Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

The primary purpose of this project is to streamline the process of creating college reports for students. In academic environments, students are often required to produce numerous reports, which can be time-consuming and challenging, especially for those who may not have strong writing or formatting skills. Our report maker aims to alleviate these challenges by providing a user-friendly platform that automates the report creation process.

Key objectives of the project include:

- Efficiency: To significantly reduce the time and effort required for students to produce well-structured and formatted reports.
- Accessibility: To provide an easy-to-use interface that is accessible to all students, regardless of their technical proficiency.
- Consistency: To ensure that reports adhere to standardized formats and guidelines, thereby maintaining a high level of quality and uniformity.
- Customization: To offer customizable templates and features that cater to the diverse needs of students across various disciplines.
- Integration: To seamlessly integrate with AWS services, ensuring reliable performance, scalability, and security.

1.2 Scope Of The Project

The scope of this project encompasses the design, development, deployment, and maintenance of a report maker application, hosted on AWS, that facilitates the creation of college reports for students. The following key features and functionalities are included within the scope of the project:

- User Registration and Authentication: Secure user registration and login system, integrated with AWS Cognito for user authentication and management.
- Template Library: A diverse collection of customizable report templates designed to meet various academic requirements and formatting standards.
- Report Creation and Editing: Intuitive user interface for creating and editing reports, with rich text editing capabilities including formatting, inserting images, tables, and citations. Auto-save and version control features to prevent data loss.
- Content Management: Easy import and export of content from various sources, such as text files and PDFs. Tools for organizing and managing multiple reports and sections.
- Collaboration Tools: Real-time collaboration features allowing multiple users to work on a report simultaneously. Commenting and feedback system to facilitate peer review and instructor feedback.
- Integration with AWS Services: Utilization of AWS S3 for secure storage of user data and reports. Deployment using AWS EC2 instances for scalable and reliable performance. Implementation of AWS Lambda functions for serverless computing tasks.
- User Support and Documentation: Comprehensive user guide and tutorials to help students navigate the application. In-app support and FAQ section for troubleshooting common issues.

- Security and Compliance: Data encryption at rest and in transit to ensure the privacy and security of user information. Compliance with relevant data protection regulations, such as GDPR.

Chapter 2

LITERATURE SURVEY

2.1 Research Articles In Simplified HTML: A Web-first Format For HTML-based Scholarly Articles

This paper introduces the Research Articles in Simplified HTML (or RASH), which is a Web-first format for writing HTML-based scholarly papers; it is accompanied by the RASH Framework, a set of tools for interacting with RASH-based articles. The paper also presents an evaluation that involved authors and reviewers of RASH articles submitted to the SAVE-SD 2015 and SAVE-SD 2016 workshops. RASH has been developed aiming to: be easy to learn and use; share scholarly documents (and embedded semantic annotations)through the Web; support its adoption within the existing publishing workflow. The evaluation study confirmed that RASH is ready to be adopted in workshops, conferences, and journals and can be quickly learnt by researchers who are familiar with HTML.

RASH (and its Framework)is another step towards enabling the definition of formal representations of the meaning of the content of an article, facilitating its automatic discovery, enabling its linking to semantically related articles, providing access to data within the article in actionable form, and allowing integration of data between papers.

Chapter 3

PROBLEM IDENTIFICATION

3.1 PROBLEM IDENTIFICATION

Identifying and addressing the challenges present in current tools for report generation underscores a critical need for innovation within the educational technology landscape. The prevailing methods of manual report creation often fall short of harnessing the full potential of automation and cloud computing to enhance efficiency and accuracy. Both students and professionals encounter significant obstacles when attempting to compile and format reports in a timely and error-free manner. The deficiency in streamlined and automated report generation processes contributes to inefficiencies and errors, leading to a notable gap between the time spent on manual report creation and more productive activities.

One of the primary issues is the reliance on traditional methods of report generation that are prone to human error and time-consuming processes. Many individuals and organizations still use manual methods, such as word processors or spreadsheets, which require extensive manual input and formatting adjustments. As a consequence, users may experience delays in report completion and increased frustration due to repetitive tasks that could be automated.

Furthermore, the underutilization of cloud computing capabilities within current report generation tools represents a missed opportunity for scalable and collaborative report creation. Cloud computing offers advantages such as real-time collaboration, version control, and access from anywhere, yet many report generation tools fail to fully leverage these capabilities. This underutilization restricts the ability of users to streamline workflows, collaborate effectively across teams, and ensure data security and accessibility.

The current landscape of report generation tools lacks efficient automation and scalable cloud solutions, leading to inefficiencies, errors, and missed opportunities for collaborative and streamlined workflows. Addressing these challenges through innovative use of cloud computing and automation technologies is essential to enhancing productivity, accuracy, and user satisfaction in report generation tasks.

3.2 PROBLEM STATEMENT

The identified challenges within traditional methods of report generation underscore a compelling need for transformative innovation in educational technology. The current landscape of manual report creation reveals significant shortcomings in leveraging automation and cloud computing to streamline and enhance efficiency in generating reports. Both students and professionals encounter substantial hurdles in compiling, formatting, and finalizing reports due to the reliance on outdated, error-prone methods.

The discrepancy between manual report creation and automated solutions represents a critical gap that impedes users from achieving optimal efficiency and accuracy in report generation tasks. Many existing methods rely on word processors or spreadsheets, which require extensive manual input and formatting adjustments, leading to delays and potential errors in report completion. Consequently, individuals face challenges in meeting deadlines and allocating time to more strategic tasks that could benefit from automated report generation.

Moreover, while cloud computing offers scalability, real-time collaboration, and enhanced data security, its integration within current report generation tools remains underutilized. Cloud-based solutions could optimize workflows, facilitate seamless collaboration across teams, and ensure accessibility from anywhere, yet many report generation tools lack robust integration with cloud computing services.

In response to these pressing challenges, our project aims to pioneer a modern report generation tool that harnesses the full potential of automation and cloud computing. By integrating advanced technologies, our platform will offer users a seamless and efficient solution for creating, formatting, and collaborating on reports. Through targeted innovation and strategic implementation of automated tools, we seek to bridge the gap between manual report creation and scalable, cloud-based solutions, empowering users to enhance productivity, accuracy, and collaboration in report generation tasks.

Chapter 4

GOALS AND OBJECTIVES

4.1 GOALS

- To develop an AI-powered report generation tool that automates and enhances efficiency in creating, formatting, and finalizing reports.
- To integrate cloud computing capabilities for scalable, real-time collaboration and enhanced data security in report generation processes.
- To streamline workflows and reduce errors in report compilation through advanced automation technologies.
- To empower users with intuitive features for seamless collaboration and accessibility in generating and sharing reports.

4.2 Objectives

- Enable users to automate report creation processes, including data input, formatting, and distribution.
- Implement cloud-based solutions for real-time collaboration, version control, and secure data storage.
- Enhance user experience with intuitive interfaces for efficient report generation and collaboration.
- Optimize productivity by minimizing manual tasks and maximizing automation capabilities in report generation workflows.

Chapter 5

SYSTEM REQUIREMENT SPECIFICATION

5.1 SOFTWARE REQUIREMENTS

- Operating System: Cross-platform support
- Programming Languages: JavaScript, Node.js
- Frameworks and Libraries: React.js, Express.js
- Database: Databases provided by AWS, Google Cloud Platform, Microsoft Azure, or any other reliable cloud service provider offering a comprehensive suite of cloud services.
- Development Tools: Visual Studio Code, Git

5.2 HARDWARE REQUIREMENTS

- Processor: Intel Core i5
- RAM: 8GB Minimum
- Storage: 100GB Minimum
- Graphics Card: Integrated Graphics
- Network Connectivity: High-Speed Internet
- Peripheral Devices: Standard Peripherals

Chapter 6

PARSER OVERVIEW

6.1 Parser

The parser within our report generation tool is designed to process input data and convert it into structured reports efficiently. This chapter provides an in-depth exploration of the syntax and semantics employed by our parser.

6.2 Syntax

The syntax of our parser defines the formal rules and structure that govern how input data is formatted and interpreted. It includes defining the grammar, symbols, and rules that dictate valid combinations and sequences of input elements. The syntax ensures consistency in data input and facilitates accurate parsing into meaningful report components. Key aspects of our parser's syntax include:

- Grammar Rules: Defining the language constructs and allowable sequences of tokens or symbols.
- Data Types: Specifying the types of data that can be processed, such as text, numerical values, and structured data formats.
- Syntax Validation: Verifying input conformity to predefined rules to prevent parsing errors and ensure data integrity.

6.3 Semantics

Beyond syntax, the semantics of our parser determine the meaning and interpretation of parsed input data. It involves understanding the relationships between parsed elements, applying contextual meaning, and executing actions based on parsed instructions. Key aspects of our parser's semantics include:

- Contextual Interpretation: Assigning meaning to parsed data elements based on their context within the input data.
- Execution of Actions: Processing parsed data to generate output reports, including formatting, aggregating information, and generating summaries.
- Data Transformation: Converting raw input data into structured formats suitable for report generation, ensuring accuracy and relevance.

Understanding the syntax and semantics of our parser is crucial for users to effectively utilize our report generation tool. It ensures that input data is processed correctly and that generated reports accurately reflect the intended information derived from the input. In the following sections, we will delve deeper into the implementation details and operational principles of our parser, highlighting its capabilities and optimizations that contribute to efficient and reliable report generation.

Chapter 7

IMPLEMENTATION

7.1 Tech Stack

Frontend:The frontend of our application is developed using JavaScript and utilizes React.js as the primary framework. React.js allows for efficient and dynamic user interfaces, enhancing the user experience with its component-based architecture and virtual DOM rendering.

Backend:Our backend infrastructure is built on Node.js, leveraging its asynchronous and event-driven architecture to handle concurrent requests efficiently. Express.js is used as the web application framework, providing robust routing and middleware capabilities to streamline API development and server-side logic.

Authentication:For authentication purposes, we have implemented OAuth for secure, delegated access to user resources without exposing credentials. Additionally, Passport.js is employed for authentication strategies, simplifying the integration of various authentication providers and ensuring seamless user authentication and authorization flows.

Caching and Performance:To optimize performance and enhance scalability, our application utilizes caching mechanisms. We leverage caching strategies to store frequently accessed data in-memory or through reliable cloud-based caching services. This approach minimizes database load and reduces latency, thereby improving overall application responsiveness and user experience.

7.2 User Authentication

Our application employs robust authentication mechanisms to ensure secure access for users. We offer two primary methods for authentication: OAuth and Passport.js.

OAuth enables seamless and secure authentication through Google Sign-In, allowing users to authenticate using their Google accounts. This approach enhances user convenience by eliminating the need for separate account credentials while maintaining stringent security standards.

In addition to OAuth, we utilize Passport.js to manage authentication strategies within our application. Passport.js streamlines the integration of authentication providers, offering flexibility and reliability in handling user login sessions. This framework supports various authentication mechanisms, ensuring a seamless and consistent authentication experience across different platforms and devices.

Together, OAuth and Passport.js empower our application with secure, efficient, and user-friendly authentication solutions, ensuring that user data remains protected while enhancing usability and accessibility.

7.3 Report Generation

- Report Parser: Utilizes C programming with essential libraries to parse input and generate reports efficiently.
- Frontend Interface: Users can input report content via a user-friendly interface and initiate report generation with a click.
- Document Repository: A centralized repository allows users to view and manage previously generated documents.

7.4 Backend Infrastructure

- Backend Logic: Hosted on AWS EC2, manages report parsing, storage, and user interactions.
- GraphQL API: Implements a GraphQL layer for efficient data fetching and manipulation, enhancing application performance and flexibility.

- No-IP Integration: Manages the elastic IP address for consistent access to the EC2 instance hosting the application.

7.5 Next.js And GraphQL Integration

- Next.js Framework: Enhances frontend performance with server-side rendering and optimized client-side navigation.
- GraphQL Integration: Facilitates efficient data querying and manipulation, improving application responsiveness and scalability.

7.6 AWS Services Utilization

- AWS EC2: Hosts backend logic and interfaces with frontend for report generation and user management.
- AWS S3: Stores and retrieves generated reports securely, ensuring data durability and accessibility.
- No-IP Integration: Manages the elastic IP address for reliable access to the application hosted on EC2.

Chapter 8

CODE IMPLEMENTATION

8.1 PARSER CODE

```
#include "api/libs.h"

int main(int argc, char* argv[]){
if(argc < 2){
usage_error("Please provide the filename");
}

load_sections(argv[1]);

time_t ts_epoch;
//printf("%ld", (long)ts_epoch);
struct tm *broken_time = (struct tm*)malloc(sizeof(struct tm));
broken_time = localtime(&ts_epoch);

create_pages();
get_elements();
// init_output();
// write_markup_to_file(broken_time);

return EXIT_SUCCESS;
}
```

The provided code is a C program that processes an input file and performs various operations such as loading sections, creating pages, and extracting elements. The program begins by including the necessary header file "api/libs.h", which likely contains declarations for functions and definitions required for the operations in the program. In the main function, the program first checks if the user has provided the required command-line argument (the filename). If the argument is missing, it calls the usage_error function to display an error message, prompting the user to provide the filename.

Once the filename is provided, the program calls the load_sections function, passing the filename as an argument. This function likely reads the file and loads its contents into appropriate sections for further processing. The program then prepares to work with timestamps by declaring a time_t variable named ts_epoch, which is intended to store an epoch timestamp. The localtime function is used to convert the epoch timestamp into a struct tm structure, which holds broken-down time values such as year, month, day, hour, minute, and second. Memory is allocated for this structure using malloc.

Following the time handling setup, the program proceeds to call several functions that carry out the main tasks. The create_pages function is likely responsible for creating or setting up pages based on the loaded sections. The get_elements function extracts specific elements from the loaded sections for further processing or manipulation. The commented-out lines suggest that there are additional functions, init_output and write_markup_to_file, which are intended to initialize the output and write the processed data, possibly along with the broken-down time information, to a file. Finally, the program returns EXIT_SUCCESS, indicating successful execution.

8.2 The Input File

The code defines the structure for generating a report using a custom DSL (Domain-Specific Language). It organizes the content into pages and sections with specific headings, titles, paragraphs, and items.

Page Definitions: Each page tag represents a section of the report. Pages are numbered sequentially (e.g., page0, page1, page2, etc.)and are assigned specific content related to different aspects of the report.

Heading and Title: Each page contains a heading and one or more title attributes. The heading defines the main heading of the page, while the title represents the subheadings under which paragraphs and items are organized.

Paragraphs: The paragraphs attribute holds a list of text blocks. These paragraphs contain detailed information or descriptions related to the title. Each paragraph is a string and multiple paragraphs can be added to provide comprehensive information.

Items: The items attribute is used for listing bullet points or key points. Similar to paragraphs, items are strings organized in a list and can include essential points, objectives, features, requirements, or any structured list of information.

Page Tag: Each page tag encapsulates all related content for a section of the report. This helps in organizing the content methodically.

Heading and Title: These tags help to structure the content hierarchically, ensuring that the report is easy to navigate.

Paragraphs and Items: These tags are used to add detailed text and bullet points respectively, making the report both informative and easy to read.

8.3 Sample Input Type To The Parser

```
{
"pages": {
"page0": {
"heading": "introduction",
"title": "problem statement",
"paragraphs": [
"the primary purpose of this project is to streamline the process of creating
college reports for students. in academic environments, students are often
required to produce numerous reports, which can be time-consuming and challenging,
especially for those who may not have strong writing or formatting skills.
our report maker aims to alleviate these challenges by providing a user-friendly
platform that automates the report creation process.",
"key objectives of the project include:"
],
"items": [
"efficiency: to significantly reduce the time and effort required for students
```

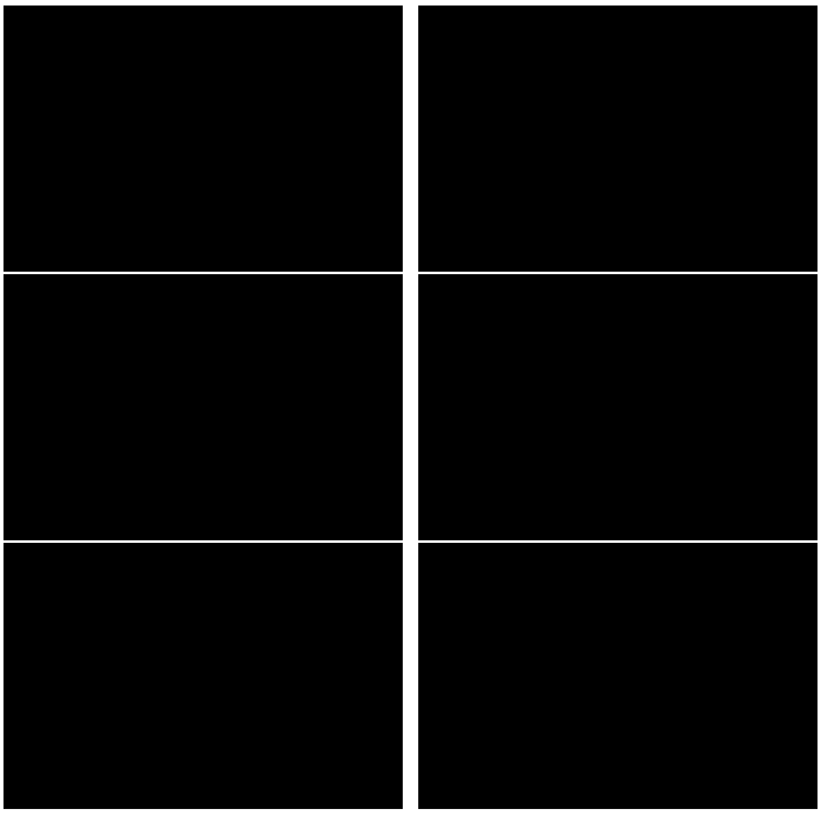
Figure 8.6: Figure 6: Frontend Implementation

```
to produce well-structured and formatted reports.",
"accessibility:  to provide an easy-to-use interface that is accessible
to all students, regardless of their technical proficiency.",
"consistency:  to ensure that reports adhere to standardized formats and
guidelines, thereby maintaining a high level of quality and uniformity.",
"customization:  to offer customizable templates and features that cater
to the diverse needs of students across various disciplines.",
"integration:  to seamlessly integrate with aws services, ensuring reliable
performance, scalability, and security."
]
}
}
```

Chapter 9

FUTURE ENHANCEMENTS

Moving forward, our focus will be on enhancing the functionality and usability of our report generation tool. One key area of development will be implementing mechanisms to avoid duplicates of the same file during updates. This enhancement will streamline document management by automatically identifying and managing duplicate reports, ensuring data integrity and reducing redundancy in storage.

Additionally, we plan to introduce support for different template formats in our report generator. This expansion will cater to diverse user preferences and document requirements, providing flexibility in document styling and layout. Users will have the option to choose from a variety of templates, optimizing the presentation of their reports according to specific needs and industry standards.

Furthermore, the integration of spell check functionality is on our roadmap. This feature will enhance the quality and professionalism of generated reports by automatically identifying and correcting spelling errors. By implementing robust spell check capabilities, we aim to improve document accuracy and readability, meeting high standards of professional communication.

Another significant future enhancement includes integrating plagiarism check capabilities into our report generation tool. This feature will empower users to ensure the originality and integrity of their content by detecting and highlighting potential instances of plagiarism. By incorporating plagiarism check tools, we uphold academic and professional integrity standards, providing users with confidence in the authenticity of their reports.

Lastly, we are committed to continuously evolving the parser in our application, adding more advanced features to enhance parsing accuracy and efficiency. Future updates will focus on refining parsing algorithms, supporting additional data formats, and improving error handling to further optimize the report generation process.

These future implementations underscore our dedication to innovation and meeting the evolving needs of our users. By prioritizing these enhancements, we aim to solidify our position as a leading solution for efficient and reliable document creation in academic and professional settings.

Chapter 10

TESTING AND RESULTS

Figure 10.2: Figure 2: Running The Parser

Chapter 11

CONCLUSION

the development of our report generation tool represents a significant advancement in leveraging modern technologies to enhance productivity and user experience in document creation. By integrating C programming with AWS services such as EC2 and S3, we have created a robust backend infrastructure capable of efficiently parsing inputs and generating reports. This infrastructure is complemented by a responsive frontend interface built on Next.js, providing users with intuitive controls for inputting content and initiating report generation seamlessly.

The adoption of GraphQL has further optimized data fetching and manipulation, enabling faster response times and scalable data handling capabilities. OAuth authentication, coupled with Google Sign-In, ensures secure access for users while simplifying the login process. Additionally, the integration of No-IP for managing elastic IP addresses has provided reliable and consistent access to our application hosted on AWS EC2.

Our project not only addresses the practical challenges of report generation but also emphasizes usability and scalability. The centralized document repository in AWS S3 ensures secure storage and easy retrieval of generated reports, enhancing workflow efficiency. Moreover, the inclusion of features like user dashboards for managing documents and tracking report history enhances user convenience and productivity.

Looking forward, our commitment to continuous improvement and adaptation to emerging technologies will drive further enhancements in functionality and user engagement. By staying at the forefront of technological innovation, we aim to redefine document creation processes, empowering users with tools that streamline tasks and elevate their overall experience.

Our report generation tool stands as a testament to effective integration of backend technologies, frontend usability, and cloud infrastructure, setting a benchmark for efficiency and reliability in document management and creation.