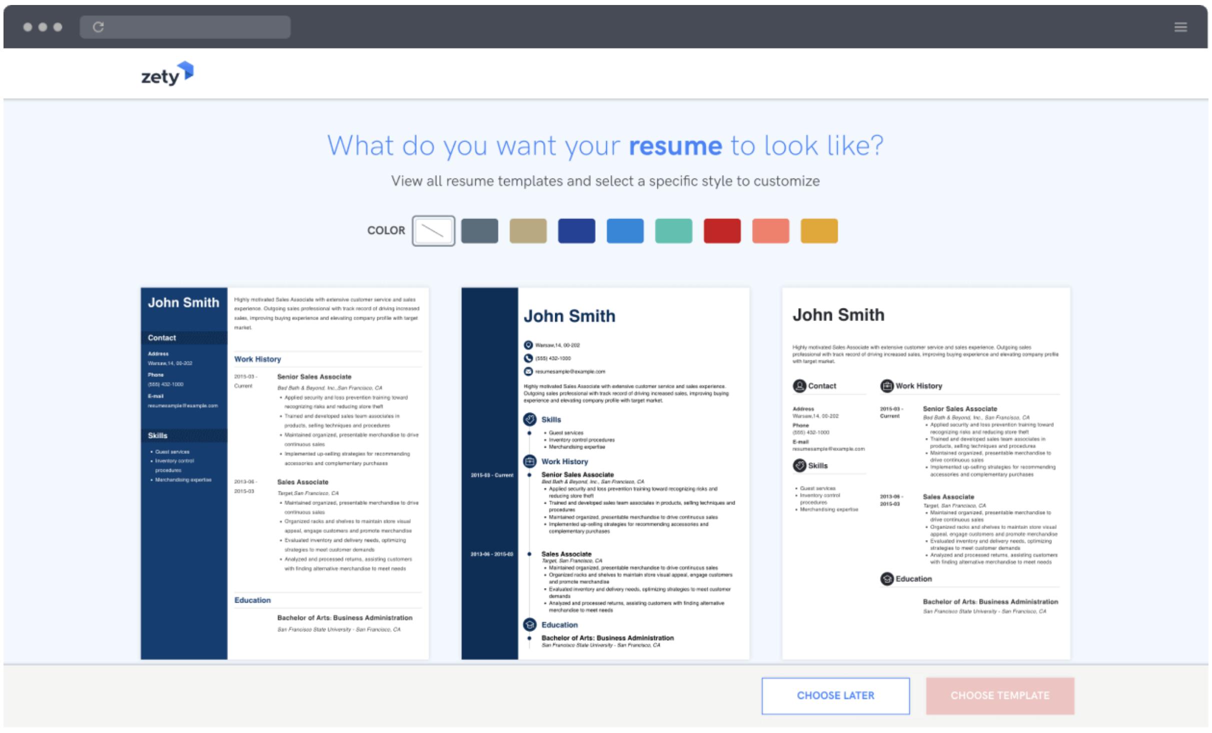
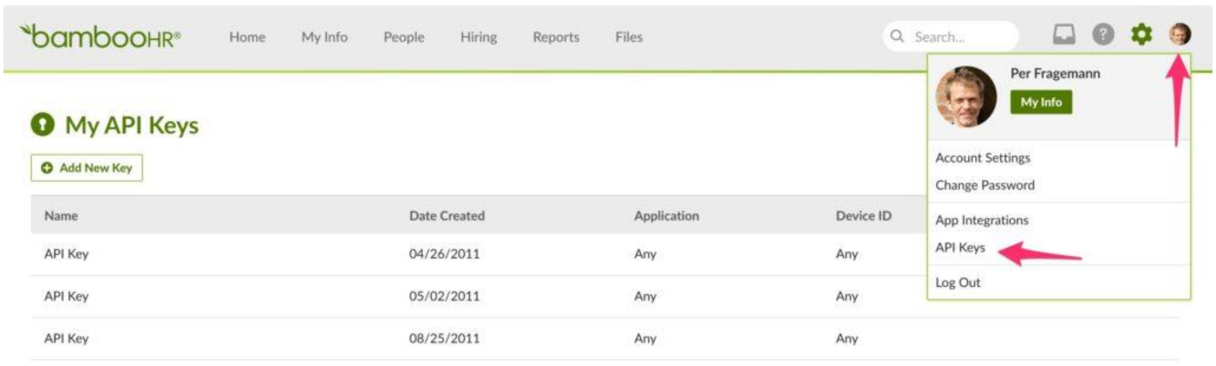

zety
What do you want your resume to look like?
View all resume templates and select a specific style to customize
COLOR
John Smith
John Smith
John Smith
CHOOSE LATER
CHOOSE TEMPLATE

Fig. 1 – Example of online CV builder (Zety)

Cloud-based CV management systems (i.e. BambooHR, Greenhouse and Submittable) provide a centralized platform for storing, managing, and updating CVs. Users can securely access their CVs from any device with an internet connection. Collaboration features enable CV sharing with employers or recruiters for review and feedback. Cloud storage ensures data integrity and availability, mitigating the risk of CV loss due to hardware failures or accidents. Additionally, these systems typically offer version control capabilities, enabling users to track changes and revert to previous versions when necessary.



My API Keys

Name	Date Created	Application	Device ID
API Key	04/26/2011	Any	Any
API Key	05/02/2011	Any	Any
API Key	08/25/2011	Any	Any

Fig. 2 – Example of cloud-based CV management systems (BambooHR)

CV parsing tools (i.e. Sovren Resume Parser, Rchilli and HireAbility) employ advanced algorithms and natural language processing techniques to extract pertinent information from existing CVs, transforming it into a structured format. These tools automate the data entry process, ensuring accurate extraction of crucial details such as work experience, education, and skills. By eliminating manual entry, CV parsing tools save time and reduce the likelihood of errors. HR professionals and recruiters handling large volumes of CVs can particularly benefit from this solution's ability to streamline processing tasks.

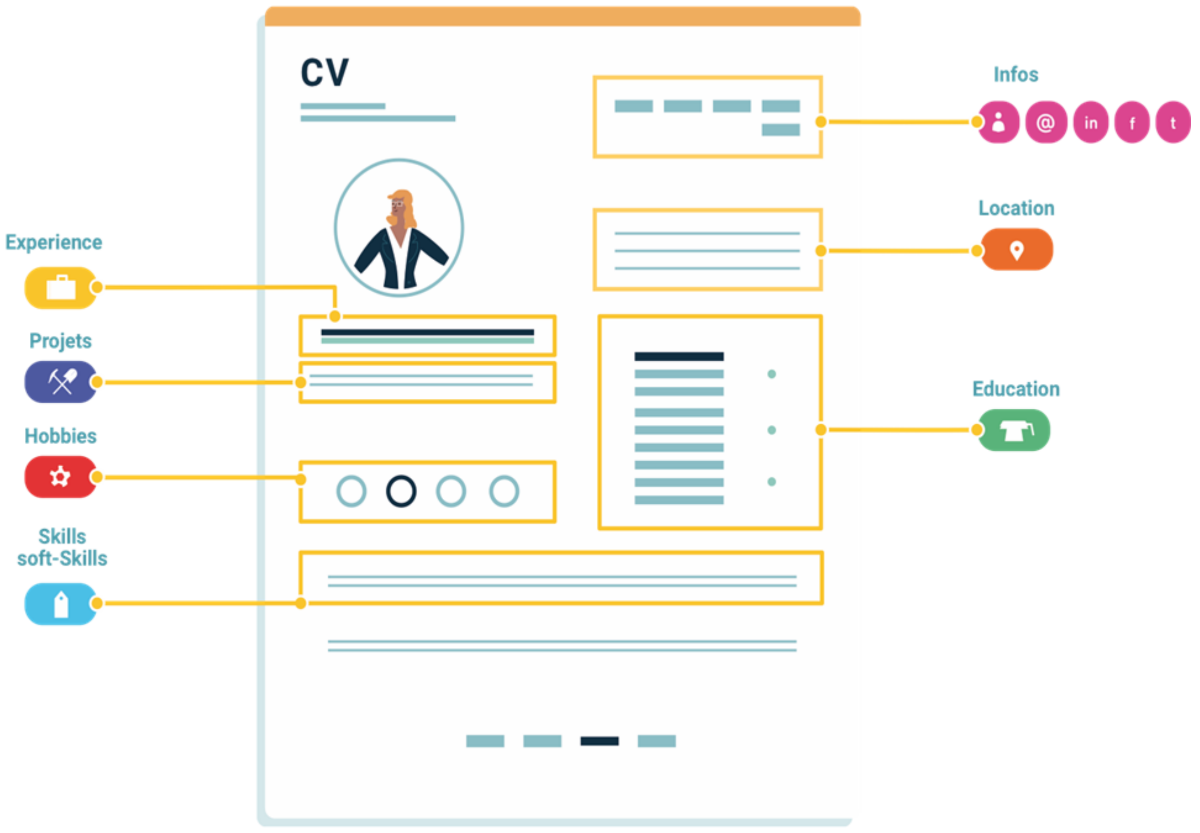


Fig. 3 – Example of how CV parsing tools extract data

However, despite the existence of these individual solutions, an encompassing all-in-one web solution that integrates the aforementioned functionalities is yet to materialize. The development of such a comprehensive solution remains a promising avenue in the field of CV management and would provide users with a holistic and streamlined approach to address the multifaceted challenges associated with CV creation, storage, and sharing.

1.3 Designing a web application

The initial phase of developing a web application entails establishing its functionality and creating preliminary documentation. In this process, it is crucial to define the criteria for developing the Minimum Viable Product (MVP). The concept of the MVP, as popularized by Eric Ries in his book "The Lean Startup" [1] aims to validate the effectiveness of a solution in a real production environment. Accordingly, the Cloud-Vitae application can be considered an MVP as it represents a fully functional prototype that effectively addresses the aforementioned problems with minimal effort.

1.3.1 The specifications of a web application

Following the development of a web solution, it is essential to provide accompanying instructions for its usage. However, the process of writing detailed specifications can be time-consuming. In accordance with Agile principles, as described in chapter 1.3.2.1, the emphasis is placed on delivering an intuitive and functional product rather than dense documentation. Thus, prioritizing the development of an efficient and user-friendly solution aligns with the Agile approach, recognizing the value of practical implementation over extensive written documentation.

The creation of a Minimum Viable Product (MVP) typically involves a systematic approach encompassing several proven steps. These steps include conducting a thorough market analysis, gaining a comprehensive understanding of the end users, defining the desired final product, and determining the essential functionalities to be incorporated. This sequential process has demonstrated its effectiveness in guiding the development of an MVP, ensuring that the resulting product aligns with market demands, addresses user needs, and encompasses the required core features.

1.3.2 The design of Cloud-Vitae

During the design phase of developing this application, meticulous attention was given to several sequential levels, each playing a crucial role in the overall process. These levels, presented in a systematic order, are as follows:

- Identifying the target audience
- Determining the base functionalities
- Selecting the appropriate application type
- Establishing the overall look and design

1.3.2.1 Identifying the Target Audience

The intended target audience of this application primarily comprises of individuals seeking a streamlined and effective solution to facilitate the creation, sharing, and modification of curriculum vitae (CV) documents. This encompasses a diverse range of users, including job seekers, professionals, students, and individuals in need of well-crafted and professional CVs. The application strives to accommodate users with varying degrees of technical proficiency and design expertise by providing an intuitive interface and automated layout functionalities. Moreover, the application's integration with recruitment platforms and incorporation of contemporary practices within the internet employment sphere render it pertinent to human resources (HR) professionals, recruiters, and organizations involved in the employment process.

1.3.2.2 Determining the Base Functionalities

Given the extensive potential user base of Cloud-Vitae, it becomes impractical to address specific individual needs, necessitating the development of a highly scalable, modular, and user-friendly web solution by default. Consequently, the application must encompass, at a minimum, the fundamental functionalities already prevalent in the market. Thus, the base functionalities of Cloud-Vitae are as follows:

- CV Creation
- CV Customization
- CV Sharing
- Cloud Storage Integration
- API Integration with Recruitment Platforms
- User management

The application affords users with a convenient means to effortlessly generate professional curriculum vitae (CVs). Through an intuitive user interface, individuals can input personal details encompassing work experience, education, skills, and contact information. The application automates the intricate process of designing and arranging CV elements, eliminating the necessity for specialized software or design expertise. Users are empowered to tailor the visual presentation and structure of their CVs according to their preferences, selecting from an array of templates, fonts, colors, and styles that facilitate personal branding.

Streamlining the dissemination of CVs, the application provides facile options for sharing with prospective employers, recruiters, or relevant stakeholders, encompassing email transmission, cloud-based storage, or the creation of QR code business cards. By simplifying CV distribution, the application heightens visibility for job seekers. Furthermore, users benefit from the secure storage of their CVs within the cloud, ensuring accessibility from any internet-connected device while mitigating the risks associated with hardware failures or inadvertent data loss. The cloud storage functionality also facilitates seamless updates and version control for CVs.

To expedite the application process and enhance the interaction between job seekers and hiring organizations, the application integrates with recruitment platforms through its application programming interface (API), facilitating the efficient exchange of data. Such integration optimizes the overall job-seeking process and streamlines communication channels. Moreover, the application incorporates user management features, enabling individuals to create accounts, manage profiles, and easily switch between multiple versions of their CVs. This empowers users to easily update their information and track their CV history. In summation, the application strives to simplify the intricacies of CV creation, customization, sharing, and storage, while incorporating contemporary practices and technologies that amplify the user experience and elevate efficiency in the pursuit of employment opportunities.

1.3.2.3 Selecting the Appropriate Application Type

Cloud-Vitae can be classified as a dynamic web application, which exhibits a higher degree of technical complexity compared to other types of web applications. Dynamic web applications utilize databases to load and update data dynamically, ensuring that the content is refreshed each time a user interacts with the application. In the case of Cloud-Vitae, it incorporates characteristics akin to a single-page application (SPA). However, instead of altering the loaded page data, the application's backend responds with a different render on the same page whenever a request is made. This approach enables seamless user experience while still being able to utilize full page reloads which efficiently deliver updated content.

1.3.2.4 Establishing the overall Look and Design

The establishment of the overall look and design of the application involved a systematic and detailed approach to create a visually appealing and user-friendly interface. Significant consideration was given to the selection of color palettes, typography, and visual elements that were in harmony with the application's intended objectives and intended user base. The design elements underwent careful refinement to optimize readability, usability, and the overall user experience. Furthermore, the arrangement of the interface components and the navigation system were strategically devised to facilitate intuitive interaction and seamless access to the application's diverse functionalities. By employing a meticulous design process, the application achieves a visually coherent and attractive interface that effectively engages users, while reflecting a contemporary and professional aesthetic.

1.3.3 Project planning

Thorough planning is an essential aspect of developing a web application, even in cases where the project is undertaken by a single individual. Throughout the years, various software development methodologies (SDM) have been employed, but the consensus has emerged that an iterative and incremental approach is most effective. Notably, Agile methodologies have gained widespread adoption in application development. Originating in Utah, USA, in 2001, Agile methodologies were initially conceived by a group of 17 prominent programmers of that era, including figures such as Martin Fowler and Robert Martin. [2] These practitioners endorsed the "Manifesto for Agile Software Development", [3] which ushered in a paradigm shift in the design and creation of applications, permanently transforming industry practices.

1.3.3.1 Agile methodologies

Agile methodologies encompass a collection of approaches that enable a flexible and adaptive view to software development, as implied by their name. These methodologies emphasize goal-oriented project management characterized by continuous iterations, encompassing both development and testing phases, commonly referred to as sprints. Agile methodologies are widely recognized as the most straightforward and efficient means of translating a vision into a software solution. They entail ongoing planning and improvement, fostering a learning mindset, fostering collaborative teamwork, fostering exponential growth in development, and facilitating timely software delivery. Several prominent approaches are commonly employed within Agile development, including iterative and incremental development, maintaining regular and direct communication with clients, segmenting projects into design-oriented individual models, concurrent error handling, employing short iteration time frames typically lasting between 2 to 4 weeks, and incorporating regression testing, in which testing occurs after each iteration to identify and rectify any issues.

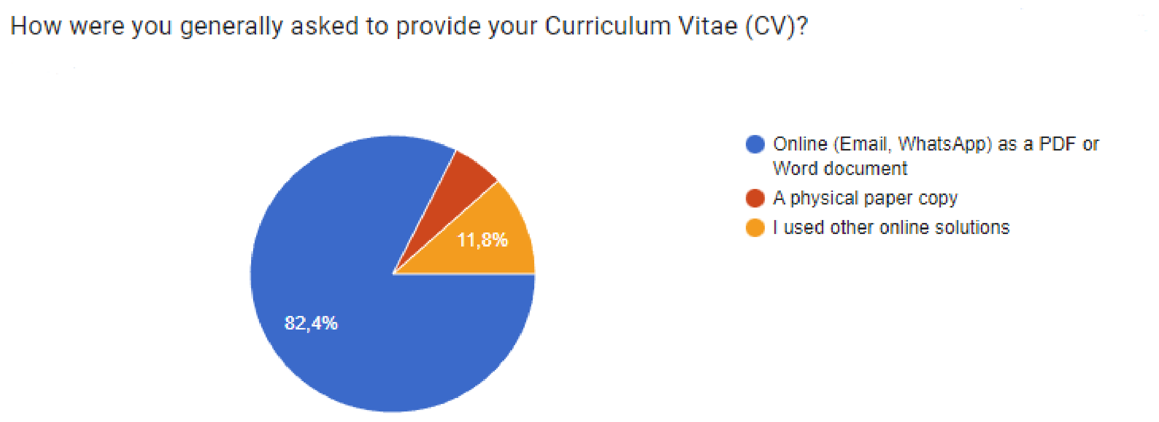
1.3.3.2 User stories in Agile development

In Agile development, user stories play an essential role in capturing and communicating the requirements of end users. These stories are concise and user-centric descriptions that outline specific features or functionalities, encompassing the needs, goals, and expectations of the users. They are written in normal languages (no code is required), in a way that is easy to understand and that best describes the user's role in the application. They are framed from the perspective of distinct user roles, delineating the "who," "what," and "why" of each requirement. [2]

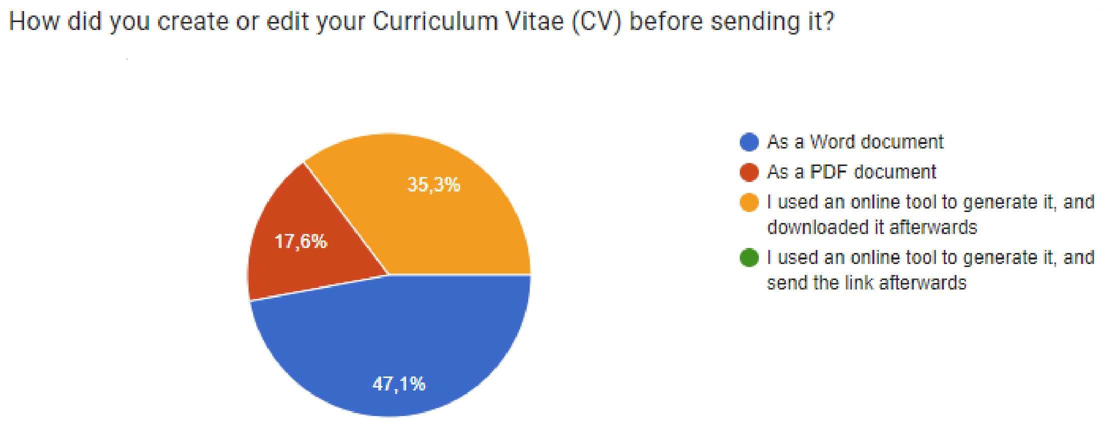
Within the context of Cloud-Vitae, user stories are employed to identify and articulate the specific needs of the target audience, namely job seekers, professionals, and students. An illustrative user story example would be: "As a job seeker, I aspire to effortlessly input my work experience and educational details during the CV creation process, facilitating the presentation of a comprehensive overview of my qualifications to prospective employers." User stories serve as a foundation for prioritizing features, planning iterations, and ensuring the effective fulfillment of user requirements. While Cloud-Vitae is developed by a single individual without external stakeholders, user stories still provide valuable insights into the desired appearance of the final product and inform the shaping of the application to optimize user experience.

1.3.3.2 Client communication

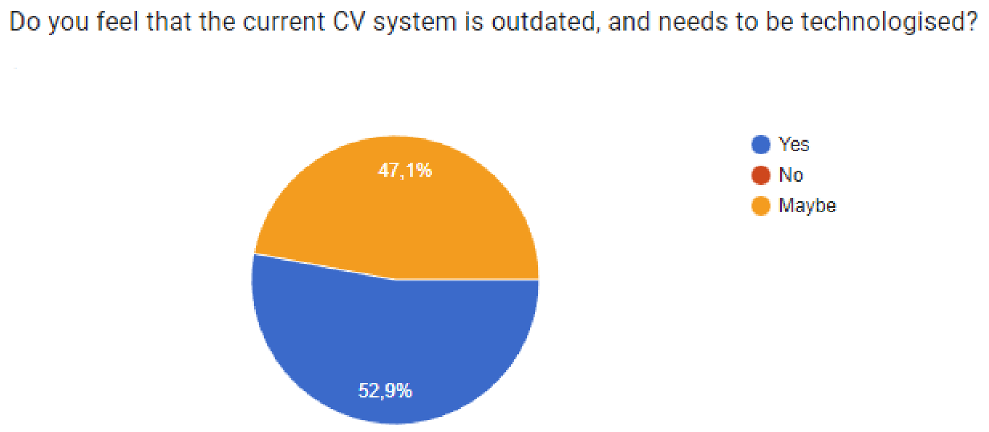
As stated earlier, the absence of external stakeholders and the presence of a sole developer in the case of this application deviate from the typical Agile methodology, which emphasizes continuous communication and feedback from clients. To overcome this, a modified approach was adopted while developing Cloud-Vitae, wherein the end user itself is considered the client. In order to validate the information gathered through user stories and ensure alignment with user needs, a select group of potential users, closely aligned with the application's objectives, was identified. Throughout the various development phases, these users actively participated in the process. They were engaged in activities such as completing quizzes and forms to gain insights into their preferences and requirements. Additionally, during each sprint, they played a crucial role in the testing phase, providing direct feedback on the newly implemented features. This active involvement of the selected user group was a fundamental step in the realization of the overarching goal of Cloud-Vitae.



As we can see by the submitted answers, over 80% of users were required to send their CVs as an electronic copy online.



However, none of the users reported sharing their CVs as a link to a cloud storage solution. This is just one of the many areas of opportunity that Cloud-Vitae aims to revolutionize. Compared to other industries, the employment world falls behind from a technological standpoint, and considering the advancements made in the last century, even the users are able to identify this problem, as we can understand from the graph bellow.



2. Technology and Development Stack

In programming, a development stack refers to a combination of software tools, frameworks, libraries, and technologies that are used together to develop and deploy applications. It is a set of tools that work synergistically to provide a complete development environment and support the entire software development lifecycle, and it is crucial to select them before entering the development stage.

2.1 PHP as a Programming Language

A programming language serves as a formal means of communication between humans and computers, enabling the transmission of instructions that can be understood and executed by the computer. It encompasses a defined set of rules and syntax that empowers programmers to express their algorithms and logic in a structured manner. Programming languages are extensively employed in the development of software applications, websites, and other computer programs, facilitating the creation of complex systems by providing a medium through which programmers can articulate their instructions effectively. [4] Each programming language possesses its own distinctive syntax and semantics, dictating how code is written and subsequently interpreted by the computer. In the context of Cloud-Vitae, the PHP (Personal Home Page / Hypertext Preprocessor) programming language has been utilized.

PHP has emerged as a prominent server-side scripting language extensively utilized in web development. Originally conceived by Rasmus Lerdorf in 1994, PHP has undergone significant advancements and has garnered widespread adoption as a programming language for constructing dynamic websites and web applications. PHP is designed to seamlessly integrate within HTML code, facilitating the combination of PHP and HTML to generate dynamic web pages. Its execution occurs on the server side, wherein the PHP code is processed on the web server before transmitting the resulting HTML to the client's web browser. This capability empowers PHP to dynamically generate content, interact with databases, handle form submissions, and perform diverse server-side operations.

One of the notable advantages of PHP lies in its user-friendly nature and accessibility. It features a straightforward syntax akin to that of programming languages like C, rendering it relatively easy for beginners to grasp. Additionally, PHP benefits from a vibrant and active community of developers, ensuring a plethora of resources, libraries, and frameworks available to support PHP development. The language provides extensive support for a range of databases, including MySQL, PostgreSQL, and Oracle, thereby making it suitable for constructing database-driven web applications. Furthermore, PHP boasts robust support for web protocols and seamless integration with external application programming interfaces (APIs), thereby exhibiting versatility in terms of interoperability with other systems and services.

Overall, PHP stands as a potent and widely adopted programming language for web development, bestowing developers with flexibility, scalability, and a comprehensive array of features to create dynamic and interactive websites and web applications.

2.2 Symfony Framework and it's Components

In software development, a framework is a pre-established collection of tools, libraries, and guidelines that offers a structured and standardized approach to application development. It provides developers with a foundation and reusable components that facilitate the coding process by handling common tasks and offering a framework for organizing code. Typical features found in frameworks include database access, session management, input validation, and routing, among others, which contribute to improved development efficiency and productivity. [5]

One popular framework used in web application development is Symfony Framework. It is an open-source PHP framework that follows the Model-View-Controller (MVC) architectural pattern (chapter 3.2). Symfony Framework offers a flexible and robust foundation for building complex web applications, promoting code reusability, modular design, and maintainability.

Symfony Framework encompasses a wide range of features and components, including a comprehensive set of libraries, tools, and utilities. It provides a powerful routing system for mapping URLs to controllers, a render engine for creating dynamic views (chapter 2.2.4), and an object-relational mapping (ORM) layer for database interaction (chapter 2.2.2). Additionally, Symfony supports internationalization and localization, security mechanisms such as authentication and authorization (chapters 4.1 and 4.2), and a command-line interface (CLI) for automating tasks (chapter 2.2.5).

One of the notable strengths of Symfony lies in its focus on best practices and adherence to industry standards. It encourages the use of design patterns, coding conventions, and testing methodologies, which contribute to the development of maintainable and scalable applications. Symfony also promotes modular development through the use of bundles, enabling developers to easily add or remove functionality as needed.

In summary, Symfony Framework serves as a comprehensive and mature toolkit for PHP web development, empowering developers to create robust, scalable, and high-performance applications with efficiency and consistency. It simplifies the development process by providing a solid foundation and a wide range of features, ultimately saving time and effort while ensuring the quality of the codebase.

2.2.1 Dependency management with Composer

In PHP, dependencies are external resources such as libraries, modules, or packages that a PHP application relies on to achieve proper functionality. These dependencies offer additional features, reusable components, or specialized functionalities that enhance the capabilities of the application. The range of dependencies in PHP varies from versatile libraries that handle common tasks like database interactions, file manipulation, and HTTP requests to more specific libraries tailored for web development, image processing, authentication, and other specific purposes. When a PHP application depends on external resources, it is essential to ensure that these dependencies are correctly installed and accessible. This involves effectively managing the versions and compatibility of the dependencies, resolving any conflicts that may arise when using different libraries, and guaranteeing that all necessary components are available for the application to operate accurately. Proper management of dependencies ensures that the application can effectively leverage external resources and function as intended.

Composer is a valuable dependency management tool designed specifically for PHP projects. It serves the purpose of simplifying the management of external libraries and packages required by a PHP application. Developers can utilize Composer to specify the necessary dependencies for their project, including both Symfony components and bundles, as well as other external libraries. This is accomplished through the creation of a “composer.json” file, where developers define the specific versions or version ranges of the dependencies. By leveraging this file, Composer effectively resolves and downloads the required libraries from Packagist, a comprehensive online package repository. [6]

Composer not only ensures the retrieval of the specified dependencies but also handles their respective dependencies, thus guaranteeing a consistent and compatible set of libraries for the Symfony application. Moreover, Composer provides developers with a command-line interface to execute various dependency management tasks, such as installation, updating, and removal of packages (chapter 5.3.5). Additionally, Composer generates an optimized autoloader, enabling efficient loading of the necessary classes from the installed packages. By incorporating Composer into the development process, Symfony Framework streamlines the management of dependencies, enabling developers to seamlessly integrate and leverage third-party libraries and components within their Symfony applications. Composer contributes to promoting modularity and facilitating code reuse, ultimately enhancing the overall efficiency and maintainability of Symfony projects.

2.2.2 Doctrine and Object Relational Mapping

Doctrine is an ORM (Object Relational Mapping) library that is widely used in Symfony and other PHP frameworks. It is an essential component of Symfony for several reasons. Firstly, Doctrine simplifies the complexity of database interactions by mapping database tables to PHP objects, making database operations more straightforward and improving the maintainability of the code. It provides an ORM system that automates SQL query generation, optimizes querying, caching, and handles relationships between entities. [7]

Doctrine includes a flexible query language called Doctrine Query Language (DQL), which enables developers to write complex database queries using object-oriented syntax. This facilitates expressing entity relationships and performing advanced database operations. Additionally, Doctrine provides tools for managing database schemas, such as generating database tables based on entity definitions, handling database migrations, and ensuring schema consistency.

In the Symfony framework, Doctrine seamlessly integrates with the core components. Symfony offers native support for Doctrine, making it easy to configure and use in Symfony applications. Doctrine works in harmony with Symfony's dependency injection container and configuration system. Symfony components, including the Form component, integrate smoothly with Doctrine, simplifying the creation and handling of forms based on entity definitions.

Doctrine benefits from an active and supportive community of developers who contribute to its development and provide extensive documentation, tutorials, and resources. It also integrates well with other popular libraries and tools, expanding its capabilities and enhancing the development experience. Overall, Doctrine plays a critical role in Symfony by simplifying database interactions, improving code maintainability, and offering powerful ORM capabilities.

2.2.3 Asset Management and Webpack Encore

Webpack Encore is a library specifically used in Symfony to handle and compile frontend assets such as JavaScript, CSS, and images. It greatly simplifies the process of bundling and optimizing these assets for deployment in a production environment. Webpack Encore is built on top of Webpack, a well-known module bundler for JavaScript applications. In Symfony, Webpack Encore plays a crucial role in managing and integrating frontend assets within the application. It offers a user-friendly API that allows developers to define entry points for their assets, specify dependencies, and configure loaders and plugins to process and transform the assets as needed. [8]

One of the key benefits of using Webpack Encore is that it enables developers to effectively organize and modularize their frontend code. It fully supports modern JavaScript frameworks like React, Vue.js, and Angular, empowering developers to leverage component-based coding and take advantage of advanced frontend development techniques. When it comes to integration with Symfony, Webpack Encore seamlessly fits into the framework's asset management system. This allows developers to easily reference and include their compiled assets in Twig templates or other parts of the application. Webpack Encore automatically generates the appropriate HTML tags for incorporating the compiled assets, ensuring cache-busting and guaranteeing that the latest versions are loaded.

Furthermore, Webpack Encore provides a development server that includes a hot module replacement (HMR) feature. This functionality allows developers to witness immediate changes in the browser without manually refreshing the page. The inclusion of HMR significantly speeds up the development process and enhances the overall experience for developers.

2.2.4 HTML and Twig Render Engine

HTML, or Hypertext Markup Language, is a standard markup language used to structure and present web pages. It defines tags and elements that represent various components of a webpage, including headings, paragraphs, images, links, forms, and more. HTML allows developers to organize content and define its layout, enabling web browsers to interpret and display the page correctly. In Symfony, Twig is the templating and rendering engine used to dynamically generate HTML. Twig is a robust and secure templating language designed to simplify the process of rendering views in Symfony applications. [9]

Twig acts as an intermediary between PHP code and the HTML output. It enables developers to create templates that combine HTML structure with dynamic content. These templates contain both HTML code and Twig-specific syntax and expressions. Using Twig, developers can access and display data from PHP variables, perform logical operations, iterate over data sets, apply filters to modify content, and more. The Symfony framework employs Twig to separate the presentation layer (HTML) from the business logic (PHP), adhering to the Model-View-Controller (MVC) architectural pattern. In this pattern, the view is responsible for presenting data to users. Twig empowers developers to build reusable and modular templates that are easy to maintain and customize.

Symfony integrates Twig as its default templating engine, providing seamless integration and advanced features. This integration encompasses features such as template inheritance, layout management, and internationalization support. Twig templates in Symfony can extend a base template while overriding specific blocks to tailor the content. This fosters code reuse and consistency throughout the application. By utilizing Twig, Symfony enhances security by automatically escaping user-generated content. This precautionary measure mitigates common security vulnerabilities, including cross-site scripting (XSS) attacks, by ensuring that user input undergoes proper sanitization before being displayed in the HTML output.

In summary, Twig simplifies the process of generating HTML output in Cloud-Vitae by separating the presentation layer from the underlying PHP code. It offers a flexible and secure templating language that promotes code reusability, modularity, and consistent design. With Twig, developers can easily create dynamic and interactive views while maintaining a clear separation of concerns between the application logic and the HTML representation.

2.2.5 Symfony's CLI

A CLI, also known as a Command Line Interface, is a user interface that relies on text-based commands to interact with a computer program or operating system. Instead of using graphical elements, a CLI allows users to enter commands through a command prompt or terminal window.

In a CLI, users input text commands, which are then interpreted by the underlying software or operating system. These commands can trigger specific actions, retrieve information, modify settings, and perform a wide range of tasks. CLI interfaces are commonly utilized in operating systems, programming languages, and software applications that require advanced or specialized interactions. They are particularly favored by developers, system administrators, and power users due to their efficiency, flexibility, and the ability to automate tasks through scripting. By using a CLI, users can navigate directories, execute programs, manage files, configure system settings, install packages, and perform various other operations by typing specific commands and providing any necessary arguments. CLI commands typically follow a specific syntax and may include predefined options and parameters.

Although CLI interfaces may initially seem less intuitive for beginners compared to graphical interfaces, they offer several advantages. These include greater control over system operations, the ability to automate tasks through scripting, increased flexibility, and the capability to work in resource-constrained environments or remote systems using SSH (Secure Shell). Symfony's CLI is an essential tool within the Symfony framework that empowers developers to interact with their Symfony applications through the command line. It offers a user-friendly approach to executing diverse commands related to application development, configuration, testing, and more.

The Symfony CLI builds upon the robust Symfony Console component, which furnishes the necessary infrastructure for creating and executing command-line commands. Each Symfony CLI command is implemented as a PHP class that extends the base Command class inherited from the Symfony Console component.

To utilize the Symfony CLI, developers typically open a terminal or command prompt, navigate to the root directory of their Symfony project, and execute desired commands by inputting "php bin/console" followed by the command name, along with any required arguments or options. Symfony CLI commands encompass a broad spectrum of functionalities, including:

- Symfony CLI furnishes commands for generating code skeletons, creating CRUD operations for entities, managing database migrations, and launching a local development server.
- Developers can employ Symfony CLI commands to handle project dependencies, such as installing and updating Composer packages.
- Symfony CLI provides commands for executing tests, performing code quality checks with linters, generating code coverage reports, and more.

- Symfony CLI commands facilitate the examination and modification of various configuration settings within a Symfony application, encompassing environment variables, routing configurations, caching mechanisms, and others.
- Symfony CLI offers commands to aid in debugging and profiling applications. This includes examining request/response information, executing console commands in the context of a specific request, and other debugging features.

Symfony CLI commands are often accompanied by additional options and arguments that allow developers to customize their behavior. For comprehensive details about each command and its available options, developers can execute "php bin/console command_name --help". By harnessing the capabilities of the Symfony CLI, Cloud-Vitae was able to effectively manage and streamline diverse development tasks, automate repetitive actions, and interact with the Symfony application through a command-line interface. This invaluable tool enhances productivity and provides developers with a consistent and potent set of resources for Symfony application development. To demonstrate how easy to use Symfony's CLI is, let's take the example of creating a User Entity. Using this tool, we can complete this task without writing a single line of code. In the CLI, we will run the following commands:

```
root@5c5fe570be08:/app# php bin/console make:user
```

This command will create the User Entity based on our specified preferences. Let's say we want to have an authentication and registration system for our users. We will have to just run the following commands:

```
root@5c5fe570be08:/app# php bin/console make:auth
```

```
root@5c5fe570be08:/app# php bin/console make:registration
```

Now that we are done with creating the necessary PHP files and logic, we still have to implement the same logic on our database layer. But instead of analyzing the generated code and trying to replicate the same logic ourselves, thanks to the auto-generated Doctrine ORM annotations in the User Entity (chapter 2.2.2), the Symfony CLI knows how to do that as well. We just have to use these commands:

```
root@5c5fe570be08:/app# php bin/console make:migration
```

```
root@5c5fe570be08:/app# php bin/console doctrine:migrations:migrate
```

The first command will generate a migration file (a PHP file containing database queries) that implements the Entity logic into the database client (the migrations are located under app/migrations, and they are named after the timestamp of the moment they were created). The second command executes all the migrations generated by the first one. However, these commands just create the basic logical structure of our application. In order to develop the application further, there still is a lot of code implementation left to do.

2.3 PHP Storm as an Integrated Development Environment

An Integrated Development Environment (IDE) is a software application that offers a comprehensive set of tools and features to assist in software development tasks. It serves as a centralized platform where programmers can write, modify, debug, and test their code. IDEs typically include a code editor with functionalities like syntax highlighting, code completion, and code navigation. They also provide debugging tools, automation for building software, and integration with version control systems.

PHP Storm is a widely used IDE specifically designed for PHP development. Developed by JetBrains, it offers a diverse range of features and capabilities that aim to enhance productivity and streamline the PHP development process. PHP Storm presents an intuitive and user-friendly interface, incorporating advanced coding assistance, intelligent code completion, and real-time error detection. It caters to various web technologies and frameworks commonly employed in PHP development, such as Symfony, Laravel, and WordPress, providing specific features tailored to these frameworks. [10]

One of PHP Storm's notable strengths lies in its robust debugging capabilities, empowering developers to set breakpoints, step through code execution, and inspect variables to efficiently identify and resolve issues. It seamlessly integrates with version control systems like GIT, enabling smooth collaboration among developers and facilitating effective codebase management. Moreover, PHP Storm encompasses tools for database management, code refactoring, testing, and deployment, making it a comprehensive solution for PHP development needs.

Furthermore, PHP Storm supports customization and extensibility, allowing developers to personalize their development environment according to their preferences and workflow. It offers a broad range of plugins and extensions that expand its functionality and introduce additional features.

2.4 Running a Web Server with NGINX

NGINX (abbreviation from “engine-x”) is a popular web server software that Symfony applications use to enhance their performance, scalability, and security. Acting as a reverse proxy and load balancer, NGINX efficiently manages HTTP requests and directs them to the appropriate Symfony endpoints. It also serves static files directly, reducing the processing load on Symfony. NGINX supports caching of static content and dynamic responses, leading to improved response times.

Additionally, it handles SSL/TLS encryption and decryption, offloading the computational burden from Symfony. NGINX's load balancing capabilities allow it to distribute requests among multiple backend servers running Symfony, increasing application availability. It acts as a reverse proxy, adding an extra layer of security by forwarding requests to backend Symfony servers. By incorporating NGINX, Symfony applications benefit from optimized request handling, increased scalability, and enhanced security.

2.5 Application Management with Docker

Docker is an open-source platform that enables developers to automate the deployment and management of applications using software containers. Containers are lightweight and self-contained environments that include all the necessary dependencies and configurations to run an application. By packaging an application along with its dependencies, libraries, and configurations into a container, Docker ensures consistent and portable environments that can run on any system.

On the other hand, Docker Compose is a tool (Docker Plugin) that simplifies the management of multi-container Docker applications. It allows developers to define and configure multiple containers, their dependencies, and network connections using a YAML file. With Docker Compose, developers can easily deploy and coordinate the interaction of multiple containers as a unified application. The reason we need multiple containers is because complex web applications use multiple environment entities, and they are best kept separately. For example, the first container will be created for the PHP-FPM client, while the second one would contain the MySQL client and Database.

For PHP applications developed with Symfony, Docker and Docker Compose offer several benefits:

- Docker ensures that the development environment remains consistent across different machines and platforms. By packaging the application and its dependencies into containers, developers can work in an environment that mirrors the production setup, reducing the likelihood of environment-related issues.
- Docker enables developers to specify and manage the dependencies required by their Symfony application. This ensures that the correct versions of PHP, web servers (such as Nginx or Apache), database servers (like MySQL or PostgreSQL), and other necessary services are available and properly configured within the container. It simplifies the setup process and eliminates the need for manual installation and configuration of dependencies.
- Docker allows developers to create reproducible builds of their Symfony application. By defining dependencies and configurations in a Dockerfile, developers ensure that others who build the application using the Docker image will achieve the same results. This promotes collaboration and facilitates sharing the development environment with team members.
- Docker makes it easy to scale Symfony applications by running multiple instances of the application containers. Each container operates independently and can handle requests, providing scalability and improved performance. Additionally, containers provide isolation, so issues in one container do not impact others, enhancing overall application stability.
- Docker and Docker Compose simplify the development workflow for Symfony applications. Developers can quickly set up the development environment with all the necessary services and dependencies by executing a single command. This eliminates the manual setup process and reduces the time spent on environment configuration. Moreover, Docker integrates seamlessly with development tools, version control systems, and CI/CD pipelines, enhancing productivity and facilitating efficient development practices.

To sum up, Docker and Docker Compose are valuable tools for PHP application development with Symfony. They offer consistent, reproducible, and isolated environments, simplify dependency management, support scalability, and streamline the development workflow. These tools enhance productivity, promote collaboration, and provide an overall improved development experience when working with Symfony applications, which made them an essential part of the development stack of Cloud-Vitae.

2.6 Database management with MySQL

A database is an organized and stored collection of data in a computer system that allows for efficient management, retrieval, and manipulation of information. It is extensively utilized in software development to store and manage application data persistently.

MySQL is a renowned open-source relational database management system (RDBMS) that is widely employed in Symfony PHP applications. It is the number one choice in web development due to its performance, reliability, and user-friendly nature. MySQL follows the relational database model, where data is organized into tables with rows and columns, and it supports SQL (Structured Query Language) for querying and manipulating the data. [11]

Symfony and PHP applications often utilize MySQL for several reasons:

- MySQL provides a robust and efficient solution for storing relational data. It enables developers to define tables with structured schemas, establish relationships between tables, and perform complex queries to retrieve and manipulate data. This makes it suitable for applications with intricate data structures and relationships.
- MySQL is renowned for its scalability and performance capabilities. It can handle high volumes of concurrent requests, making it well-suited for applications that experience heavy traffic and require quick response times. Additionally, MySQL offers various optimization techniques and indexing mechanisms to enhance query performance and overall application speed.
- Symfony has built-in support for MySQL through Doctrine, its database abstraction layer. Doctrine provides an Object-Relational Mapping system (chapter 2.2.2) that enables developers to interact with the database using PHP objects instead of writing raw SQL queries. This simplifies database operations, improves code maintainability, and ensures compatibility between the Symfony framework and MySQL.
- MySQL benefits from a large and active community of developers and users, resulting in abundant documentation, tutorials, and resources. It also integrates well with Symfony through various third-party tools, libraries, and frameworks, further enhancing development productivity and efficiency.
- MySQL has a proven track record of stability and reliability. It has been extensively used in production environments for many years, powering numerous websites, applications, and enterprise systems. Its maturity and robustness make it a trusted choice for applications like Cloud-Vitae that require a stable and dependable database solution.

2.7 CSS and Styling Components

CSS (Cascading Style Sheets) is a language used to determine the visual appearance of HTML documents. It specifies how elements on a webpage should be displayed, including properties like layout, colors, fonts, and spacing. By separating the content and structure from the design, CSS allows for consistent styling across multiple pages.

In the Symfony framework, Webpack Encore (chapter 2.2.3) is a library that handles frontend assets, including CSS files. It streamlines the process of bundling, optimizing, and integrating CSS files into Symfony applications. With Webpack Encore, developers can organize and incorporate their CSS files into their Symfony projects. CSS files are treated as assets, and Webpack Encore provides a simple API for configuring how they are handled. Developers can define entry points for CSS files, specify dependencies, and utilize various loaders and plugins to process and transform the CSS. During the build process, Webpack Encore integrates CSS with other frontend assets like JavaScript (chapter 2.8) and images. It employs loaders to process CSS files and their dependencies, such as preprocessors like Sass or Less. These loaders transform the CSS code, enabling advanced features like nesting and variables. Additionally, Webpack Encore applies optimizations like minification and concatenation to reduce file size and improve performance.

Once the build process is complete, Webpack Encore generates optimized CSS files that can be included in the Symfony application. These CSS files are referenced and utilized in Twig templates or other parts of the application using Symfony's asset management system. Webpack Encore supports CSS modules, allowing developers to scope CSS styles to specific components or elements. This helps prevent conflicts and promotes modular development practices.

2.8 JavaScript and Asynchronous Server Communication with AXIOS

JavaScript (JS) is a widely used programming language that enables developers to create interactive and dynamic features on web pages. It allows for the manipulation of webpage content, responsiveness to user actions, and asynchronous communication with servers to retrieve or send data.

In the realm of Symfony web applications, Axios is a popular JavaScript library used to facilitate HTTP requests from the client-side to the server-side. It simplifies the process of exchanging data with the server using asynchronous requests.

Cloud-Vitae employes Axios for the following reasons:

- Symfony often necessitates retrieving data from the server without refreshing the entire webpage (SPA, chapter 1.3.2.3). Axios permits developers to make asynchronous requests to Symfony endpoints, enabling data retrieval in the background. This approach supports dynamic updates and delivers a smoother user experience.
- Symfony applications frequently rely on RESTful APIs to interact with external services or exchange data between the client and server. Axios offers a convenient and adaptable solution for communicating with these APIs by facilitating HTTP requests (such as GET, POST, PUT, DELETE) and managing the resulting data.
- Axios streamlines error handling by providing built-in mechanisms to intercept and manage errors that may occur during HTTP requests. It allows developers to establish global error handling or customize error handling on a per-request basis, thereby enhancing the resilience and dependability of Symfony applications.
- Axios employs JavaScript Promises, which are objects representing the eventual completion or failure of an asynchronous operation. Promises simplify the management of asynchronous tasks, enabling developers to chain multiple requests or execute actions based on the success or failure of a request.
- Axios is designed to operate consistently across different web browsers, ensuring compatibility and reliable performance for Symfony applications.

By leveraging Axios and Symfony Framework, Cloud-Vitae gains the ability to effectively manage and control data communication between the client-side JavaScript code and the server-side PHP code. Axios offers a user-friendly and efficient approach to making HTTP requests, handling responses, and seamlessly integrating with RESTful APIs. Overall, Axios enhances the functionality and interactivity of Symfony web applications by facilitating seamless communication between the client and server components.

2.9 Software Versioning with GitHub

Upon acquiring these tools, the application development process can begin, with a focus on implementing each functionality incrementally. The ongoing and active communication with the client necessitates the maintenance of a continuously functional version of the application. Furthermore, it is crucial to consider the potential for future enhancements and revisions, ensuring that the application's structure is adaptable to accommodate these changes. Embracing a versioning approach offers significant advantages. For instance, while the latest version is being tested by clients, the development team can simultaneously work on the subsequent version, addressing identified issues and introducing new functionalities. Software versioning has emerged as a prevalent practice within the industry, and its value is evident, even when the development team comprises a sole individual.

To accomplish this objective, developers rely on Version Control Systems (VCS) technologies. Throughout the years, numerous VCS systems have been developed, but the predominant choice in contemporary software development is GIT, which is utilized for applications across various project types, encompassing personal, open-source, as well as large-scale endeavors. The fundamental concept in VCS revolves around the repository, which serves as the centralized storage location for all project files. In addition to serving as a storage solution, the repository facilitates comprehensive historical tracking of previous versions. This historical insight contains details such as the specific code updates made by individual developers, the ability to revert to earlier versions, and the creation of branches to enable parallel development that can subsequently be merged.

Version Control Systems (VCS) are categorized into two distinct types: Centralized Version Control Systems (CVCS) and Distributed Version Control Systems (DVCS). CVCS examples include SVN CVS, whereas DVCS examples include GIT and Mercurial. The primary distinction between CVCS and DVCS lies in the centralization versus distribution of the repository. CVCS relies on a central server for version control operations, while DVCS allows each developer to possess their own copy of the repository, providing enhanced autonomy and flexibility. DVCS systems typically offer advanced branching and merging capabilities, facilitating more intricate development workflows. [12]

In the development of Cloud-Vitae, a DVCS system, specifically GIT, was employed. GitHub is a platform that integrates GIT, a free and open-source program initially introduced in the Linux Kernel. GIT is implemented in C to ensure robust performance. It offers a Command Line Interface (CLI) by default, allowing the execution of specific tasks through commands. Alternatively, GIT can be utilized through various integrated tools. GitHub was chosen for its accessibility, widespread familiarity, and user-friendly nature, which proved essential for our test users. Additionally, considering the importance of VCS in the development process, PHP Storm was utilized as the Integrated Development Environment (IDE), since it incorporates GIT as a VCS by default.

3. Architectural Patterns, OOP and Entity Management

Architectural patterns in programming are design principles or templates that offer solutions to common software development problems. They assist developers in organizing and structuring code to create robust, maintainable, and scalable software systems. These patterns define how different components of an application interact, their responsibilities, and relationships. Architectural patterns provide a framework for designing the overall structure of an application, including its layers, modules, and components. They address problems like separation of concerns, scalability, flexibility, code reusability, and testability. By following these patterns, developers can build software systems that are easier to comprehend, modify, and maintain.

Some commonly used architectural patterns include:

- MVC (Model View Controller) separates an application into three components - the model for data and business logic, the view for presentation and user interface, and the controller for managing data flow between the model and view. MVC promotes modularity and separation of concerns.
- Layered architecture divides an application into multiple layers, each responsible for specific functionalities. Typically, these layers include presentation, business logic, and data access layers. Layered architecture supports modularity, maintainability, and abstraction.
- Microservices architecture structures an application as a collection of small, independent services that communicate via APIs. Each service focuses on a specific business capability and can be developed, deployed, and scaled independently. Microservices promote flexibility, fault isolation, and scalability.
- Event-driven architecture relies on events and event handlers for communication between components. It allows for asynchronous and loosely coupled interactions by producing and consuming events. Event-driven architecture offers scalability, responsiveness, and extensibility.
- DDD (Domain Driven Design) emphasizes building software systems that align closely with the problem domain. It involves modeling complex domains and implementing intuitive business logic that reflects the problem domain.

3.1 Symfony 6 Framework Architecture

Symfony 6 adopts the Model View Controller (MVC) architectural pattern, which partitions an application into three primary components: Model, View, and Controller. The Model handles the data management and business logic, while the View is responsible for rendering the presentation layer. Acting as a mediator, the Controller manages user input and orchestrates the interaction between the Model and the View.

Regarding file organization, Symfony 6 employs bundles to structure the codebase, representing distinct application features or functionalities. Each bundle encompasses dedicated directories for Controllers (found under “app/src/Controller”), Views (named templates, found under “app/templates”), and Models (I.e. entities, found under “app/src/Entity”). This approach fosters a clear segregation of responsibilities and encourages modular development practices.

Symfony 6's file system encompasses directories such as configuration, public assets, templates, and source code. The configuration directory houses assorted configuration files, while public assets like CSS and JavaScript are typically stored within the public directory. However, this isn’t the case for Cloud-Vitae since it uses Webpack Encore to dynamically load such assets, which can be found outside the public directory (under “app/assets”). The templates directory contains Twig templates, which generate the application's output. Finally, the source code directory contains bundle-specific directories encompassing Controllers, Models, and related files (found under “app/src”).

The utilization of the MVC architecture in Symfony 6 ensures a distinct separation of data, logic, and presentation, facilitating easier maintenance and promoting code reusability. The well-structured file system complements this architecture, offering an organized framework that empowers developers to work efficiently and collaborate effectively.

3.2 Model View Controller Architecture

The MVC (Model-View-Controller) architecture is a widely used design pattern in programming that helps structure and organize code by separating different concerns within an application. It divides the application into three key components: the Model, the View, and the Controller.

The MVC architecture is favored in Symfony and other frameworks due to its ability to promote separation of concerns, modularity, and maintainability. By dividing code into distinct components, developers can more effectively organize their codebase. This separation facilitates easier code maintenance, testing, and reuse. Symfony utilizes the MVC architecture to structure its framework and encourages developers to follow this pattern when building Symfony applications. By adopting MVC, Symfony promotes consistent code organization and facilitates collaboration among developers working on Symfony projects. The clear separation of concerns inherent in MVC aligns with Symfony's philosophy of decoupled components, making it a natural choice for the framework.

In summary, the MVC architecture is a widely employed design pattern that divides an application into Model, View, and Controller components. It is embraced in Symfony and similar frameworks due to its capacity to provide a well-structured and organized codebase, promote code reusability and maintainability, and align with Symfony's architectural principles.

3.2.1 The Model component

The Model represents the data and business logic of the application. It encapsulates the data structures, performs operations on the data, and interacts with data sources like databases. In Symfony, entities or data models are commonly used to implement the Model. Cloud-Vitae uses a very well-structured entity scheme inspired by real-life concepts which greatly contributes to the model component.

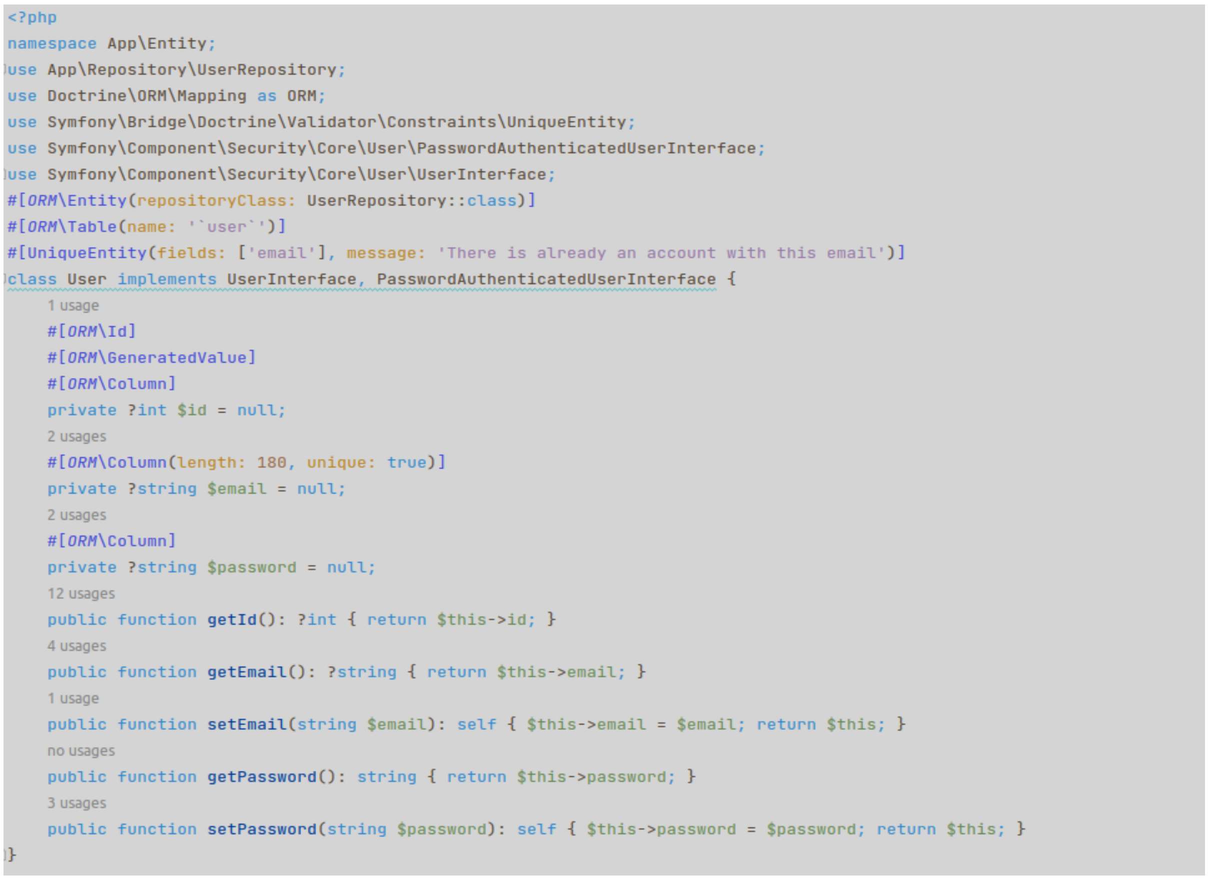
The Model component in this application is responsible for encapsulating and managing the CV data and implementing the necessary operations and rules associated with CV creation, management and sharing. It is also responsible for defining the data structure and relationships, as well as implementing the business rules and operations associated, while managing the storage and retrieval of CV data in the database. It is essentially a component that materializes the object-oriented doctrine (chapter 3.3).

One of the most important attributes of the Model component is creating, maintaining and updating entities, which are object representations defined by classes that hold variables capable of storing values for fundamental data objects. These data objects encapsulate crucial structures of application data. For example, Cloud-Vitae needs a User Entity, since that is the target audience for the application.

An Entity class in Symfony is a simple PHP file that defines how multiple entries of information grouped by attributes should act in our application. It contains a namespace (a piece of code that tells the application where it is compiling from), multiple use statements (lines of code that individually point to another PHP Class that contains already implemented functionalities that we want to use in our Controller, such as the Doctrine's ORM or the Entity's Repository which is another piece of the Model component), one or more annotated variables, each representing a real-life attribute of our Entity, and specific entity methods such as getters and setters (necessary methods for retrieving or altering data encapsulated in a class).

For classic Entities such as User, Symfony already has pre-defined interfaces that can be implemented (for User Entity, the User class implements the "UserInterface" interface, and if we need password authentication for the User, it also implements the "PasswordAuthenticatedUserInterface" interface).

When creating Entities in Symfony, it is also standard to use Doctrine's ORM Annotations. This way of writing code allows us to specify to the application more detailed information on how the Entity should be mapped to the database client. With all this information, we can already create a basic User Entity implementing email and password.



```php
<?php
namespace App\Entity;
use App\Repository\UserRepository;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;
use Symfony\Component\Security\Core\User\UserInterface;
#[ORM\Entity(repositoryClass: UserRepository::class)]
#[ORM\Table(name: '`user`')]
#[UniqueEntity(fields: ['email'], message: 'There is already an account with this email')]
class User implements UserInterface, PasswordAuthenticatedUserInterface {
    1 usage
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;
    2 usages
    #[ORM\Column(length: 180, unique: true)]
    private ?string $email = null;
    2 usages
    #[ORM\Column]
    private ?string $password = null;
    12 usages
    public function getId(): ?int { return $this->id; }
    4 usages
    public function getEmail(): ?string { return $this->email; }
    1 usage
    public function setEmail(string $email): self { $this->email = $email; return $this; }
    no usages
    public function getPassword(): string { return $this->password; }
    3 usages
    public function setPassword(string $password): self { $this->password = $password; return $this; }
}
```

3.2.2 The View component

The View is responsible for the presentation layer of the application. Its role is to display the user interface and render the data provided by the Model. In Symfony, the View is typically created using Twig templates, a powerful and flexible templating engine (chapter 2.2.4). In general, the application has its views split into three categories.

The first category is composed by the entry routes, meaning the pages the user interacts with before creating an account or while creating an account, such as the landing page, login and register pages, the LikedIn sign in page and the creation page.

The second view category contains all the pages of the application that are responsible for rendering the CV. This is the part of the application that imitates a SPA (Single Page Application, chapter 1.3.2.3), even though each page is contained in its own file. This is achieved by implementing a fast and seamless re-render of the page each time a heavier request is made by the user, such as changing the CV template.

The last view layer consists of general items, meaning the items that are rendered the most in the application. To keep this concept consistent in production, Cloud-Vitae stores all the general files in a subdirectory names “general”, and such a directory can be found on all levels of the application, be it in the assets section (CSS files, JS files and other assets) or the template section (Twig files and Email templates). In order to be flagged as general, a file has to either contain information that is rendered on multiple pages (such as a menu that appears on multiple pages) or to have rules used on multiple pages (such as a common style sheet that tells the application how to background should look on multiple pages).

This layered approach is not a common one, and it is not enforced by the view component. By today’s standard it is perfectly normal to have all the views in the same category, but since Cloud-Vitae is an application that expects to handle extremely large chunks of data, this separation seemed necessary to further improve caching systems and request efficiency on the server.

3.2.3 The Controller component

The Controller acts as an intermediary between the Model and the View. It handles user input, triggers relevant actions based on that input, and communicates with the Model to retrieve or update data. It also interacts with the View to render the appropriate response. In Symfony, Controllers are implemented as PHP classes that manage the routing and logic of the application. Cloud-Vitae contains a series of controllers which make the application run smoothly.

A Symfony Controller is nothing more than a PHP Class that implements Symfony’s pre-defined “AbstractController” Interface. It contains a namespace (a piece of code that tells the application where it is compiling from), multiple use statements (lines of code that individually point to another PHP Class that contains already implemented functionalities that we want to use in our Controller, such as the Doctrine’s “EntityManagerInterface” interface or Symfony’s “HTTP\Request”), and one or more methods, each representing a route for the application.

We can read, write and interpret such methods by using PHP’s annotation System:

```
#[Route('/user/{id}', name: 'app_user_profile', methods: ['GET', 'POST'])]
```

This code writing method allows us to specify and define to the application specific attributes of a route, such as its source and possible request parameters, a name, allowed request methods and many others, without having to pass any additional information to the method. This is an industry standard for Symfony (the annotation syntax changed a bit over major releases, but it is still used on all versions), and Cloud-Vitae's Controllers strictly respect these rules.

A controller method is defined by its name, parameters (which must contain but are not limited to request parameters specified in the annotation), and a return type. To avoid errors, the name of the method just has to be unique in the controller it is located in. But in order to keep an even standard across the applications controllers, Cloud-Vitae matches the method names to the route names as follows:

- When there is a route name available (I.e. “app_user_profile”), the method has the same name but respecting PHP’s typing standars (I.e. “appUserProfile()”), and in the extreme cases where a separate method is needed for each request type (I.e. separate methods for GET and POST), the methods name contains the request type at the end (I.e. “appUserProfileGET()”, “appUserProfilePOST()”).
- When there is no route name available (the name attribute is an optional one in Symfony, and you only need to name routes when you have to specifically call them by name from outside that specific Controller, which in almost all cases is a call to a render return method, respectively a GET request; so for POST requests that have no View component, there will never be a need to be addressed by name), the methods are named by what they do in the most specific but clear way possible. For example, we have a route that accepts requests for changing a user’s profile picture.The annotation defines no name attribute, but we have enough information already to write the methods name as “appUserChangeProfilePicturePOST(int $user_id)”.

```
#[Route('/user/changeProfilePicture{user_id}', methods: ['POST'])]
```

The last defining factor of a controller method is its return type. This is a clean way of telling the Controller component beforehand what it should do after processing a specific request. There are many options, but the most used ones in Cloud-Vitae are returning a render (it renders the specified view for the user after processing its request), returning a redirect (it redirects the user to the specified route after processing its request), returning a JsonResponse (this return is designed for asynchronous requests that happen without the user seeing them, and returns the response information without refreshing the page) and so on. All of the above are a response return type, and they are defined in a method as follows: “appUserChangeProfilePicturePOST(int $user_id): Response”.

After we defined the route annotations, method name and response types, the only thing left is writing the body of the method. This is where the actual request processing happens, and we might need to add other method parameters over time in order to achieve the route’s final goal. For example, we might want to create a user register route.

```
#[Route('/register ', name: 'app_register', methods: ['GET','POST'])]
public function register(): Response {}
```

In this case, the Controller component has to directly interact with the Model component, since we must create a user entry in the database. For this, Doctrine already provides us the easy to use “EntityManagerInterface”, which allows us to interact with and manipulate entity structures from the Model component.

```
#[Route('/register ', name: 'app_register', methods: ['GET','POST'])]
public function register(EntityManagerInterface $entityManager): Response {}
```

Since the user has to complete a form and then submit the data in a POST request, we want to also call the Request component as a method parameter to be able to access this data.

```
#[Route('/register ', name: 'app_register', methods: ['GET','POST'])]
public function register(EntityManagerInterface $entityManager, Request $request): Response {}
```

Finally, we will add the processing code and return the desired response, in our case a redirect to the home page of the application after registering.

```
#[Route('/register ', name: 'app_register', methods: ['GET','POST'])]
public function register(EntityManagerInterface $entityManager, Request $request): Response {
    //processing the register request
    return $this->redirectToRoute( route: 'app_home');
}
```

These are of course just the basics of creating a Controller, and for a complete overview of the register method explained here it is recommended to take a look at the complete code, which can be found under “app/src/Controller/RegistrationController.php”.

3.3 Object Oriented Programming

Object Oriented Programming (OOP) emerged in the 1960s and underwent further development in the 1970s and 1980s through the efforts of various computer scientists and researchers. The core ideas behind OOP drew inspiration from earlier programming languages and concepts. However, the actual term "Object Oriented Programming" was coined by Alan Kay and his team at Xerox Palo Alto Research Center (PARC, Silicon Valley, California, USA) during the early 1970s.

Alan Kay, a notable computer scientist and OOP pioneer, played a crucial role in advancing OOP concepts and creating programming languages that supported them. His work was notably influenced by Simula, a programming language developed in the 1960s. Simula introduced the concept of classes and objects, establishing the foundation for OOP. It also introduced class inheritance, enabling the inheritance of properties and behaviors between classes and forming an object hierarchy. [13]

Building upon Simula's concepts, Alan Kay and his colleagues at Xerox PARC developed Smalltalk, which is considered one of the earliest fully object-oriented programming languages. Smalltalk revolutionized software development by emphasizing the use of objects as the fundamental building blocks. It popularized key OOP concepts like encapsulation, inheritance, and polymorphism.

Object-Oriented Programming (OOP) is a programming paradigm that revolves around objects, which are instances of classes. OOP emphasizes the organization of code based on these objects, which combine both data and behavior. It provides a structured approach to software development, simplifying the management of complexity, promoting code reusability, and enhancing maintainability. It is widely adopted in programming languages like Java, C++, Python, PHP, and many others. It follows several fundamental principles such as:

- Encapsulation, which involves grouping data and related methods or functions into a single unit, known as an object. This promotes data privacy and allows for better code organization and modularity.
- Inheritance, which enables classes to inherit properties and methods from other classes, establishing a hierarchical relationship. This facilitates code reuse and facilitates the creation of specialized classes based on existing ones.
- Polymorphism, which allows objects of different classes to be treated as objects of a common superclass. It provides flexibility by using a single interface to represent multiple types, enhancing code extensibility.
- Abstraction, which focuses on representing essential features of an object while hiding unnecessary details. It simplifies complex systems by offering a high-level view and exposing only relevant information.

3.3.1 Object Oriented Programming in PHP

PHP is widely used for building robust and maintainable applications through the implementation of Object-Oriented Programming (OOP). PHP provides extensive support for OOP concepts, equipping developers with a diverse range of features and syntax to effectively utilize them and bringing several benefits.

Using OOP in PHP, code organization becomes more manageable as developers can structure their code into classes, serving as templates for creating objects. These classes encapsulate related data and functions, facilitating better code organization and promoting modular development. Consequently, code becomes easier to understand, maintain, and reuse. OOP also enables the creation of reusable classes, allowing the instantiation of multiple objects. This promotes code reuse across different sections of an application or even across various projects, resulting in saved development time and effort.

PHP's OOP features, including access modifiers (public, private, protected), support encapsulation. Encapsulation ensures that the inner workings and data of a class are shielded from external access, enhancing data security and preventing unauthorized modifications. PHP's OOP also allows classes to inherit properties and methods from other classes, establishing a hierarchical relationship (Inheritance principle). PHP facilitates class extension using the "extends" keyword, enabling code reuse and creation of specialized classes based on existing ones. Inheritance also facilitates the implementation of polymorphism, which in PHP allows objects of different classes to be treated as objects of a common superclass or interface. This flexibility empowers developers to write more versatile code that can work with various object types. Polymorphism simplifies code maintenance and enhances the flexibility and extensibility of the application.

PHP boasts a variety of frameworks, such as Symfony, Laravel, and Yii, which heavily embrace OOP principles. These frameworks provide pre-built components and adhere to OOP patterns, simplifying the development of complex web applications. OOP aligns well with the architectural and design patterns commonly utilized in popular PHP frameworks.

In conclusion, employing OOP in PHP brings advantages in terms of code organization, reusability, encapsulation, and flexibility. It enables the creation of modular and maintainable applications, reduces code duplication, and enhances code readability. Embracing OOP in PHP promotes a structured approach to development and fosters collaboration among developers, making it highly advantageous for projects of varying sizes.

3.3.2 Object Oriented Programming in Symfony and MVC architectures

Object Oriented Programming plays a vital role in Symfony application development, as the framework is built upon OOP principles and encourages their use. As previously stated, Symfony implements the Model View Controller (MVC) architecture, a widely adopted design pattern. The MVC pattern divides an application into three components: Model, View, and Controller, but OOP concepts are also greatly utilized to implement each component effectively.

In Symfony, the Model component represents entities or data models, defined as classes. These classes encapsulate data structures and business logic. OOP enables developers to define entity classes with properties and methods that define data behavior and interactions. Symfony's Doctrine ORM provides tools for mapping entities to databases, ensuring seamless data persistence and retrieval.

The View component in Symfony handles the presentation layer using Twig templates. OOP principles are employed to organize templates into reusable components, facilitating easier maintenance and modification of UI elements. Twig templates leverage inheritance and composition, enabling the creation of modular and reusable views.

Controllers act as intermediaries between the Model and the View. They are implemented as PHP classes in Symfony, allowing encapsulation of related actions and behaviors. OOP principles guide the definition of Controller classes, which interact with the Model to retrieve or update data and communicate with the View to render appropriate responses. Symfony's routing system maps requests to specific Controller actions based on predefined routes.

OOP in Symfony promotes code organization, reusability, and maintainability. Encapsulation, inheritance, and polymorphism enable the creation of modular and extensible components. This adherence to OOP contributes to the overall MVC architecture of Symfony, ensuring a clear separation of concerns and facilitating the development of scalable and maintainable applications.

Additionally, OOP in Symfony integrates well with other features and tools like dependency injection and event systems. It enables the adoption of design patterns and architectural principles aligned with MVC, such as service-oriented architecture and domain-driven design. Embracing OOP fosters a structured and consistent development approach in Symfony, enhancing collaboration among developers and facilitating better understanding of Symfony projects.

3.4 Entity Management

In programming, entity management refers to the management and manipulation of entities within a software application. An entity represents a specific and identifiable object or concept that is relevant to the application's domain. For instance, entities can include users, products, orders, or other real-world entities necessary for the application's functionality. Even though this topic has been in some sense already approached in this thesis, it was only from the perspective of the Entities that constitute the Model component of an MVC framework. However, entity management is a complex part of programming, that resumes to much more than just implementing an MVC component. They represent an essential step in building a modern OOP application such as Cloud-Vitae.

Entity management encompasses tasks such as creating, reading, updating, and deleting entities, commonly known as CRUD operations. These operations enable the management of entity data and the execution of business logic associated with the entities. Entity management often goes hand in hand with database management since entities are frequently stored and retrieved from a database. The persistence layer of an application, responsible for database interactions, plays a crucial role in entity management. It provides functionality for mapping entities to database tables, handling data retrieval and storage, and facilitating querying and manipulation of entity data.

To implement entity management, techniques such as Object-Relational Mapping (ORM) are commonly used. ORM allows entities to be represented as objects in the programming language and mapped to corresponding database tables. This approach simplifies database operations by leveraging object-oriented programming techniques and abstracting the complexities of direct database access.

In summary, entity management revolves around the handling and manipulation of entities in a software application. It involves operations like creating, reading, updating, and deleting entities, often with the assistance of a persistence layer and database management techniques like ORM. Effective entity management is essential for developing applications capable of efficiently and accurately processing domain-specific data.

3.4.1 Entity Relationship Diagrams

An Entity-Relationship Diagram (ERD) is a visual representation that depicts the entities, attributes, and relationships within a system or database. It is commonly used in software engineering and database design to model data structure and behavior in a clear and organized way.

To create an ERD for modern web applications, we start by identifying the main entities or objects that are relevant to our web application. These entities represent the major components or concepts that need to be stored and managed in the system. For example, in an e-commerce application, entities could include customers, products, orders, and reviews. After that we will determine the attributes or properties that describe and characterize each entity. These attributes represent the specific data elements associated with each entity. For instance, a customer entity may have attributes such as name, email, and address, while a product entity may have attributes such as name, price, and description.

It il also crucial to identify the relationships between entities. Relationships define how entities are connected or associated with each other. Common relationship types include one-to-one, one-to-many, and many-to-many. For example, in an e-commerce application, there would likely be a one-to-many relationship between customers and orders, as a customer can have multiple orders, but a specific order can only have one customer.

Cardinality refers to the number of instances of one entity that can be associated with another entity. Modality specifies whether the relationship is mandatory (denoted by a solid line) or optional (denoted by a dashed line). Cardinality and modality help define the nature of the relationship between entities. Both of these are great helpers in building and interpreting an ERD. It is also extremely helpful to utilize specialized diagramming tools or software that support entity-relationship modeling to create the actual ERD. These tools provide user-friendly interfaces for drawing entities, attributes, relationships, and other elements. They may also offer features to automatically generate SQL scripts or database schemas based on the ERD. However, they key to a great ERD lies in reviewing and refining to ensure accuracy and completeness, revisiting the entities, attributes, and relationships to align them with the requirements of our web application, and iteratively refining the ERD until it accurately represents the data structures.

3.4.2 Cloud-Vitae's ERD and Entity Schema

For Cloud-Vitae, creating the Entity Schema was a straightforward process. Since a CV document is oriented around the user and the user only, we had to split the user's information in two categories to avoid redundancy or hard to maintain code and logic.

The first category contains all the information about the user that cannot have multiple entries, generally personal information, such as first and last name, city and country of residence, profile picture and some logical flags. This is information that can be changed or replaced at any given time by the user but cannot have more than one instance. For example, a user cannot have multiple email addresses associated.

The second category encompasses all the information that the user can add multiple entries for and is separated as different entities based on their nature. Examples of this would be languages spoken by the user, work experience and so on. As we can see (in the fig.), each user can have one or more of these items individually, and they are grouped as separate entities. The work experience is its own entity, so are the spoken languages, and so is the education, since they all have different attributes and represent different real-life scenarios.

This concept has already been explored in the programming world in the topic of database relations. Database relations establish connections between tables in a relational database management system (RDBMS). One common type of relation is the "one-to-one" relationship.

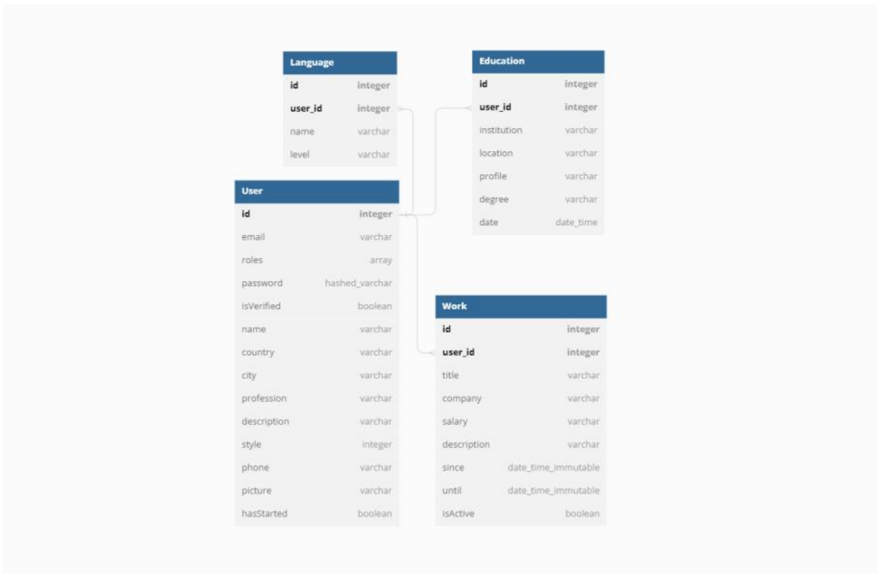
In a one-to-one relationship, one record in a table is associated with only one record in another table, and vice versa. Each record in one table corresponds to a single record in the related table. This type of relationship is useful when two entities have a unique and exclusive connection. For example, in the Cloud-Vitae database if we were to make a table just with email addresses, the relation between users and emails would be one to one.

In a many-to-many relationship, multiple records in one table are associated with multiple records in another table. This relationship is implemented using an intermediate table, often referred to as a junction table or mapping table. The junction table holds the combinations of related records from both tables. For example, in the Cloud-Vitae database if we were to make a table dedicated to University Diplomas (such as Bachelor’s, Master’s), the relation between this table and the users would be many-to-many.

A one-to-many relationship involves one record in a table being associated with multiple records in another table. This relationship is the one used in Cloud-Vitae between the user entity and all other entities. For example, in the Language table, the relation between users and languages is one-to-many, since one user can have multiple languages associated, but each language has only one user.

A many-to-one relationship is the inverse of a one-to-many relationship. It occurs when multiple records in one table are associated with a single record in another table. This relationship is often used when multiple entities relate to a common entity. For example, we can see the previously discussed one-to-many relation as a many-to-one relation, if we inverse the perspective and look from the languages table. Its relationship with the user table is many-to-one.

A self-referential relationship occurs when records in a single table are related to other records within the same table. This type of relationship is useful when modeling hierarchical or recursive data structures. For instance, in a database for employees, each employee record may have a supervisor field that references another employee within the same table. Cloud-Vitae does not have and could not have such a relationship, since in this ecosystem a user and its CV are completely unrelated and irrelevant to other users and their information.



4. Application Security

Security in a web application involves implementing measures and practices to safeguard the application and its users against unauthorized access, data breaches, and malicious attacks. It encompasses protecting sensitive data, ensuring user privacy, and preventing unauthorized manipulation or misuse of the application. During the development of Cloud-Vitae, the largest amount of time was allocated to engineering and implementing the best security patterns, since the application will process and handle large amounts of private and extremely sensitive and identifiable personal information.

The security implementations of Symfony have been a big help in building the most secure ecosystem possible for Cloud-Vitae as a skeleton for most of the industry standard requirements in this field has been already provided by the framework.

4.1 Authentication

This process verifies the identity of users accessing the application, typically involving usernames, passwords, and additional factors like two-factor authentication or biometrics. Strong authentication mechanisms ensure that only authorized users can access the application.

Cloud-Vitae implements a strong and refined authentication system, in which the user is required to register an account before being allowed to interact with the application.

4.1.1 Email Registration and Authentication

The registration and authentication process of this application is split in two possible actions, the first one being by email. When the user decides to create an account via email in order to use the application, he will be redirected to the register route (/register). Here a registration form is located, where the user is required to input his personal information such as email address, a password for his account and to agree to the terms and conditions of usage (chapter 4.10.x). After a successful registration, the application automatically sends a verification email to the registered email address. The user must follow the link contained in this email in order to validate the account and be authorized to use the application further. There is a flag on the database that monitors this specific action. Until the user verifies its email address with this action, all the applications pages that implement CRUD (create, read, update, delete) functionalities will render an error page explaining why the user is not authorized to view this page, and showing the option to resend the verification email if needed. The Controller component is responsible for this, and it carefully checks the database flag first for every request the user makes. If it is not verified, the user is immediately denied access before any other command is executed by the controller.

4.1.2 LinkedIn Registration and Authentication

The second option regarding user registration and authentication is composed by the applications link to the LinkedIn API. Third-party authentication has become a widely used functionality on most web applications today, due to its high level of security and a simpler process for the users and developers to follow. In Cloud-Vitae's development there were two platform candidates for such a system, respectively Google and LinkedIn. Despite the fact that Google has a much higher user base, LinkedIn was the desired choice since its domain of interest and action is the same one as Cloud-Vitae's.

4.1.2.1 Application Programming Interface

An API, which stands for Application Programming Interface, serves as a set of guidelines, protocols, and tools that facilitate communication and interaction between different software applications. It establishes the methods and data formats that applications can utilize to request or exchange information and carry out specific tasks.

By acting as an intermediary, an API allows applications to access the services and functionality provided by other applications, systems, or platforms in a standardized and controlled manner. It abstracts away the complexities of the underlying implementation and presents developers with a simplified interface to work with.

APIs can be classified into several types based on their purpose and functionality:

- Web APIs are specifically designed for web applications and enable access to resources and services over the internet. They typically employ HTTP protocols and support various data formats such as JSON or XML for seamless data exchange. Web APIs empower developers to integrate third-party services, retrieve data from remote servers, or perform actions on external systems.
- Library or Framework APIs are provided by software libraries or frameworks and offer pre-built functions and modules that simplify application development. Library APIs provide access to reusable code components, while framework APIs define the structure and conventions for constructing applications within a specific framework.
- Operating system APIs are made available by operating systems to enable applications to interact with system resources and services. They provide functions for tasks such as file management, network communication, device access, and process management. Operating system APIs are platform-specific, enabling developers to create applications that can run on a particular operating system.
- Database APIs facilitate application interaction with databases, allowing operations such as storing, retrieving, updating, and deleting data. They establish a standardized approach for connecting to databases and executing queries or commands. Database APIs can be specific to a particular database management system or adhere to industry-standard protocols such as SQL for relational databases.

APIs play a vital role in enabling integration, interoperability, and extensibility among various software systems. They foster code reusability, simplify development efforts, and promote collaboration by enabling developers to leverage the functionality and services offered by other applications or platforms. APIs have been instrumental in driving the growth of modern web applications, mobile apps, and the interconnectedness of diverse digital services and systems. Since Cloud-Vitae is a web application, the choice of a Web API was obvious, and the one used is LinkedIn’s API as it has a large user base, extremely useful tools for user registration and authentication and an easy to set up connection with the application.

4.1.2.2 LinkedIn’s API

LinkedIn offers an API (Application Programming Interface) that provides developers with tools, protocols, and endpoints to interact with LinkedIn's platform and access its features and data. This API allows developers to integrate LinkedIn functionalities into their own applications, websites, or services. LinkedIn provides several APIs that serve different purposes:

- LinkedIn OAuth 2.0 API facilitates authentication and authorization services. Developers can use it to implement LinkedIn's authentication system, allowing users to log in to their applications using their LinkedIn credentials.
- With LinkedIn Share API developers can incorporate LinkedIn sharing capabilities into their applications. It enables users to share content, such as articles, blog posts, or job listings, from within the application to their LinkedIn profiles or networks.
- LinkedIn Marketing Developer Platform API targets marketers and advertisers. It allows developers to create and manage advertising campaigns on LinkedIn, access campaign analytics, retrieve company and member data for targeting purposes, and more.

- LinkedIn Messaging APIs provide functionality for sending and receiving LinkedIn messages. Developers can integrate messaging features into their applications to enable communication between users through the LinkedIn messaging system.
- The LinkedIn Profile API grants developers access to LinkedIn user profiles and profile data. It enables retrieval of public profile information, including names, profile pictures, headlines, work experience, education, and other professional details.

For Cloud-Vitae's purpose the LinkedIn OAuth 2.0 API and the LinkedIn Profile API are the most fitting and were the ones chosen for development, since the application needed above all a third-party authentication system and a secure and reliable way to extract and incorporate pre-existing user data. Before Cloud-Vitae finished its initial development stage, LinkedIn unfortunately revoked most of the rights from the LinkedIn Profile API, and now complex user information such as work experience and educations are untouchable for developers. This was a big blocking point for the application, but since there is no other major competitor for LinkedIn that offers a similar API, there was no choice but to continue this way.

4.1.2.3 Cloud-Vitae's API Implementation

Using LinkedIn's OAuth 2.0 API and Profile API, users can now securely register and authenticate on Cloud-Vitae. During this process, the user is redirected to the LinkedIn authentication page, where this process is externally managed. After a successful authentication, LinkedIn redirects the user back to Cloud-Vitae along with his information such as email address, name and profile picture. The application searches for a record that matches the user information sent and validated by LinkedIn and if it is unable to find one, it registers and links an account for the user. If such a record is found, the user is authenticated.

Since LinkedIn is a safe and secure platform, it would have been a good approach to also authorize the user automatically when an account is registered trough LinkedIn's API, but in an effort to completely remove the possibility of data breaches and leaks from Cloud-Vitae, the email address provided by LinkedIn also needs to be verified by the user before he is authorized access to pages that implement CRUD (create, read, update, delete) functionalities.

4.2 Authorization

The authorization process determines the actions and resources that a user is permitted to access within the application. It involves assigning appropriate permissions and roles to users based on their authority level, preventing unauthorized access to sensitive functionality or data. Cloud-Vitae's authorization system is a simple one as the general user base of the application is not composed of multiple access groups (except admin or super-admin). The user is authorized to access certain areas of the application based on the level of completion of defined actions he must take.

4.2.1 The User Role System

Symfony offers a versatile and robust user role system as a component of its security functionality. This user role system empowers developers to define and manage roles for users in their applications, dictating their permissions and access privileges and it is implemented in Cloud-Vitae. This user role system is implemented through a combination of the security configuration and the user management system, containing a multi-layered system.

Using role definition, developers can define roles for their application by specifying them in the security configuration. These roles can be given descriptive names like "ROLE_ADMIN" or "ROLE_USER" to represent varying levels of access and privileges. Users in the application can be assigned one or multiple roles. This assignment can occur during user registration or be managed through an administrative interface. Each user can be associated with specific roles or a combination of roles. These two roles are the one used in Cloud-Vitae's configuration, but the "ROLE_ADMIN" can only be attributed manually since at the moment there is only one possible administrator for the application, the developer. The basic user role is dynamically assigned to the user upon registration, as a specified task of the Controller component, specifically the RegistrationController.

Developers can configure access control rules in the security configuration of Symfony to restrict access to specific sections of the application based on user roles. Access control rules determine which roles are required to access particular routes, controllers, or actions. For instance, only users with the "ROLE_ADMIN" role might be granted access to the administrative panel. However, for Cloud-Vitae's access management purpose the email verification restriction system achieves the desired result by itself. In addition to access control, the user role system can be employed for role-based authorization. Developers can utilize user roles in their application's business logic to determine the actions and functionality a user is permitted based on their assigned roles. This can involve limiting access to specific resources or enabling specific operations exclusively for users with specific roles.

Symfony provides helpful methods and components to programmatically verify a user's role(s). Developers can employ these utilities to confirm whether a user possesses a particular role, enabling them to make decisions within their application's code based on the user's role(s). By leveraging Symfony's user role system, Cloud-Vitae established role-based access control and authorization mechanisms within the application. This enables precise control over user permissions, ensuring that users are granted access only to the features and functionality appropriate for their assigned roles.

4.2.1 The Email Verification System

Symfony provides a flexible and secure framework that allows developers to implement an email verification system according to their specific requirements. Even though it is a straightforward process to implement such a system for email-based registration using Symfony, it does not provide any API-specific implementation, so this system has been heavily modified to fit Cloud-Vitae's needs.

To make the registration experience consistent for the user but also to avoid major data leaks, the logic of registering a user is converted and treated in the same manner when the LinkedIn API responds. For both of these registration methods, the email provided must be unique (meaning a user cannot have multiple accounts with the same email address, even if one of them is registered via email and the other via LinkedIn) and must be validated. The user may choose to not follow these guidelines, but the ecosystem operates in such a way that there is no other possible method to be authorized access to Cloud-Vitae.

The email sent to the address registered by the user is also completely unique. The verification link contains a unique key generated by the application, that is attached to the verification request when the user follows the link. The controller component receives this key as a part of the request, and checks if it is legitimate or not. This key is technically impossible to guess, but as spam bots are evolving every year, Cloud-Vitae takes yet another security step to make the authorization process as reliable as possible. The user must be logged in to verify the email address. In fact, it is the only functionality granted until the email is verified. Even if someone is able to guess or discover this secret verification key, it is not enough to simply access the link, as the Controller will instantly deny any request that is not authorized as a veridic logged in request from that specific account.

After the user successfully completes this process, he is flagged in the application as verified and the Controller component can now grant him access on all the other pages each time he makes a request.

4.3 Data Encryption

Encryption encodes data in a way that it becomes unreadable without the proper decryption key. Encrypting sensitive data like passwords or personal information protects it even if an attacker gains access to the data. Cloud-Vitae uses encryption to protect users' passwords, as they are the one single piece of data that is most commonly targeted in cyber-attacks, and it is even potentially more sensitive since most users statistically speaking tend to use the same password for multiple accounts. Therefore, encrypting passwords is not only securing users in the Cloud-Vitae ecosystem, but also all around the internet.

4.3.1 Password Hashing and Verification

In many applications, passwords are utilized for user authentication purposes. To ensure the secure storage of these passwords, it is important to hash them. Symfony's PasswordHasher component offers a comprehensive set of tools and functionalities specifically designed for secure password hashing and verification. By leveraging this component, developers can easily implement robust password hashing techniques, enhancing the overall security of the application and safeguarding user credentials.

To achieve this result, Symfony's PasswordHasher employs Bcrypt hash as a cryptographic hash function widely utilized for securely hashing and storing passwords. It is intentionally computationally intensive, providing robust protection against brute-force attacks and rainbow table attacks. Due to its security features, bcrypt is highly regarded as a reliable option for password hashing in applications. By employing bcrypt, user passwords are safeguarded, significantly reducing the likelihood of attackers obtaining the original passwords from the stored hashes. This reinforces the overall security of the system and ensures the protection of user accounts.

This way, the password never appears as it’s actual string representation, not even in the database. It is stored as a 60-character long string, which contains various characters depending on the chosen cost for the algorithm. The configuration option for bcrypt hashing in Symfony is the cost, represented by an integer ranging from 4 to 31, with a value of 13 for Cloud-Vitae. Increasing the cost value by one doubles the time required to hash a password. This design allows for the flexibility of adapting password strength to potential advancements in computational power, ensuring that the hashing algorithm remains resilient against future improvements in computing capabilities.

Using this algorithm, Cloud-Vitae does not need the plain password when verifying user input either. The application can check whether or not the password given by the user matches the encrypted password in the database. This way the plain password appears and can be seen only in one place, the field in which the use types it.

4.4 Input Validation

A web application should validate and sanitize user inputs to prevent attacks such as SQL injection, cross-site scripting (XSS), or command injection. Input validation ensures that only expected and safe data is accepted, reducing the risk of executing malicious code or commands. Input validation in the case of Cloud-Vitae refers to the process of verifying and sanitizing user input to ensure its integrity, safety, and adherence to pre-defined specific rules and constraints. Symfony provides a range of tools and components that aid in input validation, helping to mitigate security vulnerabilities caused by malicious or incorrect user input.

Developers can define validation rules using Symfony's validation component, specifying constraints on input fields such as required fields, data types, length limitations, format patterns, and more. These rules can be defined using annotations, XML, YAML, or PHP configurations. Once the validation rules are established, they can be applied to user input during form submission or any interaction involving user data. Symfony's form component automatically handles the validation of form fields based on the defined rules.

4.4.1 Raw Data Validation

Symfony offers a wide array of built-in validation constraints that can be applied to fields, including constraints for non-blank fields, length restrictions, email format validation, regular expression matching, and choice selection validation, among others. These constraints ensure that input adheres to specific criteria and meets the desired validation requirements. In cases where user input fails to satisfy the defined validation rules, Symfony automatically generates error messages that can be presented to the user. These error messages provide feedback on which fields are invalid and detail the specific validation constraint that was not met.

Symfony allowed Cloud-Vitae to also develop custom validation constraints when the built-in constraints were insufficient. By extending Symfony's Constraint class and implementing the necessary validation logic, developers can define their own validation constraints to suit specific requirements. Symfony supports the concept of validation groups, which enables developers to define different sets of validation rules for various use cases or scenarios. This capability allows for the validation of specific fields or groups of fields based on the context in which they are utilized. In this application, the most important validation rules applied are for email addresses and passwords. These rules force the user to input valid email addresses and assure bare minimum password security standards (I.e a password has to be at least 6 characters long). In addition, all input fields that appear on Cloud-Vitae are protected against SQL injection.

By incorporating input validation in Symfony, the application ensures that user-provided data is reliable, secure, and meets the required standards. This practice helps combat common vulnerabilities like SQL injection, cross-site scripting (XSS), and other malicious attacks that exploit improperly validated input.

4.4.2 CSRF Tokens

Cross-Site Request Forgery or CSRF tokens are an important security feature in Cloud-Vitae that safeguard against CSRF attacks. CSRF attacks occur when malicious actors deceive a user's browser into executing unintended and harmful requests on their behalf, exploiting the user's authenticated session. To counter CSRF attacks, Symfony incorporates CSRF tokens.

When a form is rendered in Symfony, a unique CSRF token is created and associated with that specific form. This token is typically a random string consisting of characters. The generated CSRF token is securely stored either in the user's session or a cookie, depending on the Symfony configuration. The CSRF token is embedded within the form as a hidden field or included as part of a request header, based on the form's submission method. Upon form submission, the application automatically verifies the CSRF token's validity. It compares the submitted token with the one stored in the user's session or cookie. If the tokens don't match or if no token is provided, Symfony identifies the request as invalid.

By validating the CSRF token, Cloud-Vitae ensures that the form submission originates from the same application and user who initiated the form request. This precautionary measure stops attackers from manipulating the user's session and performing unauthorized actions on their behalf. Symfony simplifies the implementation of CSRF tokens by offering built-in functionalities for token generation, storage, and validation. This integration streamlines the process for developers, reinforcing application security by fortifying against common web application vulnerabilities. A CSRF token is used on all forms found on Cloud-Vitae.

4.5 Secure Communication

Utilizing secure communication protocols like HTTPS encrypts data transmitted between the user's browser and the application's server. This prevents eavesdropping, tampering, and interception of sensitive information during transit. Secure communication in a web application involves implementing various measures to protect the confidentiality, integrity, and authenticity of data transmitted between the web server and the client's browser. The goal is to ensure that sensitive information remains private, cannot be tampered with, and that the parties involved can verify each other's identities. Some important aspects of secure communication in a web application are:

- Transport Layer Security (TLS)/Secure Sockets Layer (SSL) which are cryptographic protocols used to establish secure connections over the internet. They encrypt the data exchanged between the server and the client, preventing unauthorized access or interception. Implementing TLS/SSL requires obtaining and installing an SSL certificate on the server, enabling HTTPS (HTTP over TLS/SSL) communication.
- The HTTPS protocol is the secure version of HTTP. It utilizes TLS/SSL to encrypt data during transmission, ensuring the protection of sensitive information like passwords, financial details, and personal data. HTTPS is essential for securing sensitive transactions, login pages, and any communication involving confidential information.
- SSL/TLS certificates are digital certificates issued by trusted certificate authorities (CAs). They verify the authenticity of the server and establish a secure connection. SSL/TLS certificates enable the browser to validate the server's identity and ensure that the connection is encrypted. These certificates are typically obtained and renewed from reputable CAs.
- Encryption involves encoding data in a way that makes it unreadable to unauthorized parties. In web application security, encryption is used to protect sensitive information

such as passwords, credit card numbers, and personal data. Data encryption ensures that intercepted communication remains unintelligible without the encryption keys.

- Web applications should employ secure authentication mechanisms to verify user identities. This often includes implementing strong password policies, multi-factor authentication (MFA), and protection against brute-force attacks. Authorization ensures that users have the appropriate permissions to access specific resources or perform particular actions within the application.
- Proper input validation is crucial to prevent security vulnerabilities like SQL injection and cross-site scripting (XSS) attacks. Input validation involves verifying and sanitizing user input to ensure it conforms to expected formats and does not contain malicious code.
- Web servers can include security headers in HTTP responses to enhance security. These headers provide instructions to the browser on how to handle various aspects of the web application, such as content security policies, cross-origin resource sharing, and HTTP Strict Transport Security (HSTS).
- Regular security audits help identify and address vulnerabilities in a web application. It is important to keep all software components, frameworks, and libraries up to date with the latest security patches to protect against emerging threats.

Cloud-Vitae implements a completely secure authentication system, proper input validation and data encryption as previously mentioned. To ensure the most secure communication possible, a domain name was purchased for the application (www.cloud-vitae.com) alongside a hosting plan which follows and provides veridic SSL and HTTPS protocols and certificates.

4.6 Session Management

Effective session management is crucial for securely maintaining user sessions. It involves generating secure session identifiers, setting timeouts, and properly handling session data to prevent session hijacking or fixation attacks. In Cloud-Vitae, the management of user sessions in is facilitated by the Symfony Session component. This component offers a convenient way to handle sessions and includes various essential elements.

Symfony allows developers to configure session-related parameters in the application's configuration files. This configuration encompasses settings such as the chosen storage mechanism (such as files, databases, or caching systems), session lifetime, session cookie options, and other relevant options. When a user interacts with Cloud-Vitae, a session is automatically initialized for that user. This initiation occurs upon the user's initial request to the application. During this process, a session object is created and associated with a unique session ID. Symfony provides diverse options for session storage, including storing session data in files, databases, or utilizing caching systems like Redis or Memcached. The chosen session storage mechanism is responsible for persisting the session data between requests.

Developers can interact with the session data through the session object provided by Symfony. This object acts as a wrapper around the session data, offering methods to read, write, and manipulate session variables. Additionally, Symfony's session component incorporates flash messages. Flash messages are temporary messages meant to be displayed to the user for a single request and are automatically removed afterward. These messages are commonly utilized for displaying success messages, error messages, or notifications following form submissions or other actions. For example, in Cloud-Vitae Flash messages are used to display registration errors.

To ensure the security of session data, Symfony implements protective measures. This involves using session IDs that are not predictable or guessable, as well as offering safeguards against session fixation attacks and session hijacking. By leveraging Symfony's session management capabilities, Cloud-Vitae easily manages user sessions, stores user-specific data, and performs necessary session-related operations within the web applications.

4.7 Security Testing

Regular security testing, including vulnerability scanning, penetration testing, and code reviews, identifies and addresses potential weaknesses in the application. This proactive approach enables patching vulnerabilities before they can be exploited. Cloud-Vitae's security was tested throughout all stages of development using multiple tools provided by Symfony. Security testing often includes scanning the application for known vulnerabilities and weaknesses. This can be done using automated vulnerability scanning tools that check for common security issues such as outdated libraries, misconfigurations, SQL injection, cross-site scripting (XSS), and more. Symfony provides tools like SensioLabs Security Advisories and SymfonyInsight to assist with vulnerability scanning.

Conducting a thorough review of the application's source code is another essential aspect of security testing. This involves analyzing the codebase to identify insecure coding practices, potential vulnerabilities, and areas where security improvements can be made. Code review helps ensure that security best practices, such as input validation, output encoding, proper authentication, and secure data handling, are implemented correctly.

Penetration testing, also known as ethical hacking, involves actively simulating attacks on the application to uncover potential vulnerabilities. Skilled security professionals perform manual tests to identify weaknesses in the system's architecture, authentication mechanisms, input validation, and other security controls. The goal is to assess how effectively the application can resist real-world attacks. This was done manually by the developer.

Security testing should evaluate the effectiveness of the authentication and authorization mechanisms implemented in the Symfony application. This includes verifying the strength of password policies, testing multi-factor authentication (MFA), evaluating access controls, and ensuring that only authorized users have appropriate access to sensitive resources.

The session management process should be thoroughly tested to ensure that session data is adequately protected. This includes verifying session ID generation, checking for session fixation vulnerabilities, and ensuring that session data is properly encrypted and stored securely.

Reviewing the application's configuration files and settings is crucial to identify any security weaknesses or misconfigurations. This includes checking for insecure default settings, validating encryption and hashing algorithms, verifying secure communication protocols (I.e. HTTPS), and reviewing permissions and access controls.

Security testing should examine how the application handles errors and exceptions. This includes verifying that error messages do not reveal sensitive information, validating that exceptions are properly caught and logged, and ensuring that error handling does not expose potential vulnerabilities.

By conducting thorough security testing on the application using Symfony’s tools, we can identify and address potential security vulnerabilities, enhance the application's resilience to attacks, and safeguard sensitive user data. It is an essential aspect of the software development lifecycle to ensure the overall security and integrity of Symfony applications. The steps mentioned above are the ones taken to ensure security for Cloud-Vitae.

4.8 Error Handling and Logging

Proper error handling and logging mechanisms help identify and troubleshoot security issues or unauthorized activities. Detailed logging provides valuable information for forensic analysis and auditing. These two are crucial components in Cloud-Vitae for managing and resolving errors and exceptions that occur within the application. These processes involve capturing, recording, and dealing with errors to ensure proper diagnosis and resolution of issues.

Symfony provides a robust mechanism that allows developers to define how different types of errors and exceptions are handled. This includes PHP errors, application-specific exceptions, and HTTP-related errors. Developers can customize error pages to provide a consistent and user-friendly experience when errors occur. They can also configure exception handling, specifying actions such as redirecting to specific error pages, displaying custom error messages, or logging exceptions for further analysis. Additionally, Symfony's event system allows developers to register error event listeners, enabling additional actions like sending notifications, performing logging, or triggering specific error recovery processes. For example, Cloud-Vitae implements custom error and warning pages. One of the most frequently seen ones is the access denied based on unverified email address. When the controller denied access to the user based on this condition, it does not render a generic error page, but a custom one that explains why the user is seeing this, how to fix it and the option to re-send the verification email.

Logging, on the other hand, involves recording and storing relevant information about the application's runtime behavior, including errors, exceptions, warnings, and other loggable events. Symfony offers a powerful logging component that supports different log levels, such as debug, info, warning, error, and critical. Developers can specify the severity of logged events to filter and prioritize information based on importance. Symfony's logging system also supports the concept of log channels, allowing developers to categorize log entries based on different areas or components of the application. This helps organize and differentiate logs for easier troubleshooting. Symfony provides various log handlers that determine where log entries should be stored or sent, such as writing logs to files, sending them to external services or databases, or even delivering logs through email or other notification channels. Additionally, Symfony's logging system enables developers to include contextual information in log entries, such as the current user, request details, environment variables, or any other relevant data, facilitating error understanding and diagnosis. In the case of Cloud-Vitae, even though logging is such a useful tool, it was completely unnecessary since general debugging was enough to solve any unexpected development problems. However, for the production environment of Cloud-Vitae, Symfony is set up to save general purpose logs in a dedicated application file on the hosting server, just as a backup in case other errors unexpectedly occur after launch.

4.9 Secure Configuration

Ensuring proper configuration of the web server, application framework, and other components with secure settings is vital. This includes disabling unnecessary services, applying security patches and updates, and implementing secure configurations recommended by the framework or technology used. Symfony offers a comprehensive set of tools and guidelines to ensure the secure configuration of Cloud-Vitae. Secure configuration involves establishing application settings and options in a manner that minimizes potential security risks and aligns with recommended security practices.

Cloud-Vitae advocates for the use of environment variables or specialized secret management tools to store sensitive information like database credentials, API keys, and encryption keys. This practice prevents accidental exposure of sensitive data in version control systems. The environment file is located under “app/.env” in the application. Symfony recommends hardening the configuration files by disabling unnecessary features and services that are not essential for the application's operation. By reducing the attack surface, potential security risks are minimized.

In production environments, Symfony advises disabling the debug mode to avoid exposing sensitive information. Proper error handling ensures that error messages and stack traces are not divulged to end-users, which could potentially reveal internal application details, and Cloud-Vitae falls between these guidelines. Symfony provides configuration options and libraries for secure encryption and hashing purposes (see chapter 4.3.1). This enables developers to encrypt sensitive data or hash passwords using strong cryptographic algorithms. Secure storage of encryption keys and hashing salts is also emphasized.

Symfony also facilitates the configuration of secure communication protocols such as HTTPS (HTTP over SSL/TLS) to encrypt data transmitted between the application and the client's browser. This ensures the confidentiality and integrity of sensitive information during transmission (chapter 4.5). Symfony's security component also enables developers to define access control rules based on user roles and permissions. This allows for precise restrictions on routes, controllers, or actions, preventing unauthorized access to sensitive parts of the application (chapter 4.2.1).

Symfony encourages the use of trusted and regularly maintained libraries and components. Keeping dependencies up to date with the latest secure versions helps mitigate potential vulnerabilities and ensures the application benefits from security patches. Cloud-Vitae is developed with this idea in mind, as almost all the implementations that corresponded to older versions of Symfony and are now considered deprecated have been completely replaced over time.

By adhering to Symfony's recommended practices for secure configuration, Cloud-Vitae significantly reduced the risk of security breaches and enhanced the overall security posture of the application. Staying informed about security best practices, staying up to date with security updates, and consistently applying secure configuration measures throughout the application's lifecycle are crucial for maintaining a secure Symfony application.

4.10 User Education

Educating users about best practices for strong passwords, avoiding phishing attempts, and being cautious about sharing personal information significantly enhances overall application security. Besides providing ways in which users can enhance their own security using the application, it is mandatory to provide the notices specified by law, such as the terms and conditions of usage or the privacy policy.

4.10.1 Guiding the user

Guiding users in Cloud-Vitae refers to the process of providing information, guidance, and training to users to help them understand and use the application effectively, safely, and in line with best practices. The goal of user education is to empower users with the knowledge and skills they need to navigate the application, make informed decisions, and engage with its features and functionalities. In a web application, user education plays a crucial role in enhancing the user experience, promoting engagement, and ensuring usability and security.

When users first access the application, onboarding processes can help familiarize them with its features, settings, and functionalities. User documentation, including tutorials, guides, FAQs, and tooltips, provides detailed instructions on performing tasks and navigating different sections of the application. For example, in Cloud-Vitae on the details edit page (specifically the form in which the user inputs their personal data, work experience and so on), the fields that are not easy to understand or that may create confusion for the user have an extra label that contains useful information on what is the best way to complete that field (I.e. Salary for a particular position, it refers to amount per month, currency should be specified and it is not mandatory, meaning “confidential” is a valid answer as well). The design of the user interface is intuitive and friendly, providing clear instructions and visual cues. Well-designed interfaces guide users through workflows, making it easier for them to interact with the application and achieve their goals.

Informative error messages are essential for guiding users when they encounter issues or mistakes. User-friendly error messages offer actionable guidance to resolve problems or suggest alternative actions. Feedback mechanisms, such as success messages and progress indicators, keep users informed about the outcome of their actions and maintain engagement. Cloud-Vitae implements a custom error message system that respects these criteria and has already been discussed (chapter 4.8). Regularly informing users about updates, new features, and improvements keeps them engaged and aware of the application's capabilities. This can be achieved through release notes, changelogs, email notifications, or in-app announcements. This is of course cannot be yet achieved by the application since it is on its first production version, but it is a significant piece of the future implementations plan.

Prioritizing user education empowers users to maximize the application's features, enhances their experience, reduces errors, and improves overall usability and security. It fosters a positive and productive user experience, leading to increased engagement and satisfaction.

4.10.2 Mandatory Notices

Legal compliance and protection of user rights and privacy are crucial in web applications. To meet these requirements, certain mandatory documentation is necessary. The specific obligations may differ based on the application's nature and the jurisdiction involved.

4.10.2.1 Terms and Conditions of Usage

The terms and conditions of usage define the rules, rights, and responsibilities governing the use of Cloud-Vitae. They cover aspects like user obligations, intellectual property rights, liability disclaimers, applicable laws, and dispute resolution mechanisms. They are created in such a way to strictly follow the GDPR guidelines and can be found under /legal.

It's important to recognize that the examples bellow are not exhaustive, and additional legal requirements may apply depending on the jurisdiction and specific characteristics of the web application. Consulting legal professionals or experts knowledgeable in applicable laws and regulations is necessary to ensure compliance. Cloud-Vitae took all the possible steps to follow regulations and treat this subject accordingly with its limited budget, but it is not guaranteed that all the information included as disclaimers for the users are enough to meet this scope.

4.10.2.2 Privacy and Cookie Policies

A legally binding document, the Privacy Policy outlines how the web application collects, uses, stores, and safeguards user data. It informs users about the types of data collected, purposes of data usage, potential sharing with third parties, and measures taken to ensure data security and compliance with privacy laws.

If the web application employs cookies or similar tracking technologies, a Cookies Policy is necessary. Symfony uses cookies by default, so it is mandatory that they are clearly reported in Cloud-Vitae's documentation. This policy informs users about the types of cookies used, their purposes, how they are utilized, and options for managing cookie preferences. Cloud-Vitae does not have an option to disable cookies as all the cookies used are essential for the application to function properly. Users agree to the Privacy and Cookie policies by using the website, and consent to this when registering an account.

4.10.2.3 Data Protection and GDPR Compliance

If the web application collects and processes personal data of users in the European Union (which is the case for Cloud-Vitae), adhering to the General Data Protection Regulation (GDPR) is mandatory. Compliance may entail additional documentation, such as a Data Processing Agreement (DPA) for data processors and mechanisms to obtain user consent. Cloud-Vitae's DPA is contained within its Privacy policy and Terms and Conditions of usage, and users must agree to it before creating an account on the platform.

5. Running Cloud-Vitae on a Local Environment

Running a web application on a local environment provides several advantages that simplify the development process compared to running it on a remote or production environment. For the purpose of this paper, running Cloud-Vitae on a local environment is the most effective way, since setting up a local environment is typically faster and more straightforward than configuring a remote or production environment. Anyone can easily install a local server, database, and other necessary software, enabling them to start testing the application immediately.

Running a web application locally also ensures isolation from the external network and potential security risks. This reduces the chances of unauthorized access or malicious attacks during the development and testing phases. Furthermore, utilizing a local environment eliminates the need for additional resources and infrastructure costs associated with running the application on a remote or production server. This is particularly advantageous for individual developers or small teams with limited budgets which is the exact case of Cloud-Vitae.

5.1 Creating the Local Environment

To create a local environment, we will need to install a few pieces of software necessary for the project to work optimally.

5.1.1 Setting up Ubuntu Linux

Firstly, we will need Ubuntu Linux, which is a widely used operating system that is built on the Linux kernel and is known for its user-friendly interface, stability, and strong community support. It offers a wide range of software packages for various purposes. It can be used either via a virtual machine or a main operating system (OS).

5.1.1.1 Ubuntu Linux with a Virtual Machine

Installing Ubuntu Linux via a Virtual Machine makes for a more straightforward and easier process and allows users to also keep their original operating system accessible at any time. This method is preferred, and for Cloud-Vitae it is recommended to use VM Ware Workstation for the virtual machine, since it is a free to use software solution and provides a secure and isolated environment for running Ubuntu.

Visit the official VMware website (https://www.vmware.com/) and download the appropriate version of VMware Workstation for your operating system. Ensure compatibility with your system. Visit the official VMware website (https://www.vmware.com/) and download the appropriate version of VMware Workstation for your operating system. Ensure compatibility with your system. Install VMware Workstation on your computer and open the program.

Create a new virtual machine: Click on the "Create a New Virtual Machine" option in VMware Workstation to start the new virtual machine creation wizard. In the wizard, choose the "Installer disc image file (iso)" option and navigate to the location where you saved the Ubuntu ISO file. Select "Linux" as the guest operating system and choose the version of Ubuntu you downloaded. Set the name and location for the virtual machine files, allocate the desired amount of memory (RAM) and disk space, and customize other settings according to your needs.

Once the virtual machine is created, select it from the VMware Workstation library and click on the "Power on this virtual machine" option. The virtual machine will boot from the Ubuntu ISO file. Follow the on-screen instructions to install Ubuntu within the virtual machine. You can select language preferences, configure disk partitions, set up a user account, and choose optional software packages during the installation process. After the installation is complete, Ubuntu will be installed within the virtual machine. You can log in and start using Ubuntu.

5.1.1.2 Ubuntu Linux as a Default OS

Visit the official Ubuntu website at https://ubuntu.com/ and download the ISO image of the desired Ubuntu version. Make sure to choose the appropriate version based on your system architecture (32-bit or 64-bit). Once the ISO file is downloaded, you need to create a bootable USB drive or burn it onto a DVD. You may require third-party software like Rufus (for USB) or ImgBurn (for DVD) to perform this step. Insert the bootable USB or DVD into your computer and restart it. Access the BIOS or UEFI settings to adjust the boot order, giving priority to the USB or DVD drive.

After rebooting, the Ubuntu installer will load. Choose your preferred language and select the "Install Ubuntu" option. Alternatively, you can try Ubuntu without installing it to test it before proceeding. Follow the on-screen instructions to configure settings such as language, keyboard layout, network configurations, and disk partitioning. Depending on your preferences, you can install Ubuntu alongside another operating system or wipe the disk and install Ubuntu as the sole OS. Provide a username, password, and computer name for your Ubuntu system. This account will have administrative privileges.

The installer will copy the necessary files and install Ubuntu on your system. This process may take some time, so it's important to be patient. Once the installation is complete, you will be prompted to restart your computer. After rebooting, Ubuntu Linux will load, and you can begin exploring and using the operating system. It's important to note that the installation process may vary slightly depending on the specific version of Ubuntu and the hardware configuration of your computer. For more detailed installation instructions and troubleshooting guidance tailored to your situation, it is recommended to consult the official Ubuntu documentation or seek assistance from the Ubuntu community resources.

5.1.2 Setting up Docker and Docker Compose Plugin

After installing Ubuntu Linux either via a virtual machine or as a main operating system, we have to set up a few tools in order to run the Cloud-Vitae project properly. Installing Docker is a simple process, and more information can be found in Docker’s official documentation (https://docs.docker.com/engine/install/ubuntu/).

To start, open a terminal instance in Ubuntu Linux. To install Docker, you will need to copy and paste the following commands one by one. Please note that in this terminal, instead of the normal paste shortcut (CTRL+V) it is better to use CTRL+SHIFT+V.

First, we need to update the apt package index by running “sudo apt-get update”, and then we can install the latest docker version by running “sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin” in the terminal. After that, a system restart is recommended.

The second step is installing the Docker Compose Plugin, but first we need to update the apt package index again by running “sudo apt-get update”, and then run “sudo apt-get install docker-compose-plugin”. To check if everything is working properly, we can run “docker” for Docker and it should return a list of possible Docker commands alongside the installed Docker version, and “docker compose version” which should return the Docker Compose Plugin version. If both of these return what they should, Docker was properly installed.

5.1.3 Setting up GIT

Installing GIT on Ubuntu Linux is a very simple process. First, we need to update the apt package index by running “sudo apt-get update”, and then we have to run “sudo apt-get install git”. If we now run “git --version” and it return our current version of GIT, it is installed correctly.

5.1.4 Setting up PHP Storm

PHP Storm can be found in the Snap Store of Ubuntu Linux, under the development section. We just have to click download and install, and we will be notified when the process is finished, and PHP Storm is ready to use.

5.2 Cloning the Repository

After the PHP Storm installation is finished, we will open the IDE. We will be prompted by a menu in which we have to open or create a project. Since Cloud-Vitae is stored online on GitHub's servers, we will have to clone it from VCS instead of copying the files manually.

5.2.1 Cloning from VCS in PHP Storm

On this menu, we will click on "Get from VCS". Another menu will appear, in which we have to input the project url (https://github.com/AN0R31/Cloud-Vitae), and select the directory under which the project will be created, and then click on "Clone". Cloning the repository might take a couple of minutes, and it also may require you to authenticate into a GitHub account that has access to the repository. If you received the project files in another way, you can just right-click on the directory containing these files and select "Open with another application", and then select PHP Storm. This will also create a clone of Cloud-Vitae.

5.2.2 Validating the project files

After successfully cloning the project, it is recommended but not mandatory to validate the files. To do this, you can merge the main branch of Cloud-Vitae's repository into your local branch and make sure GIT returns "all files are up to date" (the best way to do this is to open PHP Storm's terminal and run "git init" and then run "git remote add origin" followed by the provided repository link), or if you do not have access to merge you can check that the directory structure of the project matches the one bellow.

5.3 Setting up the project

All the previous steps should have constructed the environment needed to run Cloud-Vitae if executed correctly. Now we can move forward to warming up the project and eventually running it.

5.3.1 Building the Docker containers

If Docker was correctly installed, after running "docker ps" in PHP Storm's terminal it should return an empty list of containers. We first need to build the containers, and then run them. To achieve this, run "docker compose build --no-cache". This process could take up to one hour but has to be executed only once.

5.3.2 Running the Docker containers

After Docker finished building the containers, you will press CTRL+C in the terminal to kill them. Run "docker compose down" to terminate any remaining processes, and then "docker compose up" to start the containers.

5.3.3 Starting a Docker service in PHP Storm

After running the containers, we will click on the "Services" tab in PHP Storm next to the "Terminal" tab (on the bottom left side). Here we will click the plus sign and select "Add a Docker service". Wait a few seconds for the configuration to load and click on "Add". The Docker service should now appear on the left side of the services tab.

5.3.4 Installing NPM and Watch on the service

After adding the Docker service, in the "Services" tab double click on it. It should have multiple sub-menus, one corresponding to each Docker container. Click on the "php" container, and in the menu that open on the right side click on "Terminal". This will open a service terminal, which is used to operate directly on the project and is very useful for specific components such as Symfony's CLI and NPM. Here, run "npm install" and after it is finished run "npm run watch". This will start the Watch service, and it will refresh and compile all the assets efficiently.

5.3.5 Installing Composer Dependencies

Open another service terminal by clicking on "Terminal". Here run "composer update" to update the local dependencies and "composer install" to make sure they have all been installed. If there are red errors in the terminal after it executes, it means the dependencies were not correctly installed and Cloud-Vitae will throw errors and not run properly.

5.3.6 Running the database migrations

In the same terminal, run "php bin/console doctrine:migrations:migrate". If you get a green message saying "All migrations executed successfully", this process is finished. If you get a SQL error about the database not existing, go to "localhost:8081" in any browser and manually create a MySQL database named "cloud-vitae" and then run the migrate command again. If localhost is giving you access denied when accessing from a web browser, restart the Docker container or ultimately the computer.

5.4 Getting started

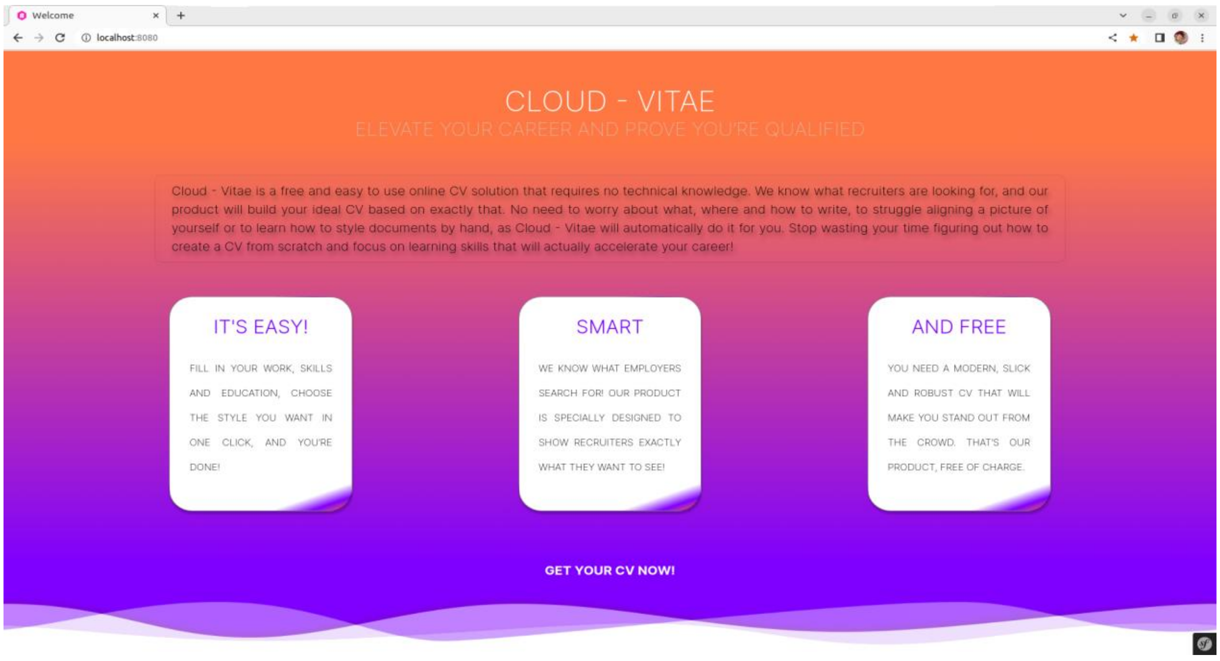
To access Cloud-Vitae, open any web browser and type "localhost:8080" in the top search bar. This link is the entry point to the application.

6. Application Guide

The design and general user walkthrough for Cloud-Vitae is very friendly and easy to understand. This chapter represents a quick overview on how to use the application and specific details and particularities on its functionalities.

6.1 Landing Page

The landing page of Cloud-Vitae is the entry point to the application and can be reached by accessing "localhost:8080" while not authenticated. It contains general information on what the application does, what its purpose is and how to use it.



CLOUD - VITAE

ELEVATE YOUR CAREER AND PROVE YOU'RE QUALIFIED

Cloud - Vitae is a free and easy to use online CV solution that requires no technical knowledge. We know what recruiters are looking for, and our product will build your ideal CV based on exactly that. No need to worry about what, where and how to write, to struggle aligning a picture of yourself or to learn how to style documents by hand, as Cloud - Vitae will automatically do it for you. Stop wasting your time figuring out how to create a CV from scratch and focus on learning skills that will actually accelerate your career!

IT'S EASY!

FILL IN YOUR WORK, SKILLS AND EDUCATION, CHOOSE THE STYLE YOU WANT IN ONE CLICK, AND YOU'RE DONE!

SMART

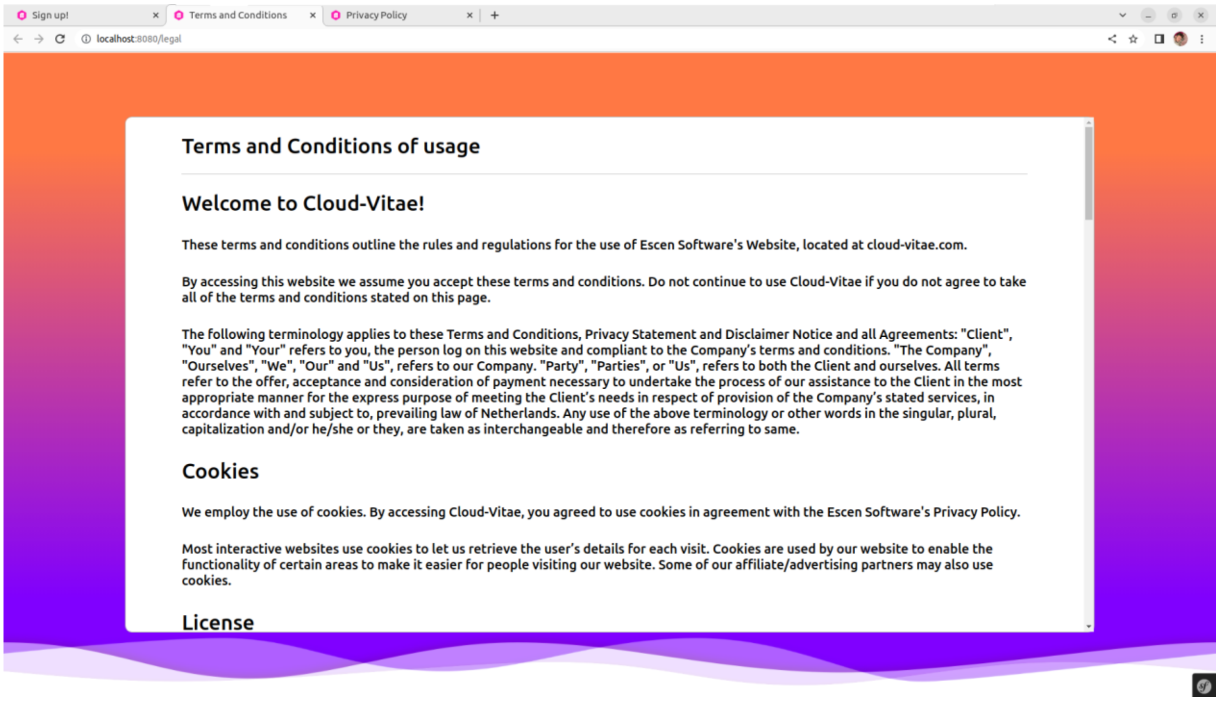
WE KNOW WHAT EMPLOYERS SEARCH FOR! OUR PRODUCT IS SPECIALLY DESIGNED TO SHOW RECRUITERS EXACTLY WHAT THEY WANT TO SEE!

AND FREE

YOU NEED A MODERN, SLICK AND ROBUST CV THAT WILL MAKE YOU STAND OUT FROM THE CROWD. THAT'S OUR PRODUCT, FREE OF CHARGE.

6.2 Legal Notices

At the bottom of the landing page, there is a button to create an account entitled "Get your CV now!". When clicked, the user must select a registration method, and if email address is chosen, then it will redirect to the registration route which can be reached by accessing "localhost:8080/register" while not authenticated. Here, the user can read Cloud-Vitae's legal notices by clicking on "Terms and conditions".



Terms and Conditions of usage

Welcome to Cloud-Vitae!

These terms and conditions outline the rules and regulations for the use of Escen Software's Website, located at cloud-vitae.com.

By accessing this website we assume you accept these terms and conditions. Do not continue to use Cloud-Vitae if you do not agree to take all of the terms and conditions stated on this page.

The following terminology applies to these Terms and Conditions, Privacy Statement and Disclaimer Notice and all Agreements: "Client", "You" and "Your" refers to you, the person log on this website and compliant to the Company's terms and conditions. "The Company", "Ourselves", "We", "Our" and "Us", refers to our Company. "Party", "Parties", or "Us", refers to both the Client and ourselves. All terms refer to the offer, acceptance and consideration of payment necessary to undertake the process of our assistance to the Client in the most appropriate manner for the express purpose of meeting the Client's needs in respect of provision of the Company's stated services, in accordance with and subject to, prevailing law of Netherlands. Any use of the above terminology or other words in the singular, plural, capitalization and/or he/she or they, are taken as interchangeable and therefore as referring to same.

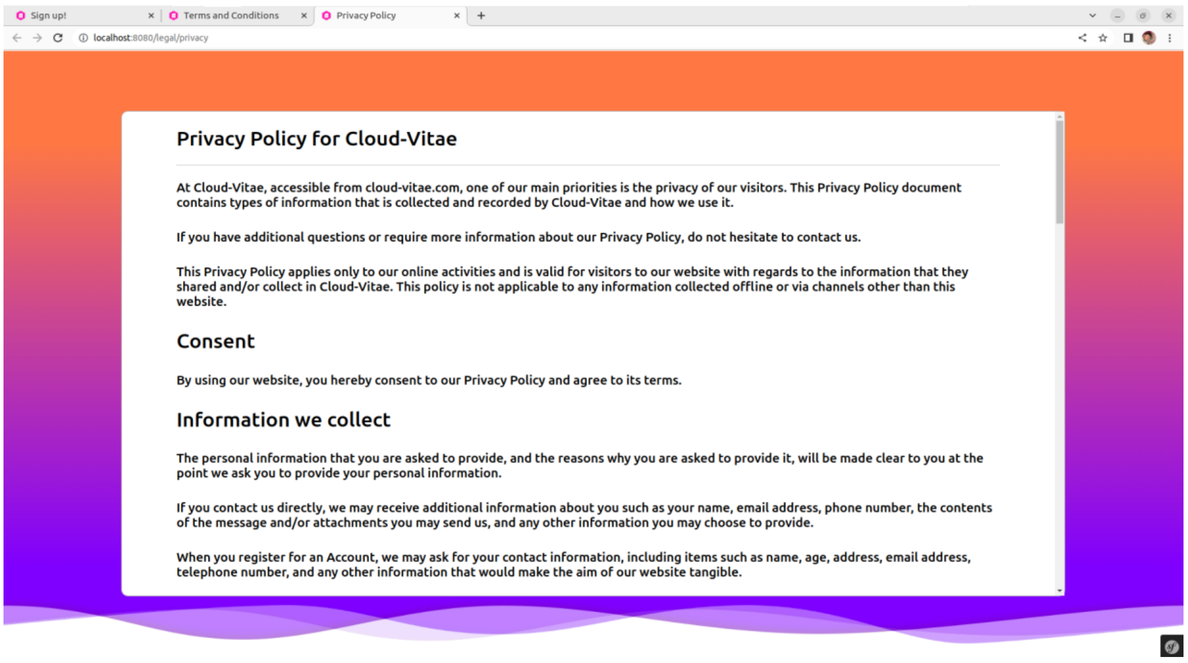
Cookies

We employ the use of cookies. By accessing Cloud-Vitae, you agreed to use cookies in agreement with the Escen Software's Privacy Policy.

Most interactive websites use cookies to let us retrieve the user's details for each visit. Cookies are used by our website to enable the functionality of certain areas to make it easier for people visiting our website. Some of our affiliate/advertising partners may also use cookies.

License

A more detailed overview of the legal notices can be found in chapter 4.10.2. This page contains Cloud-Vitae's terms and conditions of usage (which can also be accessed at "localhost:8080/legal"), and a link to the Privacy policy (also accessible from "localhost:8080/legal/privacy").



Privacy Policy for Cloud-Vitae

At Cloud-Vitae, accessible from cloud-vitae.com, one of our main priorities is the privacy of our visitors. This Privacy Policy document contains types of information that is collected and recorded by Cloud-Vitae and how we use it.

If you have additional questions or require more information about our Privacy Policy, do not hesitate to contact us.

This Privacy Policy applies only to our online activities and is valid for visitors to our website with regards to the information that they shared and/or collect in Cloud-Vitae. This policy is not applicable to any information collected offline or via channels other than this website.

Consent

By using our website, you hereby consent to our Privacy Policy and agree to its terms.

Information we collect

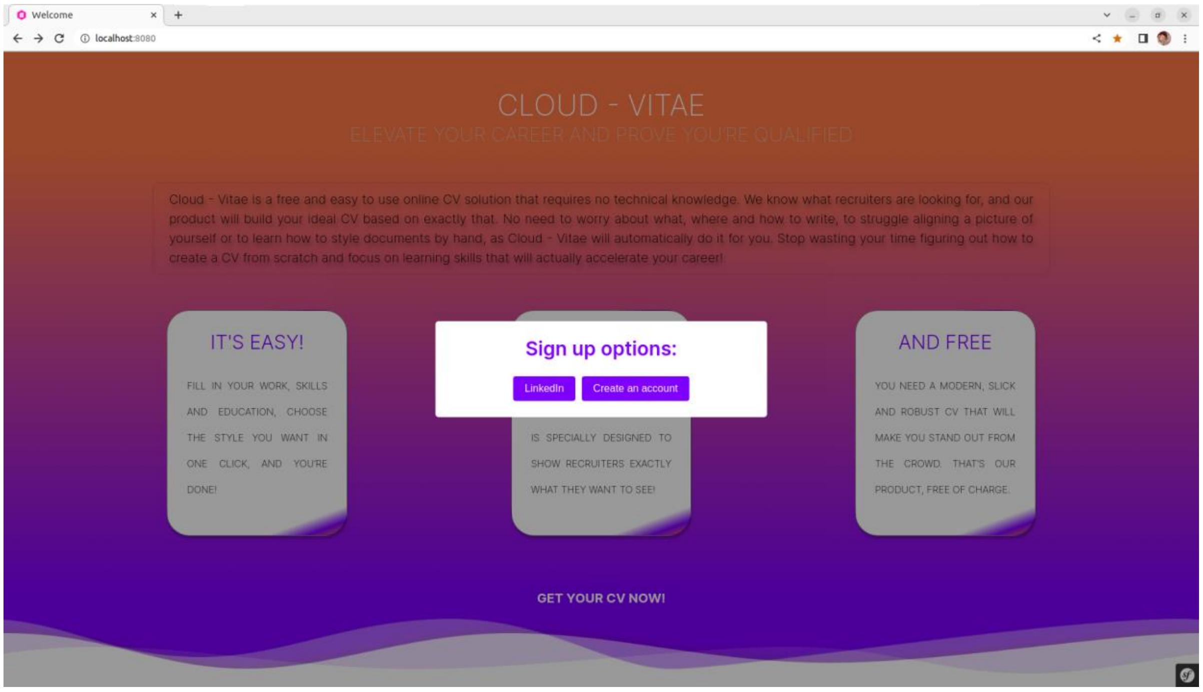
The personal information that you are asked to provide, and the reasons why you are asked to provide it, will be made clear to you at the point we ask you to provide your personal information.

If you contact us directly, we may receive additional information about you such as your name, email address, phone number, the contents of the message and/or attachments you may send us, and any other information you may choose to provide.

When you register for an Account, we may ask for your contact information, including items such as name, age, address, email address, telephone number, and any other information that would make the aim of our website tangible.

6.3 Authentication and Registration

When clicking the "Get your CV now!" button at the bottom of the landing page, the user has to choose between authentication with email address or with LinkedIn.



CLOUD - VITAE

ELEVATE YOUR CAREER AND PROVE YOU'RE QUALIFIED

Cloud - Vitae is a free and easy to use online CV solution that requires no technical knowledge. We know what recruiters are looking for, and our product will build your ideal CV based on exactly that. No need to worry about what, where and how to write, to struggle aligning a picture of yourself or to learn how to style documents by hand, as Cloud - Vitae will automatically do it for you. Stop wasting your time figuring out how to create a CV from scratch and focus on learning skills that will actually accelerate your career!

IT'S EASY!

FILL IN YOUR WORK, SKILLS AND EDUCATION, CHOOSE THE STYLE YOU WANT IN ONE CLICK, AND YOU'RE DONE!

Sign up options:

LinkedIn | Create an account

IS SPECIALLY DESIGNED TO SHOW RECRUITERS EXACTLY WHAT THEY WANT TO SEE!

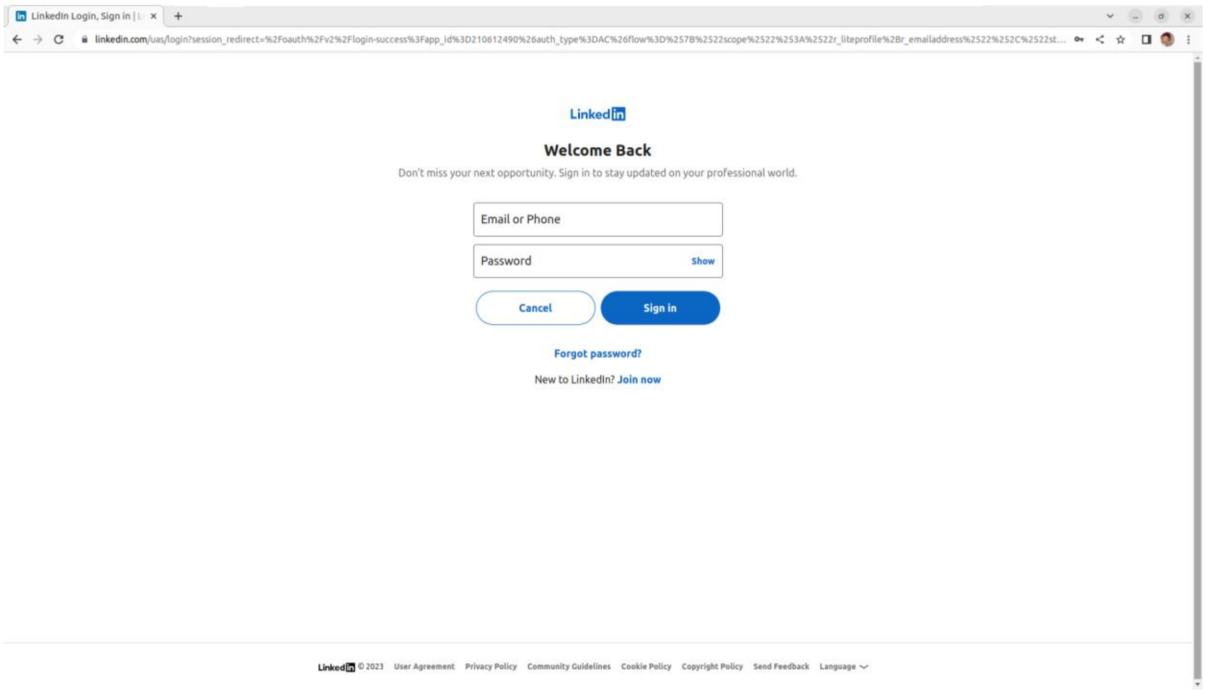
AND FREE

YOU NEED A MODERN, SLICK AND ROBUST CV THAT WILL MAKE YOU STAND OUT FROM THE CROWD. THAT'S OUR PRODUCT, FREE OF CHARGE.

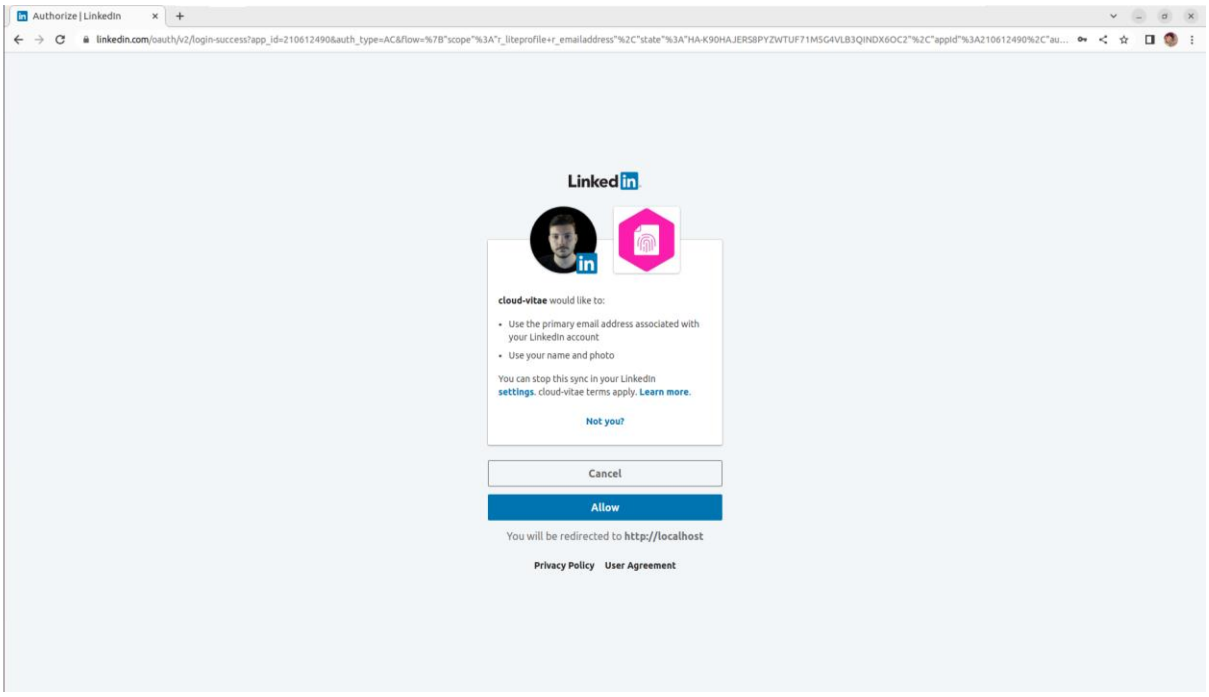
GET YOUR CV NOW!

6.3.1 Registering with LinkedIn

If the user chooses authentication with LinkedIn, he will be redirected to the LinkedIn API authentication page. Here the user must input his LinkedIn login information and click on "Sign in".


LinkedIn Login, Sign in | L
linkedin.com/uas/login?session_redirect=%2Foauth%2Fv2%2Flogin-success%3Fapp_id%3D210612490%26auth_type%3DAC%26flow%3D%257B%2522scope%2522%253A%2522r_liteprofile%2Br_emailaddress%2522%252C%2522st...
Linked in
Welcome Back
Don't miss your next opportunity. Sign in to stay updated on your professional world.
Email or Phone
Password
Show
Cancel
Sign in
Forgot password?
New to LinkedIn? Join now
Linked in © 2023
User Agreement
Privacy Policy
Community Guidelines
Cookie Policy
Copyright Policy
Send Feedback
Language

LinkedIn will then show the user the information they will share with Cloud-Vitae, and the user must agree in order to proceed.


Authorize | LinkedIn
linkedin.com/oauth/v2/login-success?app_id=210612490&auth_type=AC&flow=%7B"scope"%3A"r_liteprofile+r_emailaddress"%2C"state"%3A"HA-K90HAJERS8PYZWTUF71M5G4VLB3QINDX6OC2"%2C"appId"%3A210612490%2C"au...
Linked in
Not you?
Cancel
Allow
Privacy Policy
User Agreement

cloud-vitae would like to:

- Use the primary email address associated with your LinkedIn account
- Use your name and photo

You can stop this sync in your LinkedIn **settings**. cloud-vitae terms apply. **Learn more**.

You will be redirected to **http://localhost**

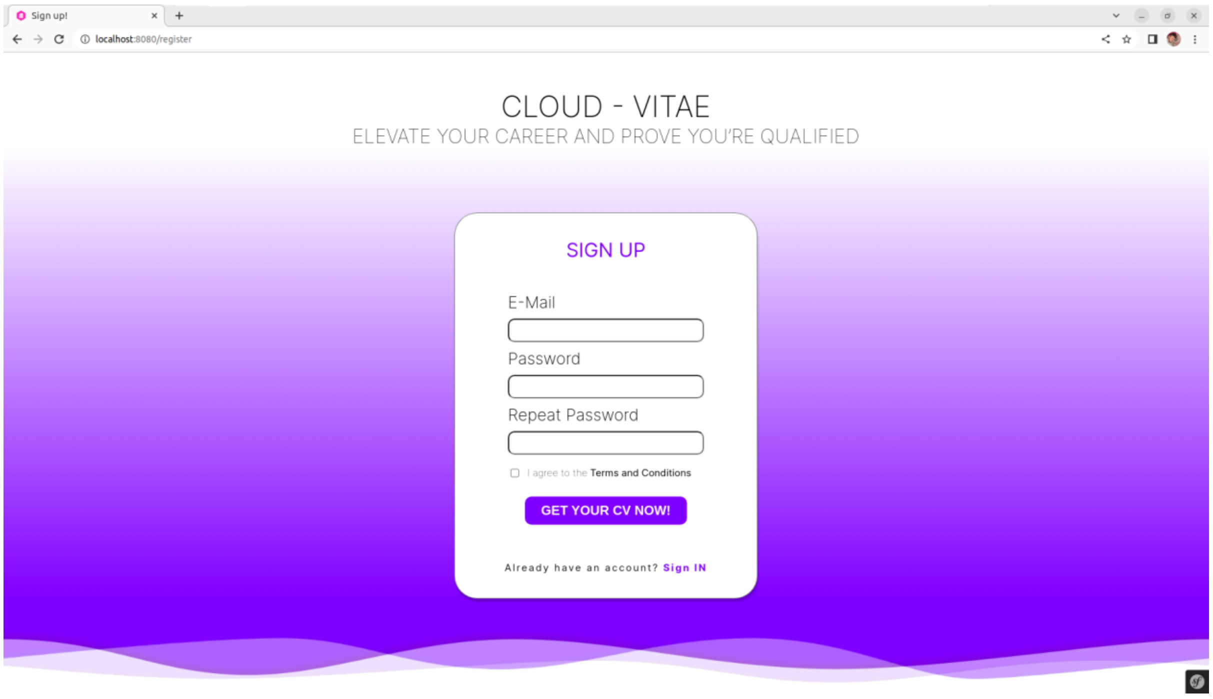
After that, the user is redirected back along with the information provided by LinkedIn's API, and Cloud-Vitae registers an account and authenticates the user automatically.

6.3.2 Authentication with LinkedIn

To authenticate with LinkedIn in an already existing Cloud-Vitae account, the user must follow the same steps as in registering an account with LinkedIn (chapter 6.3.1). The only difference is that the user will not have to review and agree again to the share of data between the two applications, and if the LinkedIn authentication is still valid in this session, the user will not have to input his LinkedIn credential again.

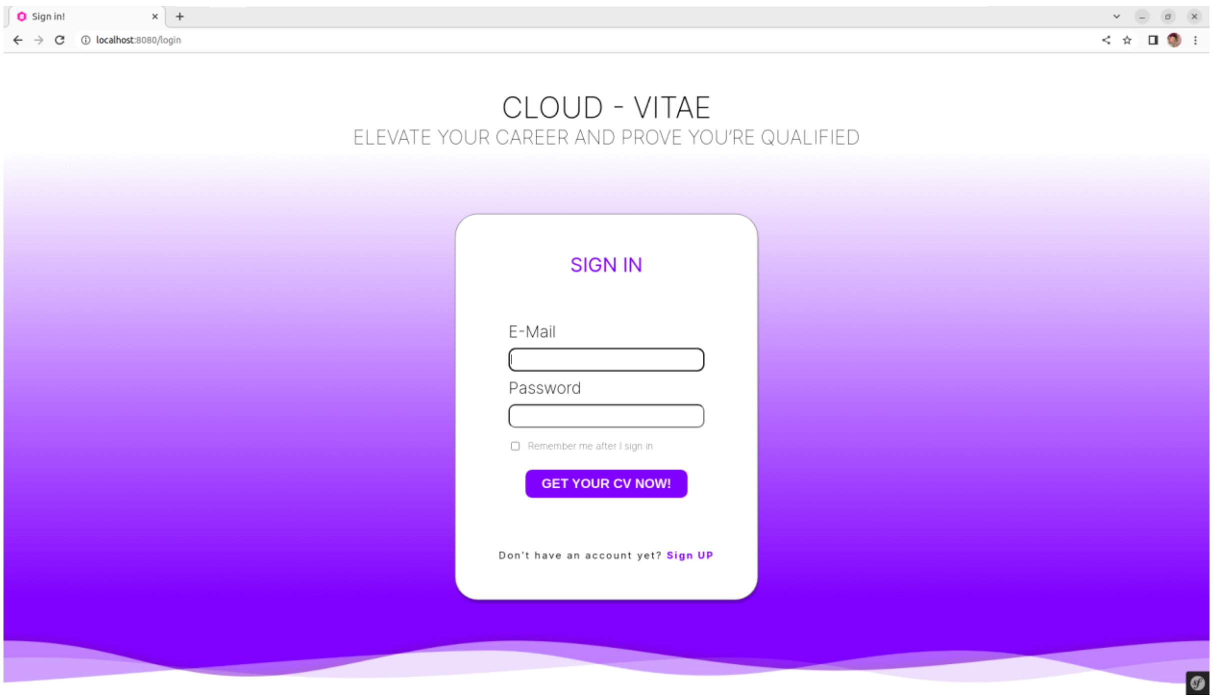
6.3.3 Registering with Email Address

If the user chooses to authenticate with an email address, he will be redirected to the register page, which can be reached by accessing “localhost:8080/register” while not authenticated. Here is located a form in which the user has to input a valid email address, password and agree to the terms and conditions. After clicking “Get your CV now!”, if the controller decides the form is validly completed, it will create an account for the user and authenticate him automatically.


Sign up!
localhost:8080/register
CLOUD - VITAE
ELEVATE YOUR CAREER AND PROVE YOU'RE QUALIFIED
SIGN UP
E-Mail
Password
Repeat Password
I agree to the Terms and Conditions
GET YOUR CV NOW!
Already have an account? Sign IN

6.3.4 Authenticating with Email Address

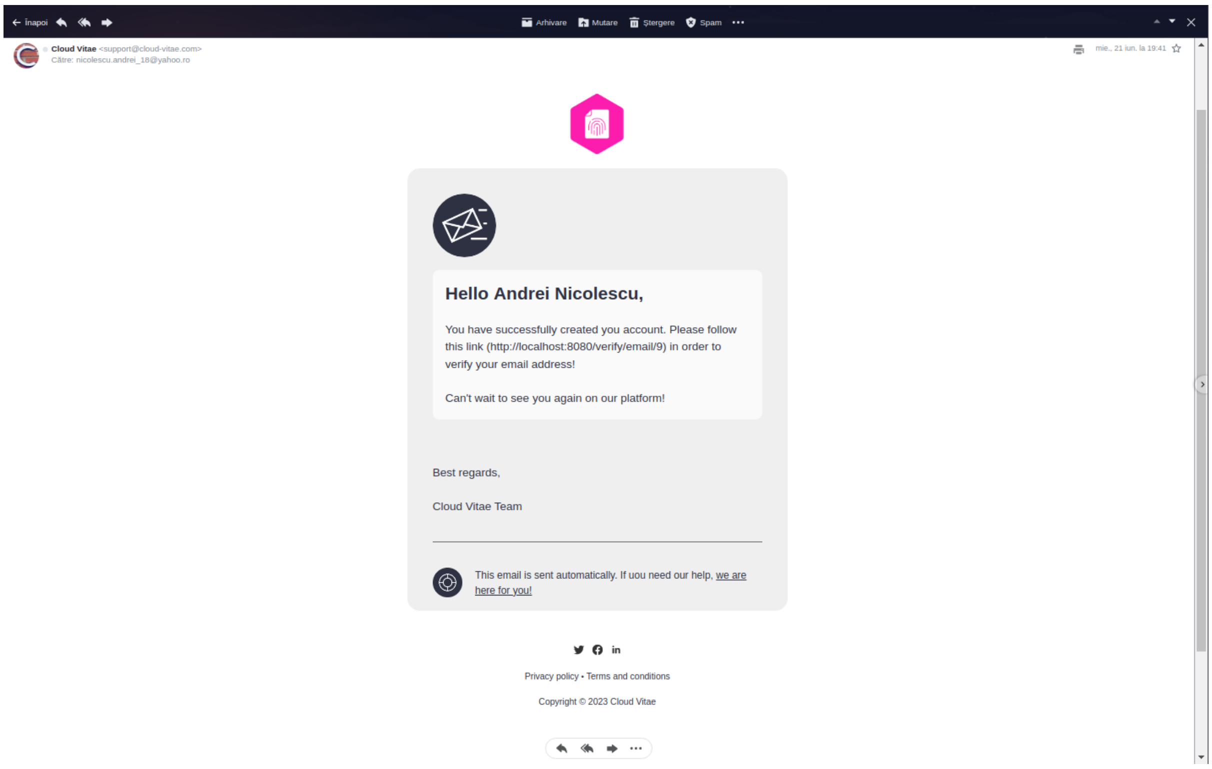
To authenticate in Cloud-Vitae with an already existing email-based account, a user must reach the login page by clicking on “Sign in” from the registration page. It can also be reached by accessing “localhost:8080/login” while not authenticated.


Sign in!
localhost:8080/login
CLOUD - VITAE
ELEVATE YOUR CAREER AND PROVE YOU'RE QUALIFIED
SIGN IN
E-Mail
Password
Remember me after I sign in
GET YOUR CV NOW!
Don't have an account yet? Sign UP

Here the user must input their Cloud-Vitae credentials and click on "Login". If the controller validates these credentials, the user will be successfully authenticated.

6.4 Verifying the Email Address

After a user register for an account, whether it is via LinkedIn or by email address, he must verify this email address. Cloud-Vitae sends an email to the associated email (for LinkedIn it is provided by the API) which contains a link that the user must click on. He must also be authenticated when following the link, otherwise the verifying process will not work.



Cloud Vitae <support@cloud-vitae.com>
Către: nicolescu.andrei_18@yahoo.ro
mie., 21 iun. la 19:41

Hello Andrei Nicolescu,

You have successfully created you account. Please follow this link (http://localhost:8080/verify/email/9) in order to verify your email address!

Can't wait to see you again on our platform!

Best regards,

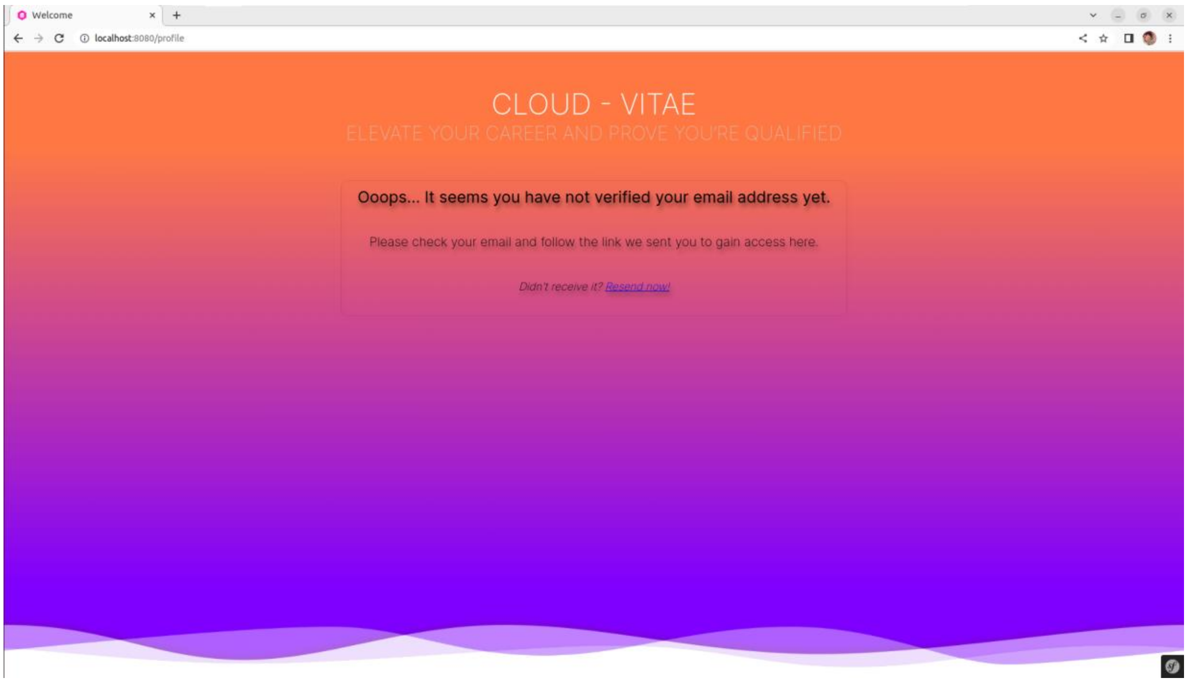
Cloud Vitae Team

This email is sent automatically. If uou need our help, we are here for you!

Privacy policy • Terms and conditions

Copyright © 2023 Cloud Vitae

Until the user does this, access will be denied to Cloud-Vitae's functionalities. The error page rendered by the controller also allows the user to re-send the verification email, in case it was lost or not received.


Welcome

localhost:8080/profile

CLOUD - VITAE

ELEVATE YOUR CAREER AND PROVE YOU'RE QUALIFIED

Ooops... It seems you have not verified your email address yet.

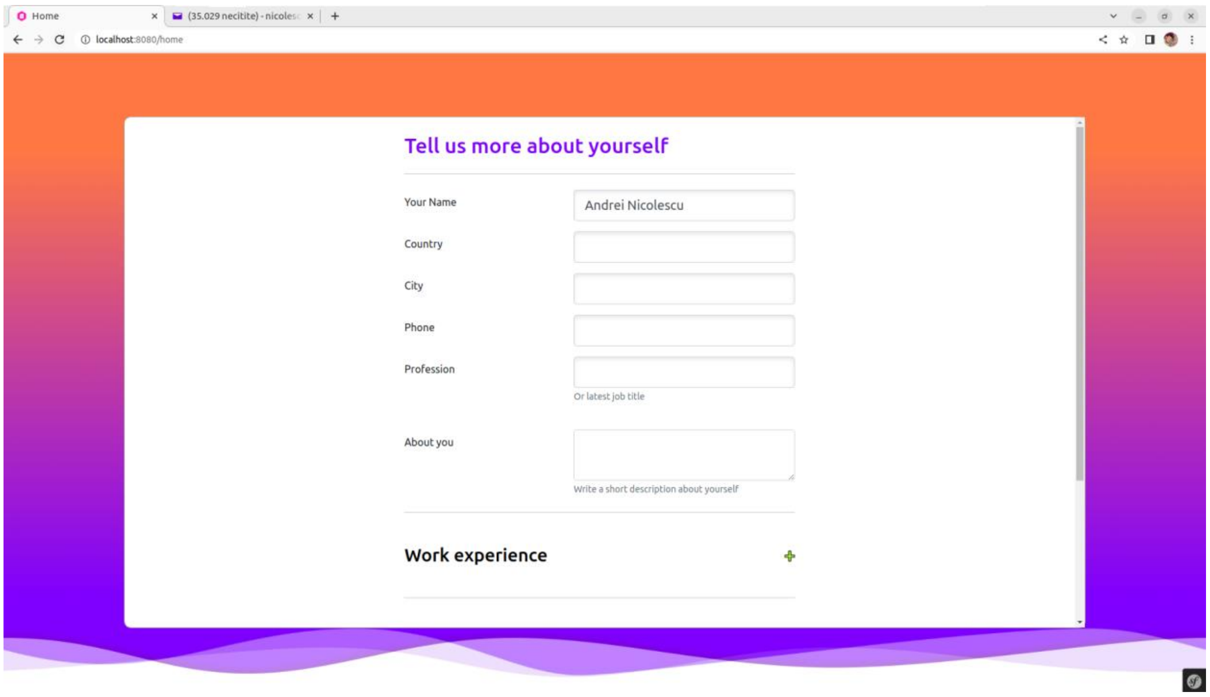
Please check your email and follow the link we sent you to gain access here.

Didn't receive it? Resend now!

After a valid verification process, the user's email address is flagged as verified by the application and the user can freely enjoy all its functionalities. This process must be completed only once per account.

6.5 Creation Page

After verifying the email address, the user is redirected automatically to the creation page.


Home | (35.029 necitite) - nicoles

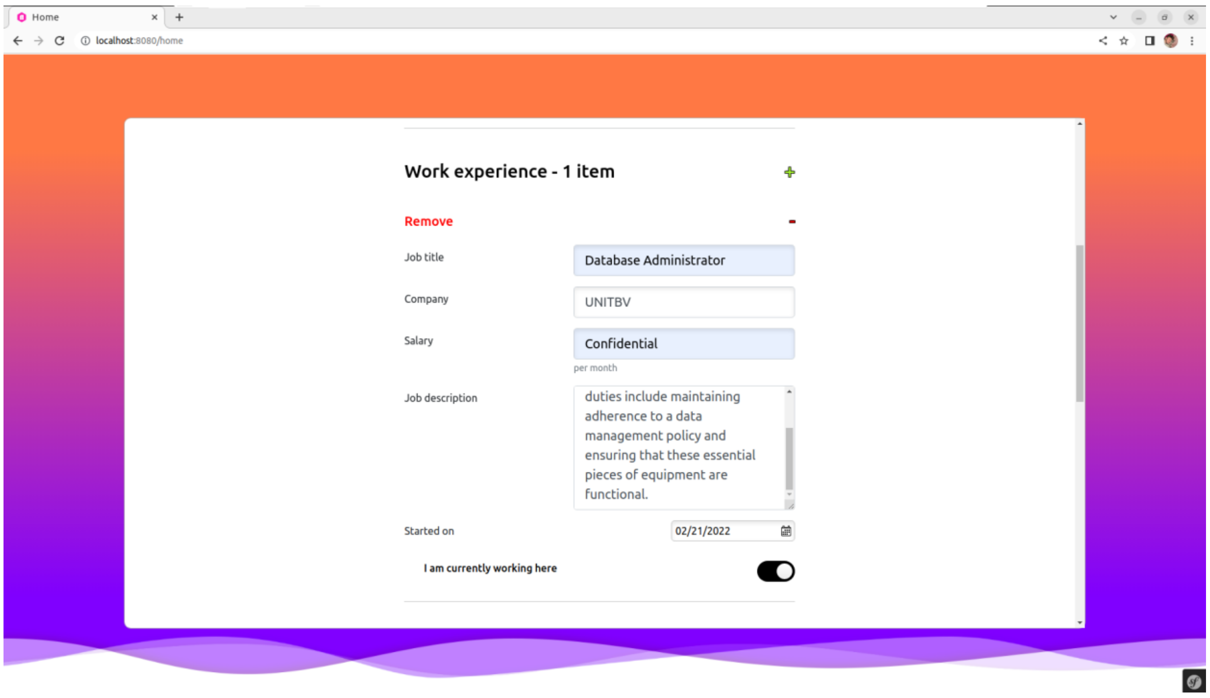
localhost:8080/home

Tell us more about yourself

Your Name	Andrei Nicolescu
Country	
City	
Phone	
Profession	Or latest job title
About you	Write a short description about yourself

Work experience +

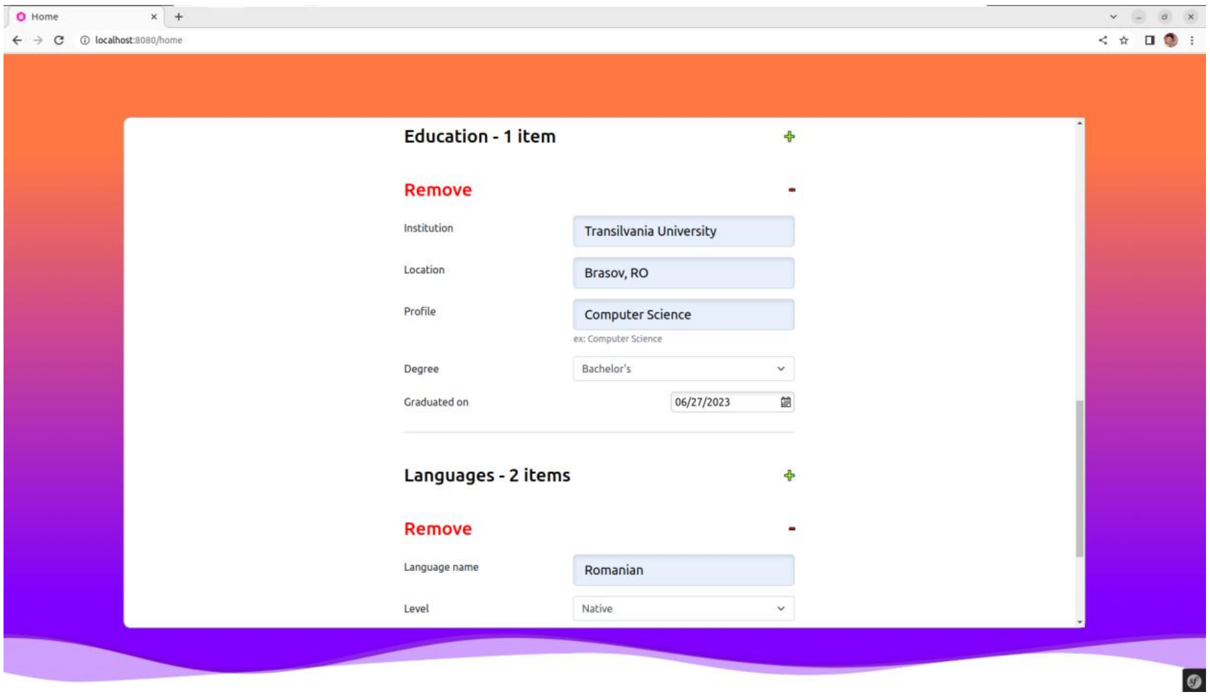
This page is composed by a detailed form in which the user must input mandatory personal information such as first and last name, job title, a short description but also optional information such as work experience and spoken languages.


Home
localhost:8080/home

Work experience - 1 item

Remove

Job title	Database Administrator
Company	UNITBV
Salary	Confidential per month
Job description	duties include maintaining adherence to a data management policy and ensuring that these essential pieces of equipment are functional.
Started on	02/21/2022
I am currently working here	


Home
localhost:8080/home

Education - 1 item

Remove

Institution	Transilvania University
Location	Brasov, RO
Profile	Computer Science ex: Computer Science
Degree	Bachelor's
Graduated on	06/27/2023

Languages - 2 items

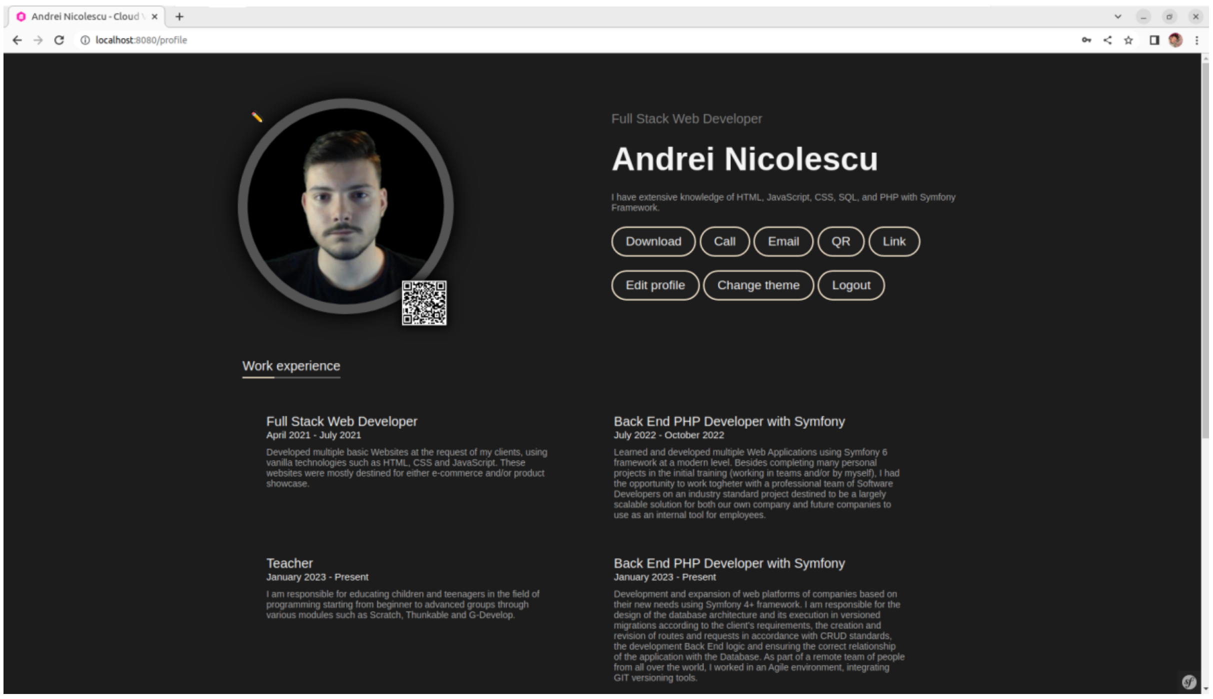
Remove

Language name	Romanian
Level	Native

To submit this form after completing it, there is a button at the bottom of the page entitled “Generate!” which must be clicked. Until this form is submitted for the first time, the user will be always redirected to the creation page since until this process is completed, there will not be enough information to create a CV for the user.

6.6 CV Page

After correctly completing and submitting the creation page form, the user is redirected to the CV page, where a default styling is applied. This is essentially the main page of the application after the registration process is finished.



Full Stack Web Developer

Andrei Nicolescu

I have extensive knowledge of HTML, JavaScript, CSS, SQL, and PHP with Symfony Framework.

Work experience

Full Stack Web Developer
April 2021 - July 2021
Developed multiple basic Websites at the request of my clients, using vanilla technologies such as HTML, CSS and JavaScript. These websites were mostly destined for either e-commerce and/or product showcase.

Back End PHP Developer with Symfony
July 2022 - October 2022
Learned and developed multiple Web Applications using Symfony 6 framework at a modern level. Besides completing many personal projects in the initial training (working in teams and/or by myself), I had the opportunity to work togheter with a professional team of Software Developers on an industry standard project destined to be a largely scalable solution for both our own company and future companies to use as an internal tool for employees.

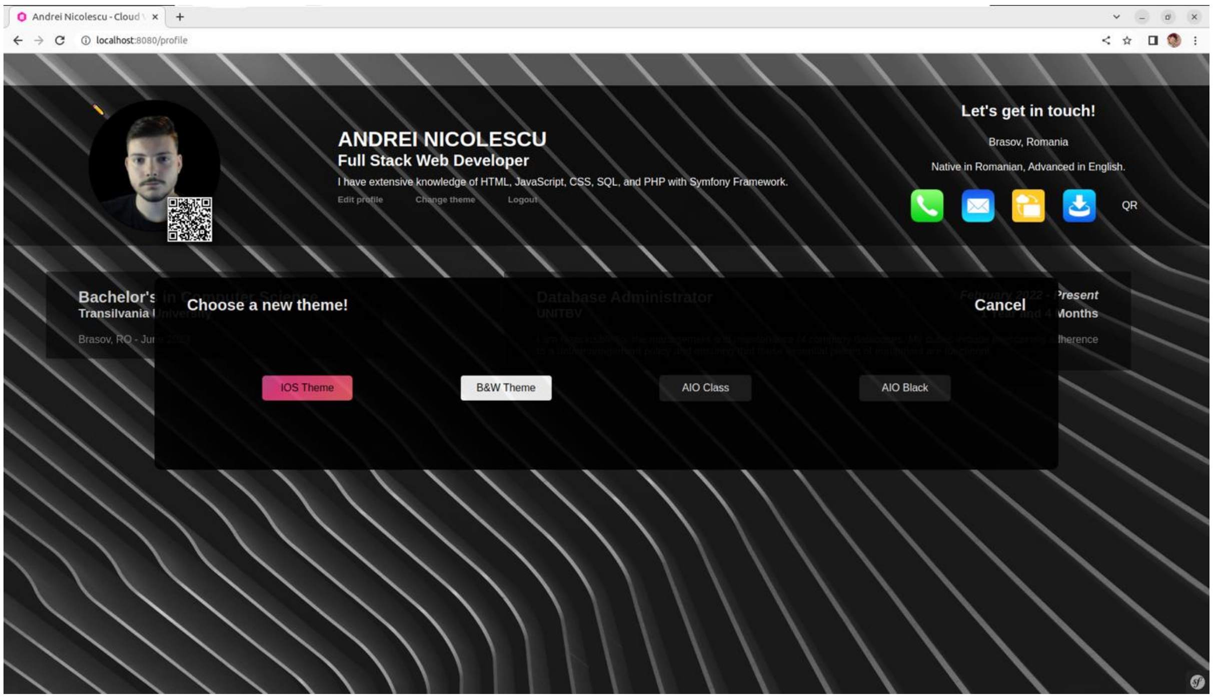
Teacher
January 2023 - Present
I am responsible for educating children and teenagers in the field of programming starting from beginner to advanced groups through various modules such as Scratch, Thunkable and G-Develop.

Back End PHP Developer with Symfony
January 2023 - Present
Development and expansion of web platforms of companies based on their new needs using Symfony 4+ framework. I am responsible for the design of the database architecture and its execution in versioned migrations according to the client's requirements, the creation and revision of routes and requests in accordance with CRUD standards, the development Back End logic and ensuring the correct relationship of the application with the Database. As part of a remote team of people from all over the world, I worked in an Agile environment, integrating GIT versioning tools.

From this page the user can change the styling of the CV document, download it as a PDF, get a cloud hosting link, download a QR code that redirects to the cloud hosting, change his information and logout. It automatically renders a page based on the chosen design to encompass all the information provided by the user in the most pleasant and easy to read way possible.

6.6.1 Changing the Styling

To change the styling of your Cloud-Vitae, you have to click the “Changed theme” button. The positioning and look of this button may differ from style to style, but it consistently has the same name and is generally located in the top right corner of the page.



Once it is clicked, a pop-up containing all the possible themes will appear. From here, the user can choose the desired theme, and can decide which fits best as each theme has a differently styled button that best describes it.

6.6.2 Downloading a PDF Copy

To download your Cloud-Vitae as a PDF document, you have to click the “Download PDF” button. The positioning and look of this button may differ from style to style, but it consistently has the same name and is generally located in the top right corner of the page. Once it is clicked, the PDF copy of you Cloud-Vitae will immediately start downloading.

6.6.3 Sharing a Cloud Hosting Link

To copy your Cloud-Vitae's cloud hosting link, you must press the “Copy Link” button. The positioning and look of this button may differ from style to style, but it consistently has the same name and is generally located in the top right corner of the page. Once this button is clicked, the cloud link is copied to the clipboard automatically, and the user can paste it wherever it is needed.

6.6.4 Sharing a QR Code Link

To download your Cloud-Vitae's QR Code, you must click the “Get QR” button. The positioning and look of this button may differ from style to style, but it consistently has the same name and is generally located in the top right corner of the page. Once clicked, a PDF document containing the QR Code will immediately start downloading. The user can either use it as it comes or take the picture and inserted wherever it is needed (i.e. personal blog, social media description, email template).

6.6.5 Changing User Information

To change your Cloud-Vitae's information by clicking the “Edit Profile” button. The positioning and look of this button may differ from style to style, but it consistently has the same name and is generally located in the top right corner of the page. After clicking this button, the user will be redirected to the creation page, except that this time the form is already filled with the previously imputed information. This way the user can in more easily edit his data in an all-encompassing page.

6.6.6 Changing the Profile Picture

You can change your profile picture in Cloud-Vitae by clicking the crayon icon in the top left corner of your already existing profile picture. The positioning and look of this button is always the same regardless of the chosen style. Upon clicking this button, the media input immediately opens, and the user can browse his personal files and choose a new image. After selecting the new profile picture and clicking on “Select”, the application automatically replaces the old profile picture with the newly selected one.

6.6.7 Logging out

If the user decides to log out of his account, he can easily do so by clicking the “Logout” button. The positioning and look of this button may differ from style to style, but it consistently has the same name and is generally located in the top right corner of the page. Upon clicking this button, the user will be logged out and redirected to the landing page.

7. Development directions and Conclusion

The application addresses the complexities and limitations associated with creating, sharing, and editing CV documents. It achieves this by providing a user-friendly interface that automates layout and design, eliminating the need for specialized software or design skills. Cloud-Vitae's purpose is to achieve this on a continuous manner. To do this, the application must be in a constant state of developing and deploying new functionalities, improvements and features.

One of the main areas in which the application is under-developed at the moment is the collection of CV themes. It is essential that the users have enough stylings to choose from, and this demand will only grow as the user base extends. Cloud-Vitae's plan is to exponentially increase its selection of CV themes to become large enough so that all users feel like they can choose a preferred theme while also remaining unique.

The second domain in which Cloud-Vitae must progress is the number of supported authentication methods and platforms. To reach the largest user base possible the application must support authentication methods with as many platforms as possible to remain an attractive solution for new users on the market. The next targeted platforms to integrate are Google and Facebook, which are commonly used in web applications in this scope and can bring a great number of new users to Cloud-Vitae.

Another area of possible new horizons for Cloud-Vitae is the CV extraction functionality. This implies that when a user registers for an account, he can upload his old CV document and the application will automatically extract and group all the information, without needing the user to input anything else in the application. Similar solutions already exist, but they have proven extremely unreliable over the years, as the way CV documents are created differs greatly and therefore applications have a hard time extracting the information based on patterns alone. Cloud-Vitae aims to revolutionize this process in the future by integrating an external API based AI solution such as Chat GPT to extract and return the information from the CV uploaded to the application, but at the moment these external APIs are paid, and the initial development budget of Cloud-Vitae was not enough to cover such expenses.

The last major way in which Cloud-Vitae could evolve even further is developing its own API intended for the integration and use by other applications and developers. Especially after the extraction functionality is implemented, the application will be such a demanded solution that aiming for a proprietary API which will allow other applications to use its unique features is the best course of action. This API will encompass all of Cloud-Vitae's functionalities and will be a “pay per request” type of interface, which will also constitute the applications main source of income and monetization.

Cloud-Vitae endeavors to establish standard practices within the realm of online employment by promoting the adoption of novel approaches, and it is on the right path to achieve this goal. Envisioning the current state of the application combined with its development direction, Cloud-Vitae's aim to revolutionize the employment world with its unique features becomes more and more tangible.

Bibliography and Webography

1. “The lean startup” by Eric Ries, 2011

2. “Agile Software Development, Principles, Patterns, and Practices” by Robert C. Martin, 2002

3. “Manifesto for Agile Software Development”, from https://agilemanifesto.org/

4. “What is a programming language?” https://codeinstitute.net/global/blog/what-is-a-programming-language

5. Symfony documentation, from https://symfony.com/doc/current/index.html

6. Composer documentation, from https://getcomposer.org/doc/01-basic-usage.md

7. Doctrine in Symfony, from https://symfony.com/doc/current/doctrine.html

8. Symfony’s Webpack Encore, from https://symfony.com/doc/current/frontend.html

9. Twig documentation, from https://twig.symfony.com/

10. PHP Storm, from https://www.jetbrains.com/phpstorm/

11. “What is MySQL?”, from https://www.oracle.com/mysql/what-is-mysql/

12. “Version control with GIT” by Jon Loeliger, from https://www.foo.be/cours/dess-20122013/b/OReilly%20Version%20Control%20with%20GIT.pdf

13. “Object-Oriented PHP”, by Peter Lavin from https://doc.lagout.org/programmation/tech_web/No.Starch.Press.Object.Oriented.PHP.Concepts.Techniques.and.Code.210pp.6-2006.pdf

14. Docker documentation, from https://docs.docker.com/get-started/overview/

15. NGINX documentation, from https://docs.nginx.com/nginx/admin-guide/web-server/