



A Next.js-Based Application for Crafting ATS-Compatible Resumes

Duc Thai

BACHELOR'S THESIS

April 2024

Software Engineering

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Software Engineering

THAI, DUC

A Next.js-Based Application for crafting ATS-Compatible Resumes

Bachelor's thesis 50 pages, appendices 12 pages

April 2024

The recruitment landscape increasingly relies on Applicant Tracking Systems (ATS), necessitating job seekers to optimize their resumes. This thesis presents a web-based resume builder and parser application tailored to assist users in crafting ATS-optimized resumes. Leveraging modern technologies such as Next.js, Redux Toolkit, TypeScript, TailwindCSS, and ReactPDF Renderer, the application streamlines the resume-building process.

The objectives encompass understanding ATS mechanisms, designing a user-friendly interface, providing customizable resume templates, ensuring accessibility and responsiveness, and evaluating application performance. Through literature review, iterative design and development, usability testing, and performance evaluation, the thesis explores the effectiveness of the application in improving the resume-building experience.

Key outcomes include a fully functional web application prototype, comprehensive documentation, case studies, and recommendations for future enhancements. By addressing the challenges in navigating ATS filters and presenting qualifications effectively, this thesis contributes to advancing resume-building practices.

Keywords: applicant tracking system, resume builder, typescript, redux toolkit

CONTENTS

1	INTRODUCTION	5
1.1	Overview and Rationale	5
1.2	Research Significance	5
2	PROJECT OVERVIEW	6
2.1	Conceptualization of the Project	6
2.1.1	What Does ATS Optimization Mean in Practice?	6
2.1.2	Practical Implementation of ATS-Optimized Resumes	7
2.1.3	Bridging the Gap	7
2.2	The problem and its significance	8
2.2.1	The Challenge of ATS Compatibility	8
2.2.2	Limitations of Conventional Resume	8
2.2.3	Significance in the Digital Age	8
2.2.4	Impact on Career Opportunities	9
2.3	Project goal and expected outcomes:	9
3	TECHNOLOGY INTRODUCTION	11
3.1	The modern Web Development Stack	11
3.1.1	Next.js: Powering Server-side rendering	11
3.1.2	Redux Toolkit: Robust State Management	12
3.1.3	TypeScript: Elevating Development Experience:	13
3.2	Role of Next.js in Modern Web Applications	13
3.2.1	Enhanced Search Engine Optimization (SEO)	13
3.2.2	Efficient Page Routing and Navigation	14
3.3	Advantages of TypeScript in Development	15
3.4	The Utility-First Approach of TailwindCSS	17
3.4.1	Utility-First Philosophy	17
3.4.2	Efficiency and Customization in Styling with TailwindCSS ..	17
3.5	Document Rendering with ReactPDF Renderer	18
3.5.1	Dynamic PDF Generation	18
3.5.2	Customization of Document Layout	18
3.5.3	Responsive Design for Print	19
4	TECHNOLOGY FOUNDATION AND RESUME ESSENTIALS	20
4.1	Crafting Intuitive User Experiences	20
4.1.1	Interface Development: Next.js and Tailwind CSS	20
4.1.2	Code Integrity with TypeScript	20
4.1.3	Responsive Design Techniques	21
4.2	State Management with Redux Toolkit	21

4.2.1	Redux Toolkit for Predictable State Management	21
4.2.2	DevTools Integration for Debugging	22
4.3	Résumé Essentials: Tradition and Digital Adaptation	22
4.3.1	Traditional Formatting Conventions.....	22
4.3.2	Navigating ATS Requirements	23
5	SOLUTION IMPLEMENTATION.....	24
5.1	Crafting Resume with ease:	24
5.2	Optimizing Résumés for ATS Compatibility	25
5.3	Workflow showcase	25
5.3.1	Resume Parsing Workflow	25
5.3.2	Creating Resume from Scratch Workflow.....	28
5.4	User Interactions and Experience Design.....	34
5.4.1	User Onboarding	34
6	DISCUSSION AND ANALYSIS	36
6.1	User Experience Enhancement	36
6.2	ATS Compatibility and Parsing Accuracy.....	37
	REFERENCES	38
	APPENDICES.....	39
	Appendix 1. Code Excerpts	39

1 INTRODUCTION

1.1 Overview and Rationale

The digital transformation has significantly altered the dynamics of the job application process, prompting a fundamental reevaluation of how candidates showcase their qualifications. Amidst this evolution, the thesis sets out to investigate and confront the hurdles encountered by students and job seekers when crafting resumes tailored to the intricate demands of Applicant Tracking Systems (ATS).

At its core, the project endeavours to develop an innovative web application, harnessing a suite of advanced technologies including Next.js, Redux Toolkit, TypeScript, TailwindCSS, and ReactPDF Renderer.

1.2 Research Significance

In the contemporary landscape dominated by ATS, this research assumes critical importance. Its primary objective is to revolutionize the process of resume construction by offering a comprehensive understanding of the complexities inherent in ATS operations.

This study fills a significant void by delivering a solution that enables individuals to fashion resumes finely attuned to excel in the digital employment milieu. While numerous resume-building applications and services exist, they often fall short in addressing the nuances of ATS or fail to provide adequate tools for ensuring seamless compatibility with these systems. Therefore, the project's outcome is poised to bridge this gap by furnishing a holistic solution that not only aids users in crafting professional resumes but also empowers them with the requisite knowledge and resources to navigate the intricacies of ATS effectively.

2 PROJECT OVERVIEW

2.1 Conceptualization of the Project

The genesis of this project lies in recognizing the evolving landscape of job applications. The conceptualization phase involves a deep dive into the challenges faced by students and job seekers, revealing a critical need for a modernized approach to resume creation.

The project envisions a user-centric web application that not only simplifies the process but also ensures resumes are tailored to navigate the intricacies of contemporary Applicant Tracking Systems (ATS). This conceptual foundation establishes the project's mission to empower individuals in transforming their career aspirations into tangible opportunities.

2.1.1 What Does ATS Optimization Mean in Practice?

ATS optimization entails tailoring resumes to align seamlessly with the requirements and preferences of Applicant Tracking Systems. In practical terms, this involves structuring resumes in a manner that ensures easy parsing and interpretation by ATS algorithms. Key aspects of ATS optimization include:

Keyword Optimization: Identifying and strategically incorporating relevant keywords and phrases from job descriptions into resumes to increase visibility and ranking within ATS.

Formatting Compatibility: Adhering to ATS-friendly formatting guidelines, such as using standard fonts, avoiding graphics or images, and organizing content in a clear, hierarchical manner to facilitate easy parsing.

Content Relevance: Ensuring that resume content is tailored to match the job requirements and industry norms, thereby enhancing the likelihood of passing through ATS screening filters.

2.1.2 Practical Implementation of ATS-Optimized Resumes

The resulting resumes from the application are deemed "ATS-optimized" when they meet the following criteria:

Enhanced Keyword Integration: The application intelligently suggests and integrates relevant keywords based on the job description provided by the user, thereby maximizing the resume's visibility to ATS.

Seamless Parsing: Resumes generated by the application are structured in a format optimized for ATS parsing, ensuring that all pertinent information is accurately extracted and categorized by ATS algorithms.

Compatibility Testing: The application offers a built-in feature allowing users to preview their resumes through the lens of various ATS platforms, enabling them to assess and fine-tune the resume's compatibility with different systems.

Overall, the practical implementation of ATS optimization within the application involves a combination of intelligent keyword integration, format compatibility, rigorous testing to ensure that resumes are finely tuned to navigate the complexities of ATS effectively.

2.1.3 Bridging the Gap

The conceptualization phase aims to address a distinct gap in the current landscape of resume-building tools and services. While there are existing platforms available, such as Enhancv.com, that claim to assist users in creating ATS-friendly resumes, our research has identified several shortcomings and unmet needs within this domain.

Existing tools often focus primarily on aesthetic design elements and fail to adequately address the complex requirements of Applicant Tracking Systems (ATS). They may lack robust features for optimizing resumes to ensure compatibility with ATS algorithms, leading to suboptimal parsing and reduced visibility for job seekers. Additionally, many existing platforms require authentication and charge an amount of fee using user's social media and credit cards may cause future problems (information leaking, unwanted spam email etc..)

Therefore, there remains a clear gap in the market for a comprehensive solution that combines technological innovation with user-centric design principles to empower individuals in crafting resumes that not only adhere to industry standards but also maximize their chances of success in a highly competitive job market. By leveraging cutting-edge technologies and incorporating insights from user feedback and industry research, our project seeks to fill this gap and provide users with a tool that offers unparalleled support in navigating the complexities of the modern job application process.

2.2 The problem and its significance

2.2.1 The Challenge of ATS Compatibility

ATS, while streamlining recruitment processes, introduces a challenge for applicants. Resumes must not only capture attention visually but also align with the intricate algorithms of these systems to avoid being filtered out.

2.2.2 Limitations of Conventional Resume

Conventional resume-building methods lack the sophistication required to navigate through this digital obstacle course. As a result, qualified individuals often face missed opportunities due to resumes that fall short of meeting ATS criteria.

2.2.3 Significance in the Digital Age

The significance of this problem is underscored by the transformative shift in how job applications are processed. Digital advancements necessitate a solution that harmonizes traditional resume elements with the nuances of digital screening, ensuring that individuals are not overlooked due to formatting or keyword-related issues.

As more companies use digital tools for hiring, it's essential for job seekers to understand how to make their resumes work in this new environment. If they don't, they might miss out on job opportunities.

2.2.4 Impact on Career Opportunities

The gravity of the problem lies in its direct impact on career opportunities. As industries embrace digitalization, which encompasses the transformation of processes and workflows alongside the adoption of digital formats, a proficient understanding of how to present oneself in the digital realm becomes a fundamental skill. The project acknowledges that the inability to navigate this digital transition hampers career growth and limits access to a broader spectrum of employment prospects.

This problem has a big impact on your career opportunities. As more industries go digital, it's crucial to know how to present yourself online. If you can't adapt to this digital shift, it can slow down your career growth and limit the kinds of jobs you can get.

2.3 Project goal and expected outcomes:

The project sets clear goals to tackle the identified problem and envision specific outcomes to make a substantial impact on the resume-building landscape.

The project sets out several key goals to guide its development:

1. ATS Optimization: Develop a web application that comprehensively understands and integrates with ATS, ensuring resumes created through the platform meet the criteria set by these systems.

2. User-Centric Design: Prioritize user experience by incorporating a seamless, intuitive interface. The application aims to be user-friendly for individuals with varying levels of technical expertise.

3. Customization and Adaptability: Provide users with the ability to customize resumes to reflect their unique experiences and skills. The platform should be adaptable to various industries and career levels.

4. Real-Time Preview: Integrate a real-time preview feature, allowing users to visualize the appearance of their resumes as they make modifications.

By focusing on these goals, the project aims to deliver a comprehensive and user-friendly web application that empowers individuals in crafting resumes optimized for success in the digital employment landscape.

The project anticipates several key outcomes upon completion:

1. Fully Functional Prototype: Develop a prototype that showcases the core features of the resume builder and parser, allowing users to interact with and understand the application's capabilities.

2. Comprehensive Documentation: Create thorough documentation detailing the functionality, development process, and user instructions for seamless navigation and utilization of the application.

3. Performance Evaluation Report: Conduct a comprehensive analysis of the application's performance. This includes user feedback, speed tests, and compatibility checks across different devices and browsers.

4. Case Studies and User Testimonials: Compile case studies demonstrating the positive impact of the application on resume-building. Include before-and-after analyses to showcase improvements in ATS compatibility.

3 TECHNOLOGY INTRODUCTION

3.1 The modern Web Development Stack

3.1.1 Next.js: Powering Server-side rendering

Next.js is a significant component of the development stack, renowned for its robust support of server-side rendering (SSR). Next.js allows you to perform SSR for specific components, allowing you to leverage its server-side rendering capabilities. Some scenarios where it can be beneficial is initial page load, dynamic content.

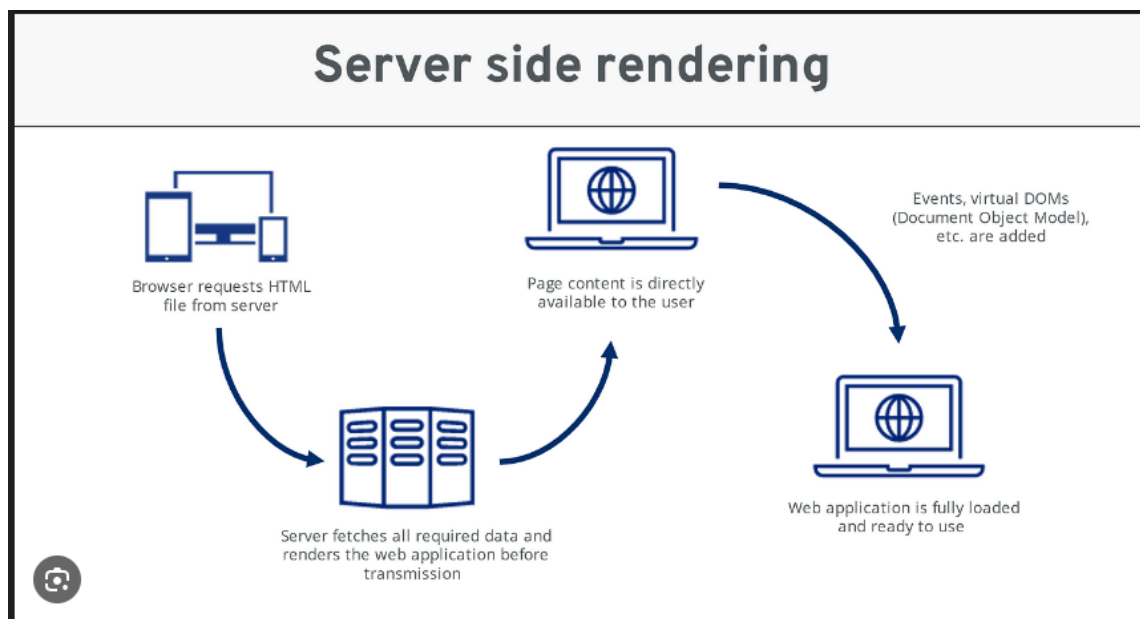


Figure 1: Next.js server rendering.

Source: Swas-tik Supakar – Jun 16, 2023

As seen from **figure 1**, SSR is crucial for optimizing performance and improving Search Engine Optimization (SEO) by delivering pre-rendered HTML pages to users and search engine crawlers. According to the [2023 State of JavaScript survey](#), Next.js is one of the most popular frameworks for SSR, with a high satisfaction rate among developers. While there are several alternatives to Next.js for SSR, its popularity and widespread adoption in the development community highlight its reliability and effectiveness in rendering dynamic content on the server side.

3.1.2 Redux Toolkit: Robust State Management

The project harnesses the power of Redux Toolkit for efficient state management. This ensures a scalable and organized approach to handling application state, crucial for the complexity of a resume-building tool.

The chart in **Figure 2** shows how Redux works in a practical application which has 4 components. ComponentA accepts user input text and passes it over to ComponentB as a prop. At the same time, ComponentA dispatches the action to save the data in the store so that ComponentC and componentD can use it (Adapted and rephrased, and referring to the original source: "*Medium – Lada496*").

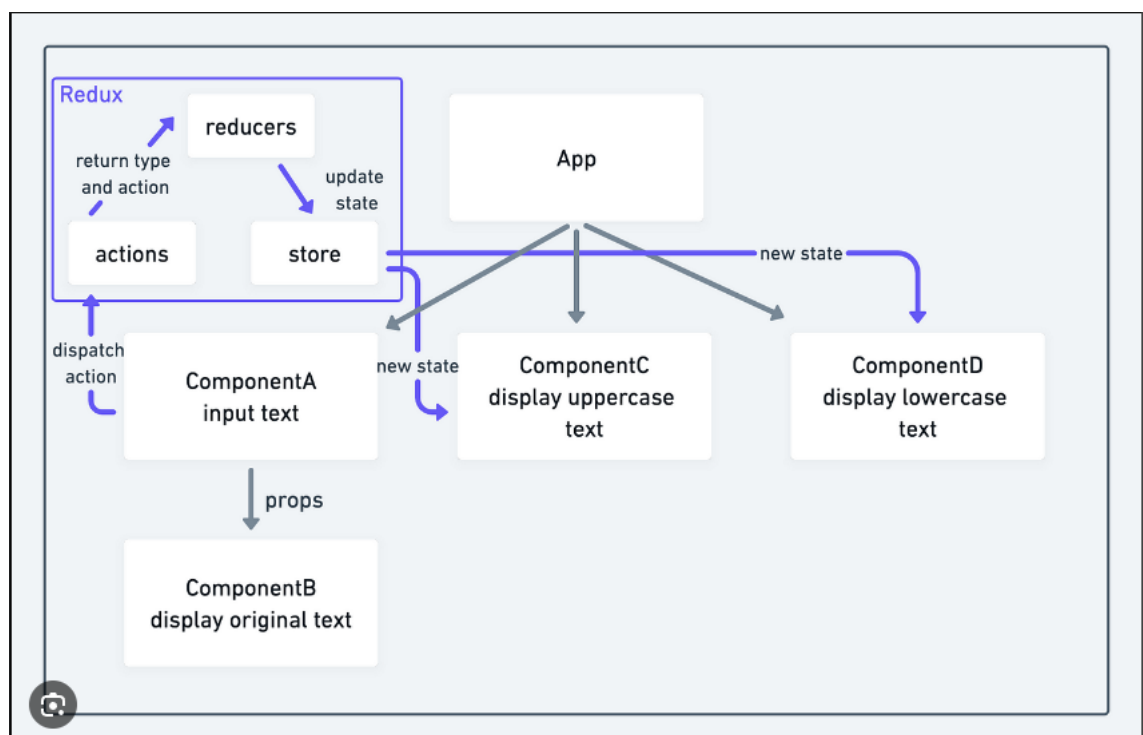


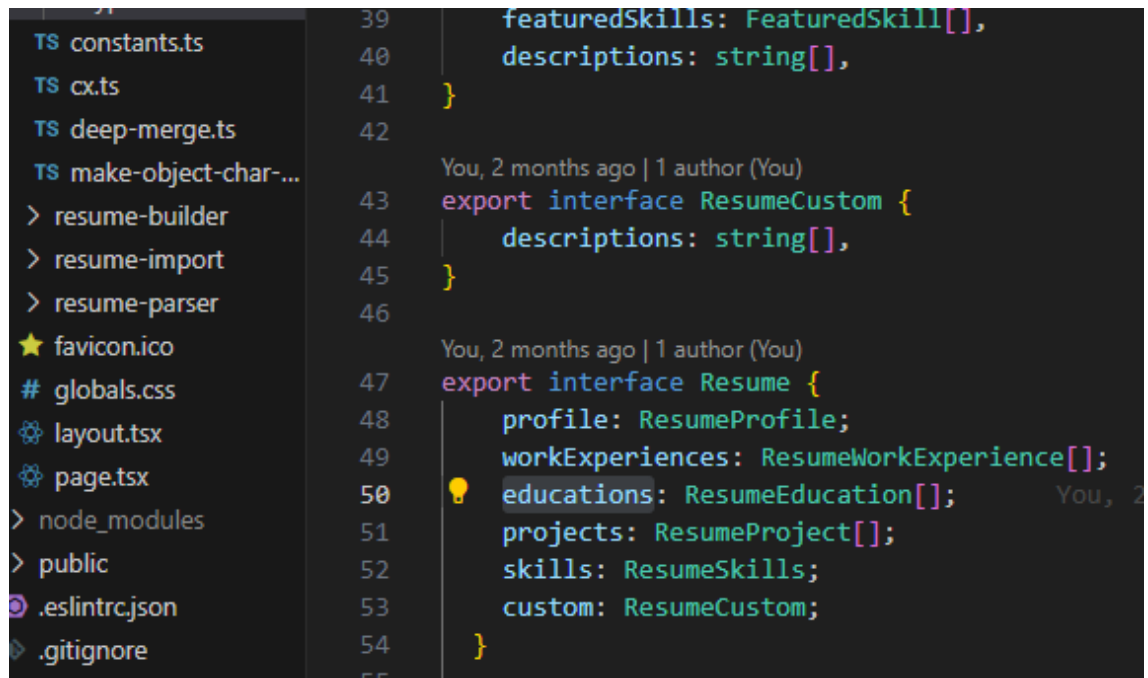
Figure 2: Redux Toolkit State Management.

Source: *Medium – Lada496*

3.1.3 TypeScript: Elevating Development Experience:

TypeScript is employed to elevate the development experience by introducing static typing. This not only catches potential errors early in the development process but also enhances code readability and maintainability.

In figure 3 below is a piece of code in the project and is an example of static typing which will automatically inform developers if the types do not match.



```

39     featuredSkills: FeaturedSkill[],
40     descriptions: string[],
41   }
42
43   You, 2 months ago | 1 author (You)
44   export interface ResumeCustom {
45     descriptions: string[],
46   }
47
48   You, 2 months ago | 1 author (You)
49   export interface Resume {
50     profile: ResumeProfile;
51     workExperiences: ResumeWorkExperience[];
52     educations: ResumeEducation[];
53     projects: ResumeProject[];
54     skills: ResumeSkills;
55     custom: ResumeCustom;
56   }

```

Figure 3. TypeScript in Actions. (Source: Projects's source code).

3.2 Role of Next.js in Modern Web Applications

Next.js, a powerful and versatile framework, plays a pivotal role in shaping the landscape of modern web applications, including the innovative resume builder and parser project.

3.2.1 Enhanced Search Engine Optimization (SEO)

Traditional Single Page Applications (SPAs) built with React or similar libraries often face SEO challenges. These challenges stem from the fact that content on these pages is dynamically loaded using JavaScript, which search engines might not always execute or index effectively. This can lead to poor SEO as the content is not immediately available to search engine crawlers.

Next.js, address these SEO challenges through several key features: Server-side rendering, Static Site Generation, Incremental Static Regeneration, etc.

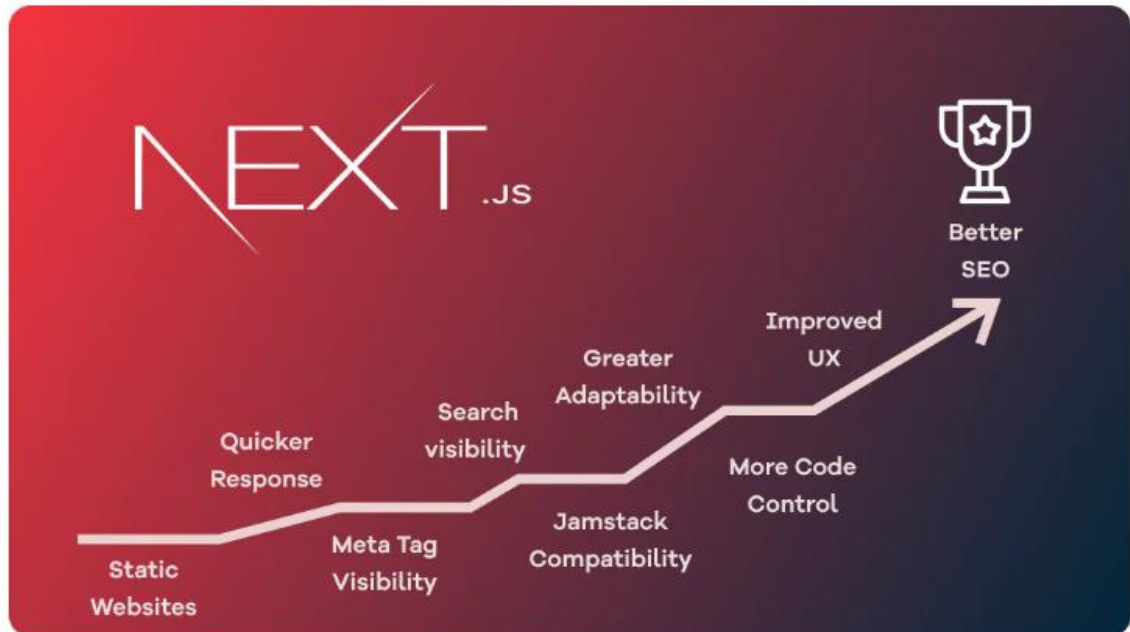


Figure 4: Key features that make Next.js is a better choice for SEO.

Source: Asad Mumtaz (Feb 26th 2024)

3.2.2 Efficient Page Routing and Navigation

Next.js have Next.js App Router, which is a routing solution for its applications, offering seamless navigation between pages while optimizing performance through server-side rendering and automatic code splitting.

The App Router has many benefits that make it a powerful and flexible tool for web development. Some of the main benefits are:

Nested route and layouts: you can create routes that have sub-routes, such as `/blog/[id]/comments`, and layouts that wrap around other components, such as a header or a sidebar. This helps you organize your application's structure and UI in a declarative and intuitive way.

Simplified data fetching: You can use suspense to fetch data from React Server Components or from APIs without writing complex logic or managing state. Suspense is a React feature that lets you show a fallback UI while waiting for data to load. You can also use custom hooks to share data between components or prefetch data for faster navigation.

Streaming and suspense: You can stream data from React Server Components to the client in chunks instead of all at once. This improves the performance and user experience of your application, especially on slow networks. You can also use suspense to show loading indicators or fallback UI while waiting for data or components to load.

Built-in SEO support: You can optimize your website for search engines like Bing by using the App Router's built-in SEO support. The App Router automatically generates meta tags, sitemaps, and robots.txt files for your application, which help search engines understand and index your content. You can also customize these files or add your own SEO logic using React Server Components.

Source: Charles Ilo

3.3 Advantages of TypeScript in Development

TypeScript, a statically typed superset of JavaScript, as can see from **Figure 5** below, emerges as a strategic choice in the development stack, contributing numerous advantages to the creation of the resume builder and parser application.

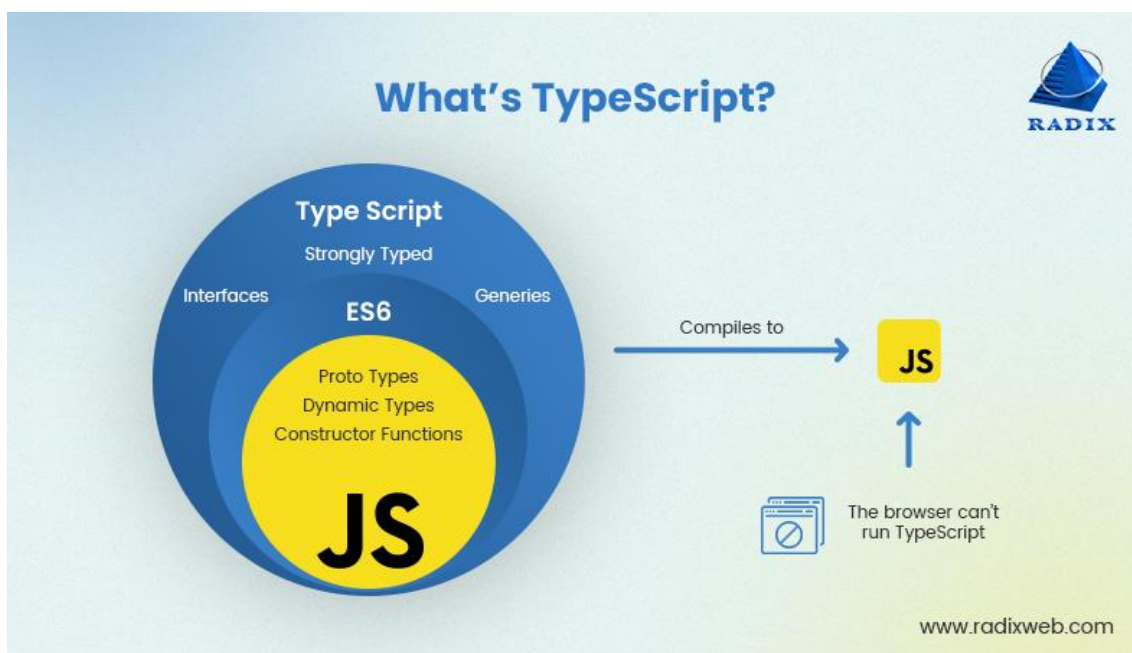


Figure 5: What is TypeScript?

Source: Nihar Raval

TypeScript introduces static typing, enabling developers to catch potential errors during the development phase. This early error detection mechanism ensures a more robust and reliable codebase.

By explicitly defining variable types, TypeScript enhances code readability. This results in code that is not only more understandable for developers but also facilitates collaborative efforts and future maintenance.

Refactoring code becomes more efficient with TypeScript, as the compiler assists in identifying and updating all instances of a particular type or variable. This support streamlines the code improvement process.

TypeScript benefits from a robust tooling ecosystem, including integrated development environments (IDEs) and build tools. This extensive support ensures a seamless integration of TypeScript into the development workflow.

3.4 The Utility-First Approach of TailwindCSS

3.4.1 Utility-First Philosophy

TailwindCSS embraces a utility-first philosophy, a design approach where styling is achieved primarily through applying pre-defined utility classes directly in the HTML markup. This methodology prioritizes the use of utility classes to style elements, offering developers a highly efficient and flexible way to design interfaces. By using utility classes, developers can quickly apply styles without writing custom CSS, resulting in unparalleled flexibility and speed in designing interfaces.

Source: According to the [TailwindCSS Documentation v3.4.3](#), the utility-first philosophy is a core principle of the framework, emphasizing the use of utility classes for styling elements.

3.4.2 Efficiency and Customization in Styling with TailwindCSS

The utility-first approach allows developers to rapidly style components by combining classes, ensuring a consistent visual language across the application. This results in efficient styling without the need for extensive custom CSS.

TailwindCSS provides extensive customization options, allowing developers to tailor the default styles to match the application's branding and design requirements. This level of configurability ensures a personalized and cohesive visual identity.

TailwindCSS simplifies responsive design by offering utility classes for different breakpoints. This enables the creation of interfaces that seamlessly adapt to various screen sizes without the need for complex media queries.

TailwindCSS integrates seamlessly into the development workflow, complementing other technologies in the stack. Its utility-first nature aligns well with the component-based approach, enhancing the overall efficiency of the styling process.

3.5 Document Rendering with ReactPDF Renderer

3.5.1 Dynamic PDF Generation

ReactPDF Renderer enables the creation of dynamic and visually appealing PDF documents directly from React components. As Next.js, a framework based on React, is utilized in the project's technology stack, leveraging React components for various functionalities is inherent. Therefore, integrating ReactPDF Renderer seamlessly aligns with the project's architecture. This feature ensures that resumes generated through the application maintain a professional and standardized format.

The integration with React components allows for a consistent and modular approach to document creation. Each section of the resume, whether it be the introduction, education, work experience, or skills, is represented as a React component, providing maintainability and flexibility.

3.5.2 Customization of Document Layout

ReactPDF Renderer equips developers with the necessary tools to customize document layouts, allowing them to meet industry standards and accommodate user preferences effectively. While ReactPDF Renderer offers robust capabilities for layout customization, it's imperative to recognize that the actual implementation within the application significantly influences the outcomes achieved. Developers can leverage the features of ReactPDF Renderer, but the thoughtful execution of design choices within the application itself is paramount. By strategically integrating ReactPDF Renderer and implementing design decisions that consider both ATS optimization and visual appeal, developers can ensure that resumes are presented in a professional and well-structured manner.

In essence, ReactPDF Renderer serves as a powerful ally in the customization of document layouts, offering a wide array of functionalities to tailor resumes according to specific requirements. However, the success of this customization relies heavily on how effectively these tools are utilized within the application context. Therefore, developers must carefully consider design choices and implementation strategies to achieve the desired results in document layout customization. By harnessing the capabilities of ReactPDF Renderer and implementing thoughtful design decisions, developers can create resumes that not only meet industry standards but also resonate with users, ensuring a seamless and engaging experience.

3.5.3 Responsive Design for Print

Responsive design tailored for print is a notable feature offered by ReactPDF Renderer. This feature enables developers to create PDFs that are optimized for printing, ensuring that the generated documents maintain their formatting and visual consistency across various printing devices and paper sizes. By incorporating responsive design principles, developers can guarantee that the PDF resumes generated by the application meet professional standards and provide a polished presentation to potential employers.

The implementation of responsive design for print involves careful consideration of factors such as layout, typography, and color usage to ensure readability and aesthetic appeal in printed form. Developers can utilize features provided by ReactPDF Renderer to dynamically adjust the layout and styling of the PDF documents based on the selected paper size and printing settings. This flexibility empowers developers to create PDFs that are not only visually appealing on screen but also translate seamlessly to physical copies.

Furthermore, responsive design for print aligns with modern printing practices, where individuals often print documents from various devices and expect consistent results. By integrating this feature into the application, developers enhance the user experience by ensuring that the generated PDF resumes meet the expectations of both digital and print environments. This attention to detail reflects positively on the professionalism and quality of the application, reinforcing its value as a comprehensive tool for resume creation and presentation.

4 TECHNOLOGY FOUNDATION AND RESUME ESSENTIALS

4.1 Crafting Intuitive User Experiences

4.1.1 Interface Development: Next.js and Tailwind CSS

Next.js serves as the foundation of our front-end architecture, providing unparalleled capabilities in server-side rendering. By rendering pages on the server and sending pre-built HTML to the client, Next.js enhances performance and search engine optimization (SEO), ensuring faster page loads and improved visibility on search engine results pages (SERPs).

Complementing Next.js is Tailwind CSS, a utility-first CSS framework that enables rapid UI development. With Tailwind CSS, we can apply pre-defined utility classes directly in the HTML markup, streamlining the styling process and ensuring consistency across the application. This utility-first approach empowers developers to create visually appealing interfaces without the need for custom CSS, resulting in cleaner and more maintainable code.

4.1.2 Code Integrity with TypeScript

To maintain code integrity and enhance development efficiency, we integrate TypeScript into our front-end stack. TypeScript introduces static typing to the JavaScript ecosystem, allowing us to catch errors during development and enforce stricter data types and interfaces. By providing a layer of type safety, TypeScript promotes cleaner, safer, and more maintainable code, ultimately reducing bugs and improving code quality.

In the project, we are utilizing Enums and Union Types can help make the code more expressive and less prone to errors. Furthermore, TypeScript offers advanced features like generics, namespaces, and decorators, which can help write more reusable and modular code. Integrating TypeScript's type checking into the build process can help catch errors before deployment.

4.1.3 Responsive Design Techniques

Responsive Web Design is a contemporary strategy in website development that underscores the significance of adjusting to the user's actions and surroundings. This includes considering factors like screen size, platform, and device orientation.

In today's era, where the majority of the global population utilizes mobile phones, a website's adaptability is not merely a choice but a requisite. To achieve this, we implement responsive design methodologies to seamlessly tailor our application to various viewing contexts, augmenting accessibility and user contentment. - (Adapted and rephrased, and referring to the original source: "Responsive Web Design Basics")

By employing fluid layouts, flexible grids, and media queries, we ensure that our application maintains functionality and aesthetic appeal across diverse devices and screen dimensions.

4.2 State Management with Redux Toolkit

4.2.1 Redux Toolkit for Predictable State Management

Redux Toolkit provides a predictable state container for TypeScript applications, facilitating the management of the application's state in a consistent and organized manner. This predictability is crucial for maintaining a clear data flow within the application.

The application utilizes a centralized store to hold the state of the application. This centralized approach ensures that all components have access to the most up-to-date and synchronized data, preventing inconsistencies in the user interface.

Redux Toolkit employs actions and reducers to manage state changes in a controlled manner. Actions define the type of change, while reducers specify how the state should be updated based on these actions. This structure ensures maintainability and clarity in state transitions.

4.2.2 DevTools Integration for Debugging

Redux DevTools is an indispensable tool for debugging and analyzing the state of your application built with Redux. It provides a real-time view of your app's state and helps you track changes as they happen.

The integration of Redux DevTools provides powerful debugging capabilities. Developers can visualize and track state changes over time, making it easier to identify and resolve issues during the development process.

In this application, we need to save state to local storage and load state from local storage so user can continue from where they left off, therefore having assistance from DevTool is much needed for Time-travel Debugging, State Inspection, import and export state and action dispatching.

4.3 Résumé Essentials: Tradition and Digital Adaptation

Both an ATS resume and a general CV serve the purpose of showcasing a person's qualifications and achievements, they differ in terms of format, length, content emphasis, and optimization, depending on their intended use and the expectations of the hiring process.

4.3.1 Traditional Formatting Conventions

In general, traditional includes extensive information about academic and professional achievements, research, publications, conferences attended, and more. Crafting a compelling résumé begins with adhering to traditional formatting conventions that emphasize clarity, professionalism, and relevance. They are used in various contexts, including academic applications, research positions, and international job markets. Below are key elements of traditional résumé. - "Abhishek Kundu - 21st Sep 2023"

Clear and Concise Structure: Résumés should be well-organized and easy to navigate, with clearly defined sections such as contact information, summary or objective, work experience, education, and skills.

Professional Design: A clean and visually appealing design enhances readability and leaves a positive impression on recruiters. Utilizing standard fonts, appropriate font sizes, and consistent formatting ensures a professional presentation.

Concise and Impactful Content: Each section of the résumé should contain concise and impactful content, focusing on relevant achievements, skills, and experiences that highlight the candidate's qualifications for the desired position.

4.3.2 Navigating ATS Requirements

An ATS-optimized resume is typically more structured and uses a straightforward format. It prioritizes readability for both humans and automated software. It often includes key sections like contact information, a summary or objective, skills, work experience, and education, with clear headings and bullet points. These resumes are tailored for specific job applications and emphasize the skills and experience relevant to the particular job. Irrelevant information is minimized. - ”
Abhishek Kundu - 21st Sep 2023”

Compare to traditional resumes, ATS Resume focus more on keywords and optimization from the job posting to increase the chances of passing through the automated screening process. They are primarily used for job applications, especially when submitting online applications through ATS systems. Here are some keys factors.

Keyword Optimization: as mention above, ATS scan résumés for specific keywords and phrases related to the job description. Tailoring résumé content to include relevant keywords increases the likelihood of passing ATS screenings and reaching human recruiters.

Document Formatting: ATS have specific formatting requirements, often favoring simple layouts and standard file formats such as .docx or .pdf. Avoiding complex layouts, graphics, and images ensures compatibility with ATS and improves parsing accuracy.

Content Accessibility: clear and structured content improves ATS parsing accuracy. Utilizing standard section headings, bullet points, and consistent formatting allows ATS to extract relevant information efficiently.

5 SOLUTION IMPLEMENTATION

Implementing the core features of our solution forms the foundation of our implementation process. This involves translating the requirements outlined in our design specifications into functional components and modules that power the user interface and backend functionality. Key aspects of core feature implementation include:

Résumé Builder and Parser: the backbone of our application, the résumé builder and parser functionality enable users to create, edit, and customize their résumés with ease. Leveraging technologies such as React components and state management libraries, we provide users with an intuitive interface for crafting professional résumés.

ATS Compatibility Integration: integrating ATS compatibility features into our solution ensures that résumés created through our platform are optimized for parsing by Applicant Tracking Systems. This involves implementing algorithms and parsing techniques that align with ATS requirements, allowing users to maximize their chances of success in job applications.

To check out the detailed specific algorithms used in this application (check out appendix 1)

5.1 Crafting Resume with ease:

The application simplifies the résumé-building process, allowing users to create polished résumés with minimal effort. With a user-friendly interface and intuitive design, users can easily navigate through the application and access essential features. Key functionalities include:

Résumé Creation: Users can start from scratch or upload an existing résumé to begin crafting their professional profiles. The application guides users through the process, prompting them to fill in essential sections such as personal information, education, work experience, and skills.

Customization Options: Our software offers extensive customization options, allowing users to personalize their résumés to suit their preferences and industry

standards. Users can choose from various templates, fonts, colors, and layouts to create a unique and visually appealing résumé.

5.2 Optimizing Résumés for ATS Compatibility

In today's digital job market, ATS compatibility is essential for ensuring résumés get noticed by employers. Our software seamlessly integrates ATS compatibility features, enabling users to optimize their résumés for parsing by Applicant Tracking Systems. Key functionalities include:

ATS Parsing Preview: users can preview how their résumés would be parsed by common ATS systems used in job applications. The application identifies potential formatting or keyword-related issues that may affect ATS parsing, allowing users to make necessary adjustments for optimal compatibility.

Keyword Optimization: our software analyzes résumés for relevant keywords and phrases commonly used by ATS systems. By highlighting keywords and suggesting additions or modifications, users can enhance their résumés' ATS compatibility and increase their chances of success in job applications.

5.3 Workflow showcase

5.3.1 Resume Parsing Workflow

This is a comprehensive overview of how users can leverage our software solution to streamline their résumé-building process and enhance their job application endeavors.

Step 1: Upload Resume PDF

Users initiate the process by uploading their existing resume in PDF format. The application then extracts relevant information from the document, kickstarting the parsing workflow.

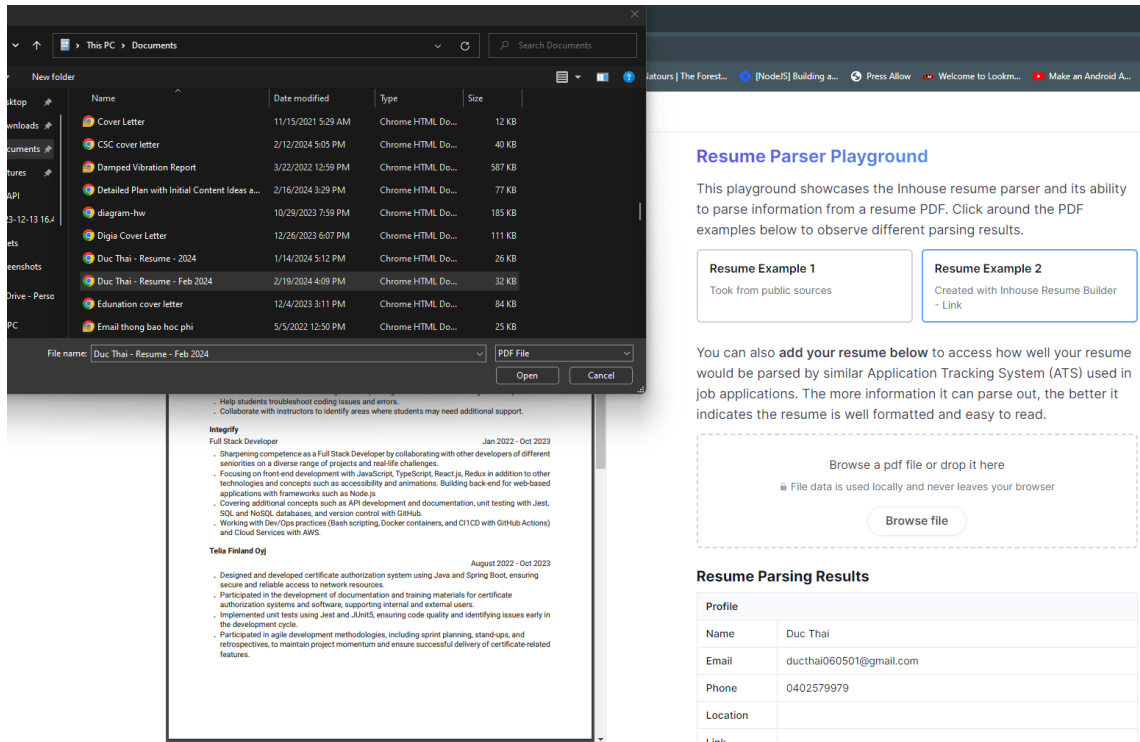


Figure 6: Resume Upload.

Source: Project Development Environment.

Step 2. ATS checking:

After successful parsing, users are introduced to the resume parser playground—a dynamic tool within our application, see [figure 7](#). Here, users can explore the capabilities of our in-house resume parser firsthand. By uploading their resume PDF, users can evaluate the accuracy and effectiveness of our parser in extracting information. This simulation allows users to gauge how well their resumes would fare when processed by various Applicant Tracking Systems (ATS) commonly utilized in job applications.

The comprehensiveness of information parsed serves as an indicator of the resume's formatting and readability. It's essential to note that while our parser provides valuable insights, its performance may differ from other ATS solutions. To enhance users' understanding of ATS requirements, we offer guidance on typical parsing patterns and preferences. Additionally, users can access resources

aimed at optimizing their resumes for ATS compatibility, ensuring their documents are well-structured and easily interpreted by a wide range of ATS platforms.

Resume Parsing Results

Profile	
Name	Duc Thai
Email	ducthai060501@gmail.com
Phone	0402579979
Location	
Link	
Summary	A dynamic and passionate Full Stack Developer with four years of experience in the tech industry, specialize in crafting scalable and responsive web applications using a blend of modern technologies like TypeScript, Next.js, Node.js, and SpringBoot. Strongly committed to continuous learning, innovation, and the application of best practices in software development.
Education	
School	Tampere University of Applied Sciences
Degree	Bachelor of Software Engineering - 4.29 / 5
GPA	4.29
Date	August 2020 - May 2024
Descriptions	
Work Experience	
Company	Tampere University of Applied Sciences
Job Title	Teaching Assistant
Date	January 2022 - April 2022
Descriptions	<ul style="list-style-type: none"> • . Mentored graduate students during their studies and supported them with their course. • . Planned and conducted live-coding workshops for graduate students learning JavaScript. • . Help students troubleshoot coding issues and errors. • . Collaborate with

Figure 7: PDF parsers playground.

Source: Application development Process

5.3.2 Creating Resume from Scratch Workflow

Users can start from scratch to begin crafting their professional profiles. The application guides users through the process, here blow are the steps.

Step 1. Initiate Creation

Users opting to create a resume from scratch begin by clicking the “Create Resume” button (**Figure 8**), triggering the resume-building process.

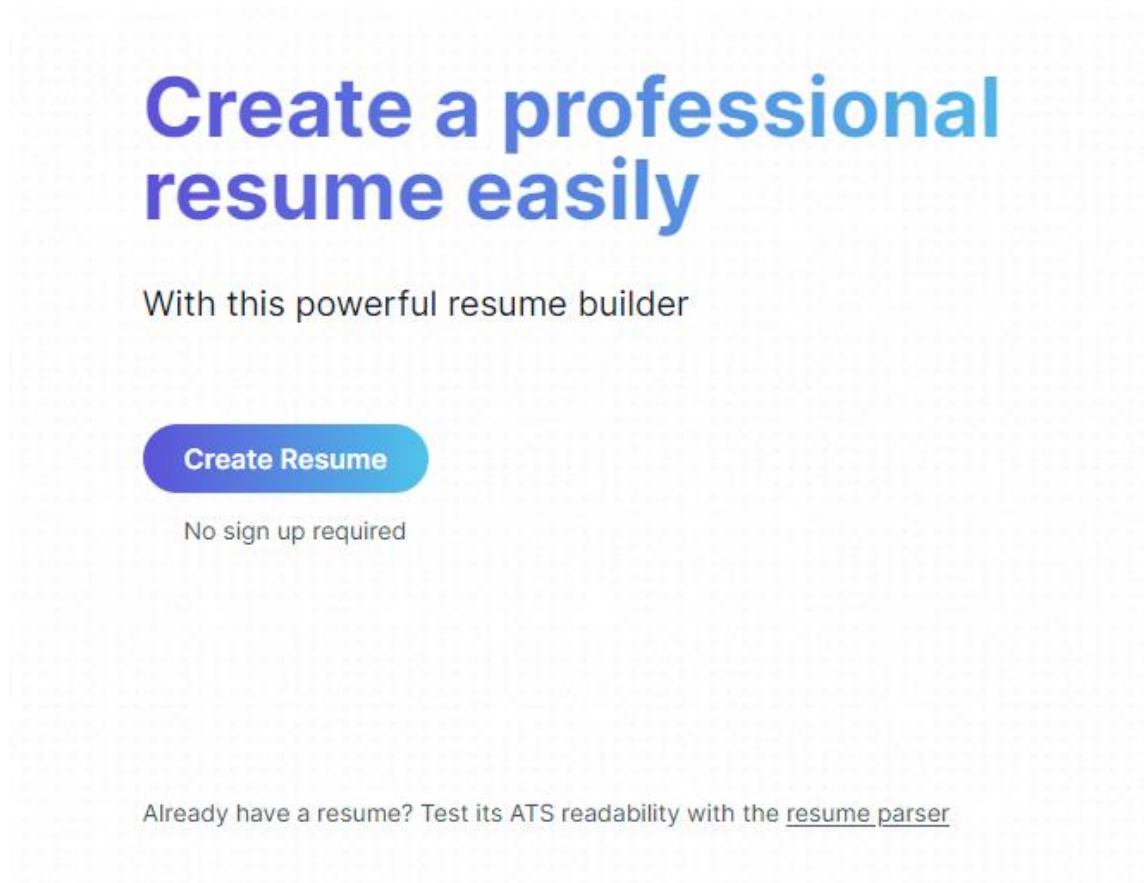


Figure 8: "Create Resume" Button Activation.

Source: Application Development Process

Step 2. Form-Based Input:

A user-friendly form is provided, prompting individuals to input details for various resume sections such as introduction, education, work experience, projects, and skills as shown in **figure 9**.

The image shows a form-based input interface for a resume. It is divided into two main sections. The top section contains personal information fields: Name (Duc Thai), Objective (A dynamic and passionate Full Stack Developer with four years of experie), Email (ducthai060501@gmail.com), Phone (0402579979), Website (willowy-kringle-c7cf3e.netlify.app/), and Location (Tampere, Finland). The bottom section is titled 'EDUCATION' and includes a dropdown arrow and an eye icon. It contains fields for School (Tampere University of Applied Sciences), Date (August 2020 - May 2), Degree & Major (Bachelor of Software Engineering), and GPA (4.29 / 5). There is also an 'Additional Information (Optional)' field with a list icon and a '+ Add Schools/Campuses' button.

Figure 9: Form-Based Input Interface.

Source: Application development process

Step 3. Customization Options:

Users can customize their resume further by adding, removing, or modifying sections based on their preferences by adding those information in the custom text-box (**figure 10**).

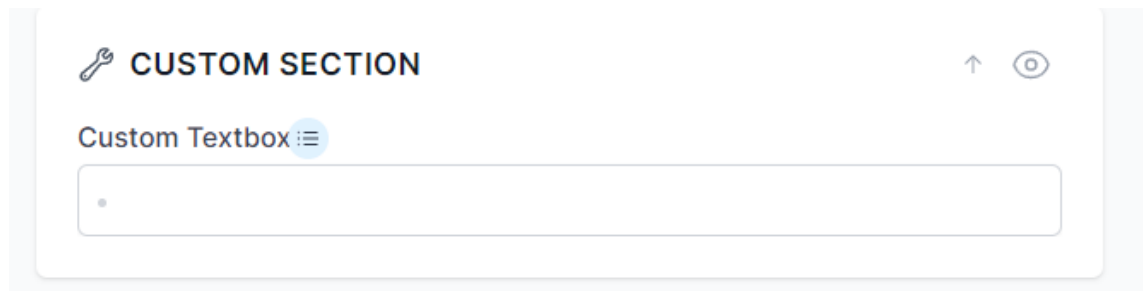


Figure 10: Customization Options Interface.

Source: Application Development process

Step 4. Real-Time Preview Update:

As users input information, the preview section is dynamically updated, allowing them to see how changes in the form directly impact the visual representation of their resume, watch **figure 11**.

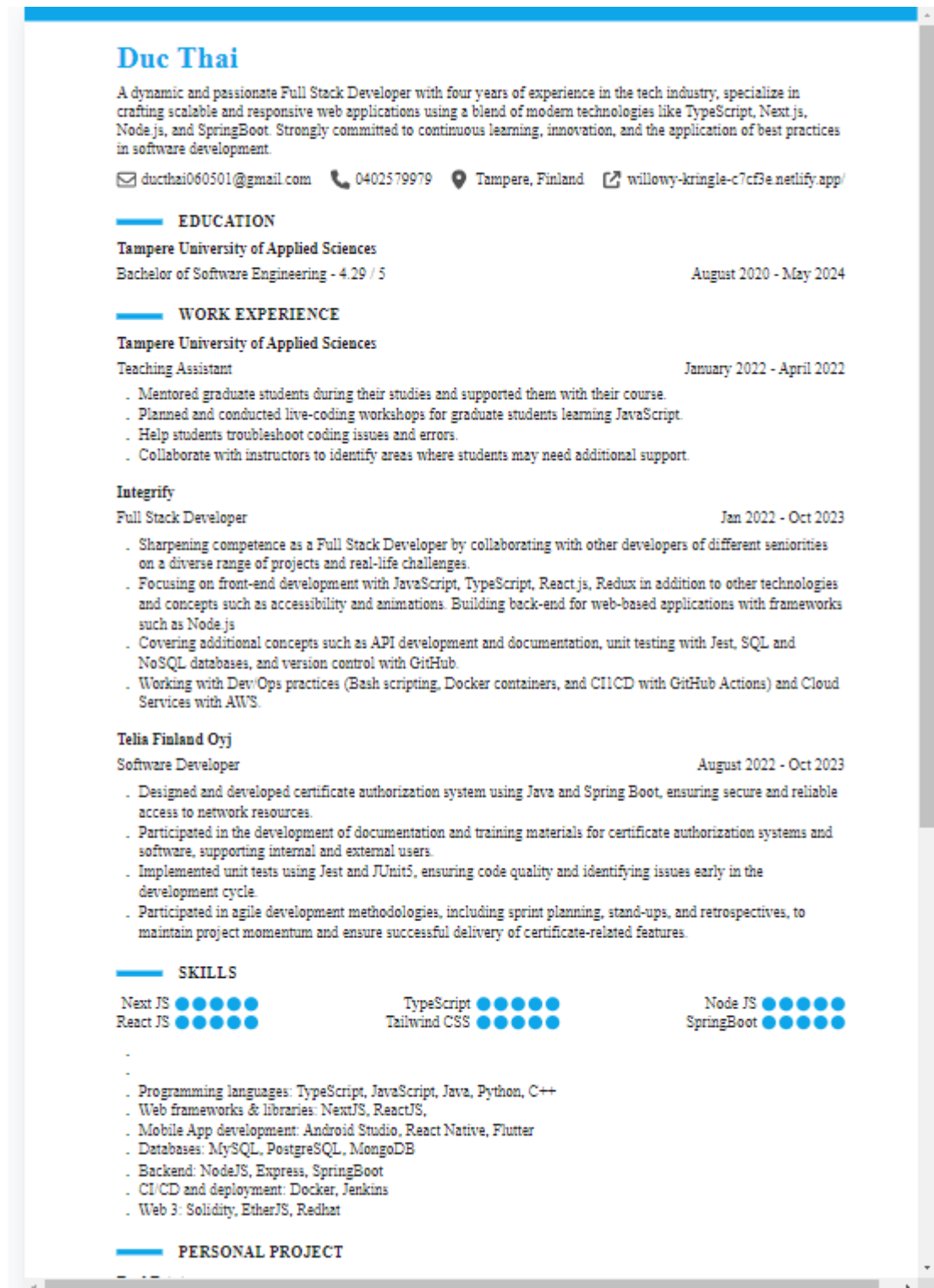


Figure 11: Real-Time Preview Update.

Source: Application development process

Step 5. Resume Settings Adjustment:

Additional options are location the resume settings (figure 12), enable users to adjust resume settings, including theme color, font, font size, and document size, ensuring the final document aligns with their personal preferences.

Resume Settings

Theme Color `#0ea5e9`

Font Family

Font Size (pt) `10`

Document Size

Figure 12: Resume Settings section.

Source: Application development process

Step 6. Download Customized Resume:

Upon completion, users can effortlessly download their customized resume, reflecting the inputted information and personalized design choices by press the “Download Resume” button **(figure 13)**.

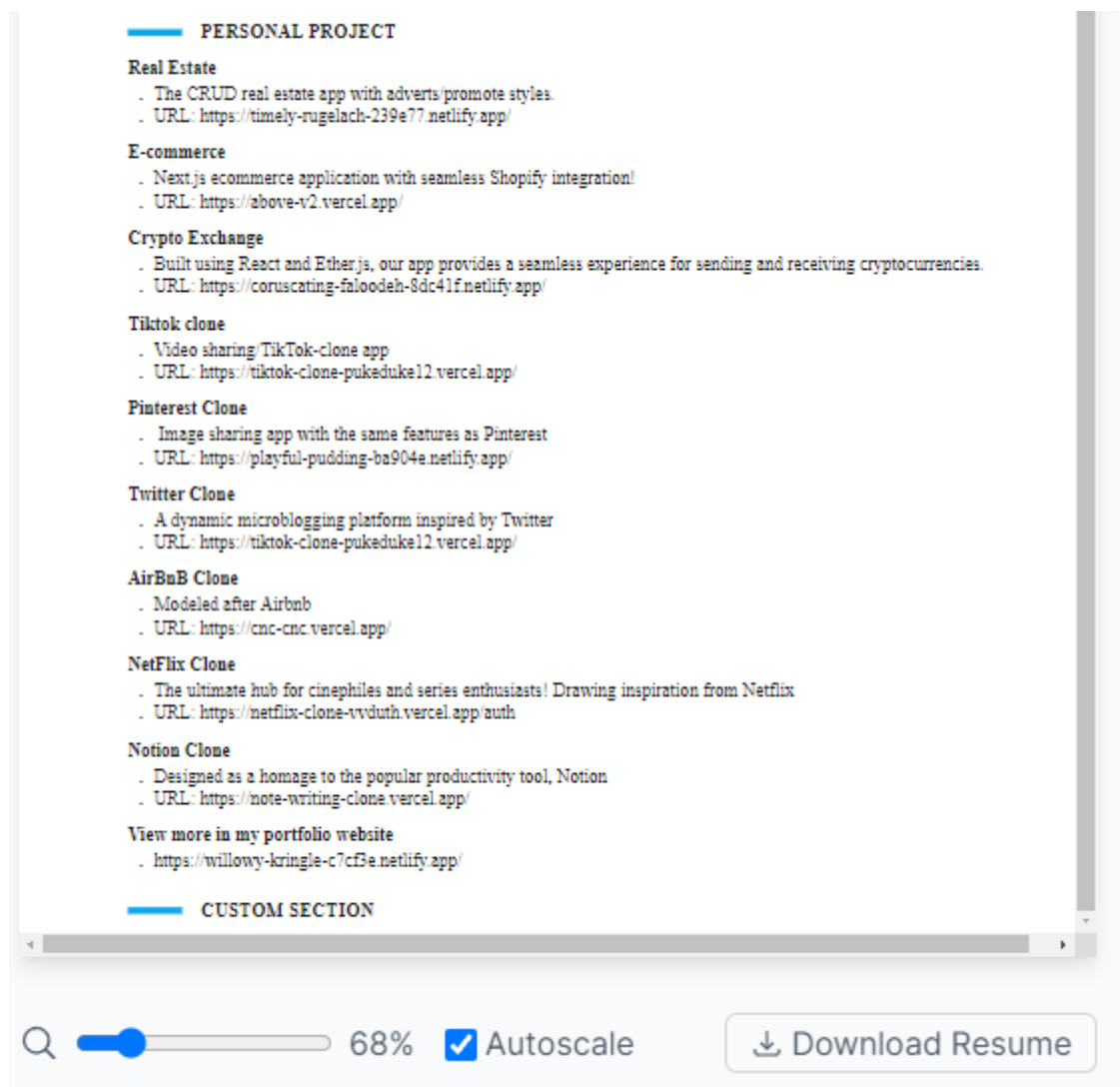


Figure 13: Download Resume Button.

Source: Application Development Process

The implementation of these core features ensures a user-friendly and comprehensive experience, empowering individuals to effortlessly create and optimize their professional resumes.

5.4 User Interactions and Experience Design

This section focuses on the critical aspects of user interaction and experience design within the resume builder application. The goal is to create an intuitive, seamless, and enjoyable user journey, ensuring that individuals can effortlessly navigate the application and derive maximum value from its features.

5.4.1 User Onboarding

1. Homepage Showcase:

The homepage serves as a visual showcase, providing users with a glimpse of the final output that the resume builder can generate. A prominent "Create Resume" button acts as the entry point, inviting users to explore the application.

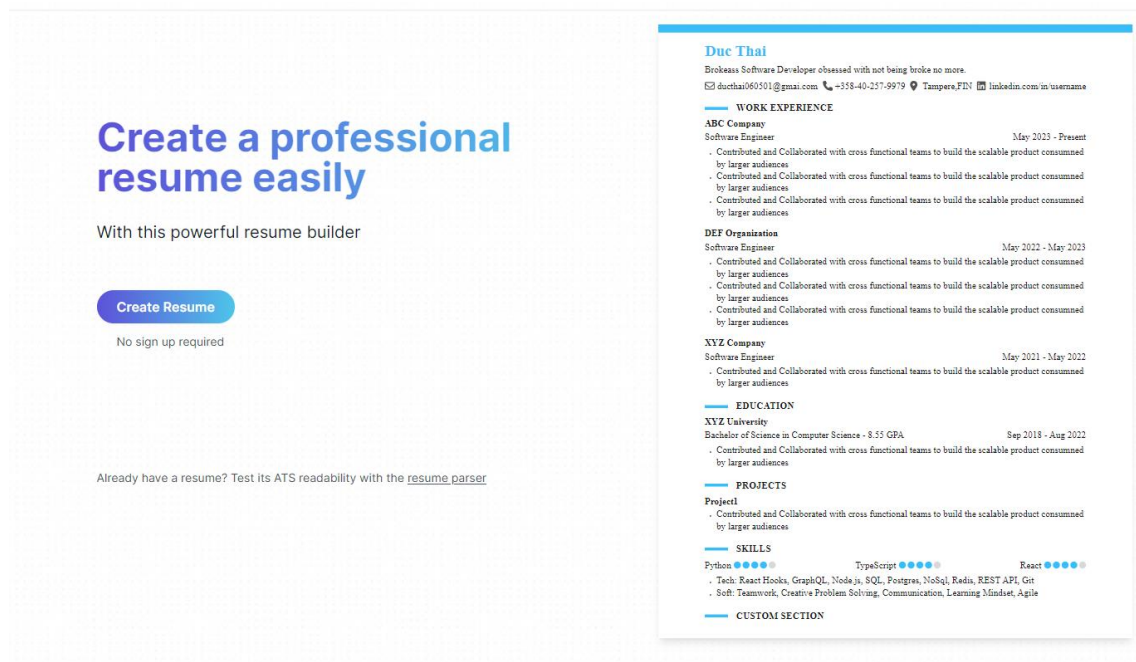


Figure 14: Homepage Showcase with "Create Resume" Button.

Source: Application Development Process

2. Guided Steps Overview:

Upon clicking "Get Started," users are introduced to a concise overview of the three simple steps required to either parse an existing resume or create one from scratch. This clear guidance enhances user confidence and understanding.

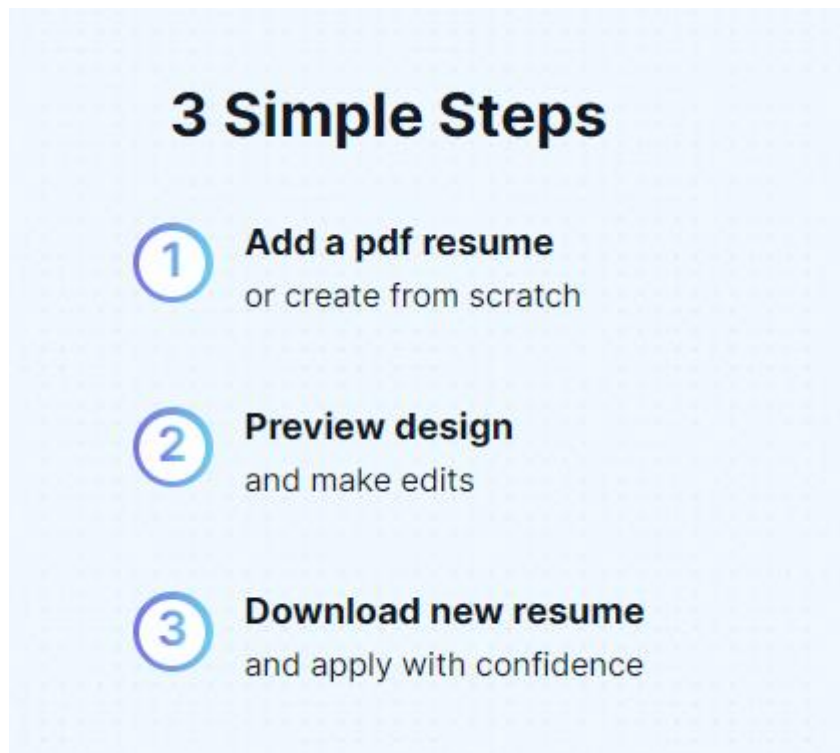


Figure 15: Guided Steps Overview Interface.

Source: Application Development Process

6 DISCUSSION AND ANALYSIS

6.1 User Experience Enhancement

User experience (UX) enhancement was a cornerstone in the development of our application. By prioritizing intuitive design and seamless navigation, we aimed to create a user-friendly interface accessible to individuals with varying levels of technical proficiency. My approach drew inspiration from Nielsen's heuristics for usability, emphasizing factors such as consistency, feedback, and error prevention.

A key focus of our UX enhancement strategy was the design of an intuitive interface that guides users through the resume-building process effortlessly. Drawing on principles of information architecture and user-centered design, we employed clear visual cues, logical flow, and concise instructions to streamline the user journey. Research by Norman suggests that intuitive design reduces cognitive load and enhances user satisfaction.

Seamless navigation was another crucial aspect of our UX enhancement efforts. Leveraging principles of Gestalt psychology and Fitts's Law, we optimized menu structures, button placement, and navigation pathways to minimize user effort and maximize efficiency. By reducing the number of clicks required to perform common tasks, we aimed to improve task completion rates and overall user satisfaction.

Effective feedback mechanisms play a vital role in enhancing user experience by providing users with timely information about their actions and system status. We incorporated visual feedback cues, such as progress indicators and success messages, to keep users informed and engaged throughout the resume-building process. Research by Nielsen suggests that immediate feedback enhances user confidence and reduces errors.

Usability testing formed an integral part of our UX enhancement strategy, allowing us to identify usability issues and gather feedback from real users. By conducting iterative design cycles based on user feedback, we iteratively refined the application interface to address pain points and improve usability. Research by Rubin suggests that iterative design leads to higher user satisfaction and product success rates.

6.2 ATS Compatibility and Parsing Accuracy

Ensuring compatibility with Applicant Tracking Systems (ATS) and achieving parsing accuracy were critical objectives in the development of our application. ATS are software tools used by employers to manage job applications and screen resumes automatically. Therefore, it was imperative to design our application to generate resumes that can be parsed accurately by ATS.

To achieve ATS compatibility, we conducted extensive research to understand the requirements and limitations of commonly used ATS platforms. This involved analyzing ATS documentation, studying industry best practices, and consulting with HR professionals. By gaining insights into ATS algorithms and parsing methods, we were able to tailor our resume parser to generate ATS-friendly resumes effectively.

The implementation of parsing algorithms was a crucial step in ensuring parsing accuracy. We developed custom parsing algorithms capable of extracting relevant information from resumes while adhering to ATS formatting guidelines. By leveraging natural language processing (NLP) techniques and machine learning algorithms, we enhanced the accuracy and robustness of our parser.

Achieving ATS compatibility and parsing accuracy is an ongoing process that requires continuous monitoring and improvement. We implemented mechanisms for collecting user feedback and monitoring parsing performance in real-time. By analyzing user feedback and parsing metrics, we identify areas for improvement and implement iterative enhancements to ensure continued compatibility with evolving ATS platforms.

Through a comprehensive analysis of our UX enhancement strategies and ATS compatibility efforts, we have gained valuable insights into the effectiveness of our application in facilitating the resume-building process and enhancing employability outcomes. By incorporating principles of intuitive design, seamless navigation, and robust parsing algorithms, we have positioned our application as a valuable tool for job seekers navigating the competitive employment landscape.

REFERENCES

1. Swas-tik Supakar – Jun 16, 2023. A Deep Dive into Server-Side and Client-Side Rendering in Next.js 13 – Swas-tik Supakar (<https://medium.com/@alexfinch607/a-deep-dive-into-server-side-and-client-side-rendering-in-next-js-13-2d0b5deecde5>)
2. Asad Mumtaz (Feb 26th 2024) - What is SEO? How can Next.JS BOOST the SEO rankings of your website. – Asad Mumtaz (<https://www.linkedin.com/in/asadmumtaz92/>)
3. Charles Ilo - The App Router: What is it? (Medium – Charles Ilo <https://medium.com/@charlesikilo>)
4. TailwindCSS Documentation v3.4.3. (<https://tailwindcss.com/docs/utility-first>)
5. 2023 State Of JavaScript Survey - <https://stateofjs.com/en-US>
6. *Medium – Lada496* - <https://medium.com/@lada496>
7. Nihar Raval - TypeScript vs Javascript: What is the difference? –(radixweb.com)
8. Abhishek Kundu - 21st Sep 2023 - <https://www.linkedin.com/pulse/ats-resume-vs-general-cv-abhishek-kundu/>
9. "Responsive Web Design Basics" - Google Developers, developers.google.com, Accessed on 8th April 2024.

APPENDICES

Appendix 1. Code Excerpts

1. Resume Parsing Algorithm

```
export const parseResumeFromPdf = async (fileUrl: string) => {
  // step 1: read a pdf resume file into text items to prepare for processing
  const textItems = await readPdf(fileUrl);

  // step 2: Group text items into line
  const lines = groupTextItemsIntoLines(textItems);

  // step 3: Group lines into sections
  const sections = groupLinesIntoSections(lines);

  // step 4: Extract resume from sections
  const resume = extractResumeFromSections(sections);

  return resume;
}

```

Figure 16: main function used to parse PDF resume.

Source: Project's source code

In **figure 16**, `parseResumeFromPdf` has 4 main functions which are put into different files, as shown in **figure 17**.

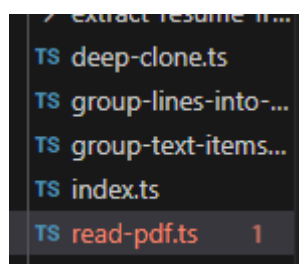


Figure 17: Structure of the files which have utility functions for `parseResumeFromPdf`.

Source: Project's source code

Code Explanation:

```
export const readPdf = async (fileUrl: string): Promise<TextItems> => {
  const pdfFile = await pdfjs.getDocument(fileUrl).promise;
  let textItems: TextItems = [];

  for (let i = 1; i <= pdfFile.numPages; i++) {
    const page = await pdfFile.getPage(i);
    const textContent = await page.getTextContent();

    await page.getOperatorList();
    const commonObjs = page.commonObjs;

    const pageTextItems = textContent.items.map((item) => {
      const {
        str: text,
        dir,
        transform,
        fontName: pdfFontName,
        ...otherProps
      } = item as PdfjsTextItem;

      const x = transform[4];
      const y = transform[5];

      const fontObj = commonObjs.get(pdfFontName);
      const fontName = fontObj.name;

      const newText = text.replace(/--/g, "-");
```

Figure 18: *readPdf* function.

Source: Project's source code

Read PDF (figure 18) and Extract Text Items:

- This function reads a PDF resume file from the provided URL and extracts text items from it.
- *readPdf* responsible for extracting content from the PDF file. It takes URL of the PDF file to be read and return a promise that resolves to an array of TextItems.
- It uses the `pdfjs.getDocument()` method to fetch the PDF document from the provide URL then iterates through each page of the PDF file using a loop. For each page it retrieves the text content using `page.getTextContent()`

- It extracts text items from the PDF content, including properties like text, font name, position (x, y), direction, transformation, etc. And then sanitizes the text by replacing any double hyphens (--) with a single hyphen. text item is transformed into a simplified object containing only necessary properties (fontName, text, x, y, and any other custom properties).

Group Text Items into Lines (figure 19):

- After extracting text items from the PDF, they are grouped into lines to prepare for further processing.

```
import { Lines, TextItem, TextItems, Line } from "./types";

export const groupTextItemsIntoLines = (textItems: TextItems): Lines => {
  const lines: Lines = [];

  let line: Line = [];

  for (let item of textItems) {
    if (item.hasEOL) {
      if (item.text.trim() !== "") {
        line.push({ ...item });
      }
      lines.push(line);
      line = [];
    } else if (item.text.trim() !== "") {
      line.push({ ...item });
    }
  }

  if (line.length > 0) {
    lines.push(line);
  }
}
```

Figure 19: *groupTextItemsIntoLines* function.

Source: Project's source code.

- This contains two main functions *groupTextItemsIntoLines* and *getTypicalCharWidth*, along with a helper function *shouldAddSpaceBetweenText*.

Function: groupTextItemsIntoLines(Figure 19):

- This function takes an array of TextItems and groups them into lines based on their vertical positions (y).
- It initializes an empty array line to store the grouped lines.
- It iterates through each TextItem in the textItems array.
- If the TextItem has an end-of-line flag (hasEOL), it pushes the accumulated line into lines and resets line to an empty array.
- If the TextItem is not an end-of-line and contains non-empty text, it adds the TextItem to the current line.
- After iterating through all TextItems, it pushes any remaining non-empty line into lines.
- It then iterates through each line to merge adjacent text items if they are close enough horizontally.

Helper Function: shouldAddSpaceBetweenText (Figure 21):

- This function determines whether a space should be added between adjacent text items.
- It checks various conditions based on the text content of the adjacent items.

Function: getTypicalCharWidth (Figure 20):

- This function calculates the typical width of characters based on the given TextItems.
- It first filters out text items with empty text.
- It calculates the most common height and font name among the text items.
- It then selects text items with the most common height and font name.
- It calculates the total width and number of characters of these selected text items.
- Finally, it computes and returns the typical character width based on the total width and number of characters.

```

const getTypicalCharWidth = (textItems: TextItems): number => {
  textItems = textItems.filter((item) => item.text.trim() !== "");

  const heightToCount: { [height: number]: number } = {};
  let commonHeight = 0;
  let heightMaxCount = 0;

  const fontNameToCount: { [fontName: string]: number } = {};
  let commonFontName = "";
  let fontNameMaxCount = 0;

  for (let item of textItems) {
    const { text, height, fontName } = item;

    if (!heightToCount[height]) {
      heightToCount[height] = 0;
    }

    heightToCount[height]++;
    if (heightToCount[height] > heightMaxCount) {
      commonHeight = height;
      heightMaxCount = heightToCount[height];
    }

    if (!fontNameToCount[fontName]) {
      fontNameToCount[fontName] = 0;
    }

    fontNameToCount[fontName] += text.length;
    if (fontNameToCount[fontName] > fontNameMaxCount) {
      commonFontName = fontName;
      fontNameMaxCount = fontNameToCount[fontName];
    }
  }
}

```

Figure 20: getTypicalCharWidth function.

Source: Project's source code

```
const shouldAddSpaceBetweenText = (leftText: string, rightText: string) => {
  const leftTextEnd = leftText[leftText.length - 1];
  const rightTextStart = rightText[0];
  const conditions = [
    [":", ",", "|", "."].includes(leftTextEnd) && rightTextStart !== " ",
    leftTextEnd !== " " && ["|"].includes(rightTextStart),
  ];

  return conditions.some((condition) => condition);
};
```

Figure 21: *shouldAddSpaceBetweenText* function.

Source: Project's source code

Group Lines into Sections (figure 22):

- Lines are grouped into sections based on the resume's structure, such as education, work experience, skills, etc.
- *GroupLineIntoSection* takes an array of lines (Lines) and groups them into sections based on certain criteria.
- It iterates through each line, checking if it is a section title using the *isSectionTitle* function.
- Finally return the grouped sections
- *isSectionTitle* checks various conditions such as whether the line is bold, contains certain keywords, and follows certain formatting rules to identify section titles.

```
export const groupLinesIntoSections = (lines: Lines) => {
  let sections: ResumeSectionToLines = {};
  let sectionName: string = PROFILE_SECTION;
  let sectionLines: any = [];

  for (let i = 0; i < lines.length; i++) {
    const line = lines[i];
    const text = line[0]?.text.trim();
    if (isSectionTitle(line, i)) {
      sections[sectionName] = [...sectionLines] as any;
      sectionName = text;
      sectionLines = [];
    } else {
      sectionLines.push(line);
    }
  }
  if (sectionLines.length > 0) {
    sections[sectionName] = [...sectionLines];
  }
  return sections;
};
```

Figure 22: *groupLinesIntoSections* function.

Source: Project's source code

```

const isSectionTitle = (line: Line, lineNumber: number) => {
  const isFirstTwoLines = lineNumber < 2;
  const hasMoreThanOneItemInLine = line.length > 1;
  const hasNoItemInLine = line.length === 0;
  if (isFirstTwoLines || hasMoreThanOneItemInLine || hasNoItemInLine) {
    return false;
  }

  const textItem = line[0];

  if (isBold(textItem) && hasLetterAndIsAllUpperCase(textItem)) {
    return true;
  }

  const text = textItem.text.trim();

  const textHasAtMost2Words =
    text.split("").filter((s) => s !== "&").length <= 2;
  const startWithCapitalLetters = /[A-z]/.test(text.slice(0, 1));

  if (
    textHasAtMost2Words &&
    hasOnlyLettersSpacesAmperands(textItem) &&
    startWithCapitalLetters &&
    SECTION_TITLE_KEYWORDS.some((keyword) =>
      text.toLowerCase().includes(keyword)
    )
  ) {
    return true;
  }
};

```

Figure 23: *isSectionTitle* function.

Source: Project's source code

Helper function: *isBold*, *isTextItemBold*, *hasLetter*, *hasLetterAndIsAllUpperCase*, *hasOnlyLettersSpacesAmperands*

```

export const isBold = (item: TextItem) => {
  return isTextItemBold(item.fontName);
};
const isTextItemBold = (fontName: string) =>
  fontName.toLowerCase().includes("bold");

export const hasLetter = (item: TextItem) => /[a-zA-Z]/.test(item.text);

export const hasLetterAndIsAllUpperCase = (item: TextItem) =>
  hasLetter(item) && item.text.toUpperCase() === item.text;

export const hasOnlyLettersSpacesAmperands = (item: TextItem) =>
  /^[A-Za-z\s&]+$/ .test(item.text);

```

Figure 24: *isSectionTitle* function

Source: Project's source code

Extract Resume Content (figure 25):

```

export const extractResumeFromSections = (section: ResumeSectionToLines) :
  const {profile} = extractProfile(section)
  const {educations} = extractEducation(section)
  const {workExperiences} = extractWorkExperience(section)
  const {projects} = extractProjects(section)
  const {skills}= extractSkills(section)

  return {
    profile,
    educations,
    workExperiences,
    projects,
    skills,
    custom: {
      descriptions: []
    }
  }
}

```

Figure 25: *extractResumeFromSections* function.

Source: Project's source code

State Management with Redux Toolkit

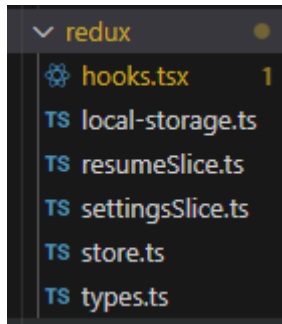


Figure 25: redux file Structure

Source: Project's source code

File structure explain (figure 25):

- store.ts: initializes the Redux store and combines reducers using Redux Toolkit's configureStore function.
- resumeSlice.ts (**figure 26**): This file contains the Redux slice for managing resume-related state, including actions, reducers, and selectors. Function createSlice is used to define a Redux slice named resumeSlice, includes reducers for changing different parts of the resume state, such as profile, work experiences, educations, projects, skills, custom sections, adding sections, moving sections, deleting sections, and setting the entire resume state.


```

export const resumeSlice = createSlice({
  name: "resume",
  initialState: initialResumeState,
  reducers: {
    changeProfile: (...
  },
    changeWorkExperience: (...
  },
    changeEducations: (...
  },
    changeProjects: (...
  },
    changeSkills: (...
  },
    changeCustom: (...
  },
    addSectionInForm: (draft, action: PayloadAction<{ form: ShowForm }>) => {
      const { form } = action.payload;
      switch (form) {...
    }
  },
    moveSectionInForm: (...
  },
    deleteSectionInFormByIdx: (...
  },
    setResume: (draft, action: PayloadAction<Resume>) => {...
  },
},
});

```

Figure 26: *resumeSlice* with its actions

Source: Project's source code

Other files: *hooks.tsx*, *local-storage.ts*, and *types.ts* contain utility functions, custom hooks, and type definitions related to Redux state management.

- *Local-storage.ts* handles the interaction with local storage to persist and retrieve application state.
- *Hooks.tsx* integrates React components with Redux store and *local-storage.useSaveStateToLocalStorageOnChange* is a custom hook that utilizes *useEffect* to subscribe to changes in the Redux store. When the store state changes, it triggers the *saveState-ToLocalStorage* function to persist the updated state to local storage.

`settingSlice.ts`: This file contains the Redux slice for managing settings-related state, including actions, reducers, and selectors. It defines the structure of the settings object, including theme colour, font settings, forms visibility, form headings, form order, and bullet points visibility.