Meta Scifor Technologies

Major Project

CV Application

Abdullah Zeeshan

B2 Batch

November 1, 2024

Abstract

This report outlines the development process of the CV Application, detailing the technologies used, key features, challenges faced, and potential improvements. The CV Application is designed to help users create professional CVs effortlessly and export them as PDFs.

Contents

Chapter 1

Introduction

1.1 Project Overview

The CV Builder project aims to provide users with a platform to create and manage their CVs easily. The application allows users to input personal details, education, experience, skills, and projects. It also offers PDF generation capabilities, ensuring that users can export their CVs for professional use.

1.2 Key Features

The CV Builder application includes several key features:

- User input forms for various CV sections (personal info, education, experience, skills, projects).
- Real-time updates to the CV preview.
- PDF export functionality with custom styling and fonts.

1.3 Objectives

The main objectives of this project are:

- Develop a user-friendly interface for inputting CV information.
- Ensure real-time updates of the CV preview.
- Implement PDF generation to facilitate easy sharing and printing.

Chapter 2

Technologies Used

2.1 Frontend

The front end of the application is built using the following technologies:

- **React**: A JavaScript library for building user interfaces.
- **React-PDF**: A library for creating PDF documents in React.
- **CSS**: For styling the application and ensuring a responsive design.

2.2 Backend

While this project does not include a backend, the architecture can be easily extended to incorporate server-side functionality using Node.js and Express.

Chapter 3

Dependencies and Development Tools

This project leverages a selection of dependencies and development tools to ensure smooth functionality and maintainability. Each tool and library used is described below to provide insight into its role and purpose within the CV Application.

3.1 Project Dependencies

The following dependencies were essential for developing and rendering the CV Application:

- **React (v18.3.1)**: The core library for building the user interface, enabling a component-based architecture.
- **React-DOM (v18.3.1)**: Facilitates the interaction between React components and the DOM, rendering React components in the browser.
- **React-PDF Renderer (v4.0.0)**: A library that allows for the creation of dynamic, custom-styled PDF documents directly in React.
- **React-to-Print (v3.0.2)**: Provides a straightforward way to print React components or save them as PDFs, which is instrumental in offering users the ability to export CVs.
- **@fortawesome/free-brands-svg-icons and @fortawesome/free-solid-svg-icons (v6.6.0)**: Font Awesome icon libraries that enhance the UI by incorporating popular icon sets.
- **@mdi/js and @mdi/react**: Material Design Icon packages provide alternative icons for flexibility and enhance visual presentation in the application.
- **html2pdf.js (v0.10.2)**: Offers additional PDF generation options by converting HTML content to a PDF, facilitating easy CV export.

3.2 Development Dependencies

To maintain code quality and streamline development, several development dependencies were used:

- **Vite (v5.4.8)**: A fast development server that bundles the application efficiently, improving development speed and experience.
- **ESLint and ESLint Plugins (React, React Hooks, React Refresh)**: ESLint (v9.11.1) along with its plugins for React, React Hooks, and React Refresh ensures that the code adheres to best practices and maintains consistency.
- **Type Definitions for React and React-DOM**: Type definitions improve code reliability by adding static type-checking for React and React-DOM, ensuring smoother integration with TypeScript or JavaScript in an IDE with IntelliSense support.
- **@vitejs/plugin-react (v4.3.2)**: Vite's official React plugin offers support for fast Hot Module Replacement (HMR) and efficient bundling for React, allowing for a seamless development experience.

Each of these dependencies and tools played a critical role in achieving the project's objectives by enhancing functionality, enforcing code quality, and improving development speed. Future enhancements to the CV Application may include expanding this toolkit to support additional features or improve performance further.

Chapter 4

React Components

The CV Application is structured around a set of React components that work together to provide an interactive and efficient user interface. Each component is designed to fulfill a specific function within the application, contributing to the modularity and maintainability of the project. This chapter provides an overview of all the components and in-depth look at the core ones.

4.1 A Glance

Here is an overview of all the components.

4.1.1 CVPreview Component

The `CVPreview.jsx` component is responsible for displaying a real-time preview of the user's CV as they input data. It updates dynamically based on user input from the `UserInput` component, allowing users to see changes immediately. Key features include:

- Mapping user input data to visual sections (personal info, education, experience).
- Styling for consistency with the exported PDF version.
- Conditional rendering for optional sections.

4.1.2 CVToPDF Component

The `CVToPDF.jsx` component handles the conversion of the CV preview into a downloadable PDF document. Utilizing libraries like `html2pdf.js` or `React-PDF Renderer`, this component captures the rendered preview and exports it as a PDF. Key features include:

- Compatibility with `React-PDF Renderer` for custom styling.
- PDF export button with customization options.
- Handling of font and layout adjustments for PDF compatibility.

4.1.3 Footer Component

The `Footer.jsx` component provides a consistent footer for the application, typically containing essential links or attributions. Although this component is simple, it contributes to a professional, cohesive layout. Key features include:

- Display of links to external resources (e.g., GitHub, documentation).
- Simple, responsive styling.

4.1.4 Header Component

The `Header.jsx` component creates a unified header for the application, often including the application title, logo, or navigation options. Key features include:

- Display of the CV Application title and branding elements.
- Possible integration of a navigation bar for ease of use.
- Responsive design adjustments for different screen sizes.

4.1.5 Main Component

The `Main.jsx` component serves as the primary container for the application content. It structures the layout, encapsulating both the `UserInput` and `CVPreview` components within a flexible grid or flex layout. Key features include:

- Managing layout and positioning for other components.
- Applying responsive styles to adapt to various screen sizes.

4.1.6 UserInput Component

The `UserInput.jsx` component serves as the primary interface for users to input and manage data for the CV application. It provides form fields to capture personal details, education, experience, projects, and skills, supporting both dynamic input and PDF generation. This component includes fields for each CV section with buttons to add or remove entries and upload sample data.

- Collects essential information from the user, such as name, contact details, and experience, to dynamically populate the CV.
- Ensures all required fields are filled out correctly, providing real-time feedback to the user for missing or incorrect inputs.
- Instantly updates and reflects the entered information on the CV preview, offering a seamless, interactive user experience.

4.2 UserInput Component

The `UserInput.jsx` component gathers all user inputs for the CV. It includes forms for various CV sections such as personal information, education, experience, and skills. Key features include:

4.2.1 Component Structure

- **State Management:** Local state variables manage the expanded or collapsed views for each section (General Information, Education, Experience, Projects, and Skills).
- **Functions:**
  - `clearForm()`: Clears the form inputs by resetting all state fields.
  - `loadDefaultData()`: Populates form fields with default data from the `sampleCV` object.

4.2.2 Form Sections

Each section is represented as a `fieldset` and wrapped in `legend` elements, making them collapsible. Users can toggle the visibility of each section by clicking on the respective `legend`.

General Information

The general information fields capture the user's name, email, and phone number using basic form inputs. The state variable `isGeneralInfoOpen` controls the visibility of this section.

Education

The education section captures information on the user's academic background. It includes fields for the school name, title of study, and dates of study. Each education entry is rendered with:

- A unique set of input fields, generated using the `education` array mapped from the `userData` state.
- A button to add new entries via the `addNewSection` function.
- A delete button for each entry to allow removal via the `removeSection` function.

Experience

The experience section includes fields for company name, position, responsibilities, and the employment dates. Each entry also includes:

- Expandable text areas for responsibilities.
- `addNewSection()` and `removeSection()` functions, similar to the education section.

Projects

Projects contain fields for title, description, technologies, and year. The technologies input accepts comma-separated values, processed into an array using `handleArrayInputChange()`. Like the education and experience sections, the projects section is dynamically extendable.

Skills

Skills are managed as a list of individual text inputs that represent each skill. Users can add or remove skills with the functions:

- `addNewSkill()`: Adds a new input field to the skills array.
- `removeSkill()`: Removes the selected skill entry.

4.2.3 File Download

A `PDFDownloadLink` from the `@react-pdf/renderer` library is integrated, generating a downloadable PDF file of the CV based on the current `userData` state.

4.2.4 Conclusion

The `UserInput.jsx` component is a central part of the CV application, allowing users to comprehensively input and manage their personal data. With features for adding, editing, and removing entries across various CV sections, it provides an interactive user experience and offers a streamlined method to generate a downloadable CV document in PDF format.

4.3 CVPreview Component

The `CVPreview.jsx` component is responsible for rendering a preview of the CV based on the user data passed to it. The component takes `userData` as a prop, which contains the user's personal information, education, experience, projects, and skills. It displays this data in a structured and styled format, providing a summary of the user's background.

4.3.1 Component Structure

The component follows a simple layout that includes the following sections:

- **Header:** Displays the user's name and contact information (email and phone).
- **Education Section:** Displays the user's education details, including the school name, degree title, and dates.
- **Experience Section:** Lists the user's work experience, including company name, position, dates, and responsibilities.
- **Projects Section:** Displays a list of projects, including the project title, technologies used, and project description.
- **Skills Section:** Shows the user's skills as a comma-separated list.

4.3.2 State Management

The component does not have its own state. It receives all the necessary data as `userData` from its parent component. The `userData` object is expected to have the following properties:

- `name`: The user's full name.
- `email`: The user's email address.
- `phone`: The user's phone number.
- `education`: An array of education objects, where each object contains `school`, `title`, and `date`.
- `experience`: An array of experience objects, where each object contains `company`, `position`, `fromDate`, `toDate`, and `responsibilities`.
- `projects`: An array of project objects, where each object contains `title`, `technologies`, and `description`.
- `skills`: An array of strings representing the user's skills.

4.3.3 Rendering Logic

- The header section displays the user's `name` and `email` and `phone` as contact information.
- The education section checks if there is any education data. If data is available, it is displayed with the school name, title, and dates. If no education data is provided, a message "No education details provided" is shown.
- The experience section works similarly, displaying the user's job experiences if available, or showing "No experience details provided" if the experience data is missing.
- The projects section lists each project with the project title, technologies used, and description, or displays "No projects provided" if no project data exists.
- The skills section lists the user's skills as a comma-separated string. If no skills are available, it displays "No skills provided."

4.3.4 Conclusion

The `CVPreview.jsx` component provides a well-structured, easy-to-read preview of the user's CV. By dynamically rendering the user's personal information, education, experience, projects, and skills, it gives the user a final look at their CV before generating a downloadable PDF. This component plays a key role in the user's experience of generating and reviewing their CV in the application.

4.4 Main Component

The `Main.jsx` component serves as the central hub for managing the user's CV data. It imports and uses the `UserInput` and `CVPreview` components to allow the user to input their CV details and preview them dynamically. The component utilizes React's `useState` hook to manage the `userData` state, which holds the data for the CV, including personal details, education, experience, projects, and skills.

4.4.1 State Management

The `Main` component manages a `userData` state, which is initialized with sample CV data imported from `sampleCV`. The state is updated through several handler functions based on user input.

The `userData` object contains the following fields:

- `name`: The user's name.
- `email`: The user's email address.
- `phone`: The user's phone number.
- `education`: An array of education objects.
- `experience`: An array of experience objects.
- `projects`: An array of project objects.
- `skills`: An array of the user's skills.

4.4.2 Handler Functions

The component includes the following handler functions to manage changes to the user data:

- `handleInputChange`: Updates individual fields in the `userData` state when a user types in a text input. It uses the `name` attribute of the input field to identify which property to update.
- `handleArrayInputChange`: Updates array-based fields (education, experience, projects) in `userData`. It accepts additional arguments for `section` (the array to update) and `index` (the index of the item to update).
- `addNewSection`: Adds a new section to `userData` (education, experience, or projects). A default value is set based on the section type.
- `removeSection`: Removes a specific entry from an array-based section (education, experience, or projects) by filtering the array based on the given index.
- `handleSkillsChange`: Updates the user's skills array when an individual skill is changed.
- `addNewSkill`: Adds a new empty skill to the user's skills array.
- `removeSkill`: Removes a skill from the skills array by filtering out the item at the given index.

4.4.3 Component Structure

The `Main` component renders two child components:

- `UserInput`: A component responsible for gathering the user's input for each section of the CV. It receives the necessary props for updating the `userData` state, adding/removing sections, and updating skills.
- `CVPreview`: A component responsible for displaying the current state of the user's CV in a preview format. It receives the `userData` state as a prop and renders the user's name, contact details, education, experience, projects, and skills.

4.4.4 Conclusion

The `Main.jsx` component serves as the central manager for the CV data. It provides the necessary state and handler functions for updating and managing user input. By linking the `UserInput` and `CVPreview` components, it allows the user to both enter and preview their CV information dynamically. This component is essential for maintaining the flow of data throughout the CV creation process.

Chapter 5

Sample CV Data

The `sampleCV` object represents the mock CV data used within the application. It contains predefined user information, such as personal details, educational history, professional experience, project contributions, and skills. This data serves as a sample for the CV builder and is used to populate fields for preview and editing in the application.

5.1 Personal Information

The `sampleCV` object includes basic personal details, such as:

- `name`: Hermione Granger
- `email`: hermione.granger@hogwarts.edu
- `phone`: 555-123-4567

These fields provide the user's contact information, which will be displayed at the top of the CV preview.

5.2 Education

The `education` array within `sampleCV` contains two entries detailing Hermione Granger's academic history:

- **Hogwarts School of Witchcraft and Wizardry**: Completed the *N.E.W.T. in Charms, Potions, and Defense Against the Dark Arts* from 1991 to 1998.
- **The Ministry of Magic**: Undertook *Postgraduate Studies in Magical Law* from 1999 to 2000.

Each educational entry includes the name of the institution, the degree or title obtained, and the duration of the study.

5.3 Professional Experience

The `experience` array provides details of Hermione Granger's professional journey, showcasing her diverse roles:

- **Ministry of Magic**: Junior Assistant to the Minister of Magic (September 2000 - June 2001). Responsibilities included drafting policies to improve magical community relations and ensuring compliance with magical regulations.
- **The Quibbler**: Contributing Writer (July 2001 - December 2003). Focused on writing articles related to magical creatures and legal issues, conducting interviews with key figures in the wizarding world.
- **Hogwarts School of Witchcraft and Wizardry**: Defense Against the Dark Arts Professor (January 2004 - Present). Taught students advanced defensive techniques, organized extracurricular activities, and developed curriculum.

Each experience entry contains the company name, the position held, the dates of employment, and a brief description of the responsibilities.

5.4 Projects

The `projects` array in `sampleCV` highlights two major contributions by Hermione Granger:

- **S.P.E.W.**: An organization she founded to promote the rights of house-elves and improve their working conditions. The project involved advocacy, public speaking, and community engagement.
- **Hogwarts Library Organization**: A project to reorganize the Hogwarts library system, aimed at improving access to resources for students and faculty. It involved project management, research, and collaboration.

Each project entry includes the title, a description, and a list of technologies or skills utilized.

5.5 Skills

The `skills` array lists the key skills that Hermione Granger possesses. These include:

- Charms
- Potions
- Magical Law
- Public Speaking
- Research
- Team Leadership

These skills represent Hermione's expertise in various areas of magic and her abilities in leadership and communication.

5.6 Conclusion

The `sampleCV` object serves as a foundational mock-up for the CV building application. It contains realistic and structured data across multiple sections—personal details, education, experience, projects, and skills—that will be displayed and editable within the user interface. This sample data is essential for demonstrating the functionality of the CV generation process.

Chapter 6

PDF Generation

In this application, the React-PDF library is utilized to render the CV as a downloadable PDF document. This approach allows us to customize the look and feel of the CV while integrating seamlessly with the React components. The process involves dynamic generation of the document content based on user input, which can be easily customized using React's declarative style. The PDF generation functionality is encapsulated in the `CVToPDF` component, where various sections of the CV (e.g., Education, Experience, Projects, Skills) are styled and displayed according to the user's data.

6.1 React-PDF Library

The application uses the `React-PDF` library, which provides several key features:

- **Custom Styling**: The PDF content can be customized with specific styles for fonts, spacing, and layout. This ensures that the final document matches the design and structure of the user's CV.
- **Dynamic Adjustments**: The content of the PDF is dynamically generated based on the user's input, ensuring that the CV reflects the most current and relevant information.
- **Integration with React**: React-PDF enables easy integration with React components, allowing developers to manage the layout and design as they would for regular UI components.

6.2 Code Overview

The key functionality of the PDF generation is implemented in the `CVToPDF` component, which takes the user data (`userData`) as a prop and renders the content using React-PDF's `Document`, `Page`, `Text`, and `View` components.

6.2.1 Font Registration

Fonts are registered using the `Font.register` method to allow custom fonts in the PDF. In this case, the `Spectral` and `Spectral SC` font families are used for styling the text. The relevant font files are linked from the public directory and include regular, bold, and italic styles for each font family.

```
Font.register({
    family: 'Spectral',
    fonts: [
        { src: '/fonts/Spectral/Spectral-Regular.ttf' },
        { src: '/fonts/Spectral/Spectral-Bold.ttf', fontWeight: 'bold' },
        { src: '/fonts/Spectral/Spectral-Italic.ttf', fontStyle: 'italic' }
    ],
});

Font.register({
    family: 'Spectral SC',
    fonts: [
        { src: '/fonts/Spectral_SC/SpectralSC-Regular.ttf' },
        { src: '/fonts/Spectral_SC/SpectralSC-Bold.ttf', fontWeight: 'bold' },
    ],
});
```

6.2.2 Styling the Document

The `StyleSheet.create` function is used to define the styles for various elements in the document. The following styles are defined:

- `page`: Sets padding, font family, font size, and line height for the entire document.
- `title`: Styles the title section (e.g., the name of the individual) with larger font size, bold weight, and centered alignment.
- `header`: Styles the header section to center the content and display contact information in a row format.
- `sectionTitle`: Defines styles for the titles of different sections (e.g., Education, Experience).
- `entry`: Defines the style for each entry (e.g., school or company name) within a section.
- `horizontalLine`: Adds a horizontal line between sections for visual separation.
- `fr`: A flex container used for sections where the content is split into two columns (e.g., the job position and dates in the Experience section).

6.2.3 Rendering the Content

The `CVToPDF` component renders the content within a `Document` component. The document contains a `Page` element, which serves as the individual page in the PDF. Here's a breakdown of how the data is rendered:

- **Personal Information**: The user's name, email, and phone number are displayed at the top of the document, using the `title` and `contact` styles.

- **Education**: Each education entry is rendered in the `EDUCATION` section. The school name, title, and date are displayed, and if no data is provided, a fallback message is shown.
- **Experience**: The user's work experience is rendered under the `EXPERIENCE` section. For each job, the company name, position, dates of employment, and responsibilities are displayed.
- **Projects**: Projects are listed in the `PROJECTS` section, with the title, technologies used, and a description of each project.
- **Skills**: The user's skills are listed in the `SKILLS` section, with each skill separated by commas.

6.2.4 Example of Data Rendering

The following code snippet illustrates how an education entry is rendered:

```
<Text style={styles.entryTitle}>{edu.school}</Text>
<View style={styles.fr}>
    <Text style={[styles.entryText, { fontStyle: 'italic' }]}>{edu.title}</Text>
    <Text style={styles.entryText}>{edu.date}</Text>
</View>
```

This snippet displays the school name as the title, followed by the degree or title (in italics), and the date of attendance. If the education data is empty, a message is shown to indicate that no education details are provided.

6.3 Conclusion

The integration of the React-PDF library into this application enables the dynamic generation of customized CVs in PDF format. The use of custom fonts, styles, and layout ensures that the generated PDF is professional and tailored to the user's input. The seamless integration with React components ensures that changes to the user data are automatically reflected in the document. This feature significantly enhances the user experience, allowing users to download their CVs in a polished, ready-to-share format.

Chapter 7

Deployment

7.1 Introduction

The deployment of the site was completed using **Vercel**, a platform optimized for front-end frameworks like React. Vercel provides a smooth and efficient deployment process, integrated with version control systems such as GitHub. By using Vercel, the site is automatically built and deployed with every push to the connected repository, making the process fast and reliable.

7.2 Why Vercel Was Chosen

Vercel was chosen as the hosting platform due to the following reasons:

- **Automatic Deployment**: Vercel integrates with GitHub repositories, enabling automatic deployment whenever new commits are pushed. This makes continuous deployment simple and fast.
- **Server-Side Rendering Support**: Vercel offers support for React apps, including the ability to render pages server-side. This ensures fast initial loading and SEO optimization.
- **Zero Configuration**: Vercel provides a zero-config deployment experience. It automatically detects the framework being used and configures the deployment environment accordingly.
- **Scalability**: Vercel can handle scalable applications with serverless functions. As the application grows, the platform can scale effortlessly.
- **Free Tier**: Vercel offers a generous free tier with unlimited deployments, which is suitable for smaller projects like this one.
- **Custom Domains and SSL**: Vercel provides the ability to add custom domains and includes free SSL certificates for secure connections, enhancing both usability and security.

7.3 Deployment Process on Vercel

The process of deploying the application on Vercel was straightforward and involved the following steps:

1. **Connecting the GitHub Repository**: The first step was to connect the GitHub repository containing the code to Vercel. This allows Vercel to access the repository and deploy it automatically.
2. **Choosing the Project**: After connecting the repository, Vercel automatically detected that the project was a React application and set up the necessary build commands.
3. **Setting the Build and Output Directories**: Vercel automatically detected the build output directory as 'build/' for React. In case of custom configurations, this could have been manually adjusted.
4. **Deployment Confirmation**: After configuration, Vercel built the project and deployed it. The deployment process was completed without any manual interventions required.
5. **Automatic Updates**: Every time changes are pushed to the repository, Vercel triggers a new build and deploys the latest version automatically.

The site was successfully deployed and is now live on a Vercel-provided URL, with the option to add a custom domain for better branding and accessibility.

7.4 Why GitHub Pages Was Not Used

While GitHub Pages is a popular and easy-to-use platform for deploying static websites, it was not suitable for this project due to the following reasons:

- **Static Site Limitation**: GitHub Pages is designed to host only static sites. This means that it cannot support dynamic functionalities, such as server-side rendering (SSR) or API calls that require a back-end server. Since the application involves client-side rendering (CSR) with React, Vercel's support for dynamic content and serverless functions was essential for the deployment.
- **No Support for Server-Side Rendering (SSR)**: React, when used with frameworks like Next.js, often requires SSR for improved performance and SEO optimization. GitHub Pages cannot handle SSR because it is strictly limited to serving static files, which would prevent the app from functioning optimally.
- **Custom Domain and SSL Challenges**: While GitHub Pages allows custom domains, the process is more manual and doesn't provide automatic SSL certificates like Vercel does. With Vercel, SSL certificates are automatically handled, ensuring secure connections without any extra configuration.
- **Complex Build Process**: Although GitHub Pages can be configured to deploy from the 'build/' directory (by using GitHub Actions), this process requires additional configuration and scripting. Vercel, on the other hand, automates this process and handles it seamlessly.

- **No Backend Integration**: GitHub Pages is purely for static content. In this project, if there was a need to integrate a back-end or API, GitHub Pages would not be suitable because it cannot host server-side applications or serverless functions, which Vercel can handle easily.

For these reasons, Vercel was chosen as the platform for deployment instead of GitHub Pages, as it offers the flexibility, scalability, and server-side capabilities needed to deploy dynamic React applications.

7.5 Conclusion

By choosing Vercel, the deployment process was significantly streamlined, and the site is now live with minimal setup required. The automatic deployment feature, scalability, and built-in support for modern web applications made Vercel an ideal choice over GitHub Pages, which would have limited the application's potential with its static site constraints.

Chapter 8

Challenges Faced

The following challenges were encountered during development:

- Ensuring the CV preview updates dynamically as users input data.
- Handling form validation and error messages effectively.
- Maintaining consistent styling across various components and screen sizes.

Chapter 9

Future Improvements

Future enhancements for the CV Builder may include:

- Additional templates for CV designs.
- Improved user experience through animations and transitions.
- A feature for saving CVs and retrieving them later.

Chapter 10

Acknowledgments

I would like to express my gratitude to:

- My mentor Lekha Savale Maam for her guidance.
- My fellow mates Sarbajit, Ashwin, and Pratyusha for their support in reviewing and providing feedbacks.
- The open-source community for providing resources and libraries that facilitated this project.

Bibliography

[1] React-PDF Documentation. *React-PDF Library.* Available at: `https://react-pdf.org/`.

[2] Spectral Font. *Google Fonts.* Available at: `https://fonts.google.com/specimen/Spectral`.

[3] React Documentation. *React Library.* Available at: `https://reactjs.org/docs/getting-started.html`.

[4] ReactDOM Documentation. *React Library.* Available at: `https://reactjs.org/docs/react-dom.html`.

[5] React to Print Documentation. *React-to-Print Library.* Available at: `https://www.npmjs.com/package/react-to-print`.

[6] Material Design Icons Documentation. *Material Design Icons.* Available at: `https://materialdesignicons.com/`.

[7] FontAwesome Icons Documentation. *FontAwesome Library.* Available at: `https://fontawesome.com/docs/web`.

[8] Vite Documentation. *Vite Build Tool.* Available at: `https://vitejs.dev/`.

[9] Vercel Documentation. *Vercel Platform.* Available at: `https://vercel.com/docs`.