Vy Nguyen Thanh

Building a Serverless Full-Stack Blog Application Using AWS Amplify

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

10 October 2022

Abstract

Author:	Vy Nguyen
Title:	Building a Serverless Full-Stack Blog Application Using AWS Amplify
Number of Pages:	32 pages + 2 appendices
Date:	10 October 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Mobile Solutions
Supervisors:	Erik Pätynen, Senior Lecturer

Building an application generally necessitates the managing of the underlying infrastructure. This often refers to handling either a virtual or physical server, along with operations necessary for the application to function. Serverless architecture appears to obviate those requirements and helps developers focus on developing the application. Cloud service providers control the allocation and provisioning of servers in a serverless computing execution model.

Amazon Web Services was utilized for the project over all other cloud computing service providers because it is the most popular and well-known cloud platform in the world and offers several benefits including dependable, scalable, and economical cloud computing services. Building a serverless full-stack application was the goal of the final year project in order to show how simple it is to develop applications using AWS Amplify, a set of tools and features that developers can use to quickly and effectively build full-stack applications on AWS, as well as the advantages of using serverless architecture. Additionally, this project attempts to illustrate the capability of building a scalable system on top of JavaScript using TypeScript.

As a result, a functioning blog application was developed by the author with distinctive features such as authentication, blog creation using a custom-rich text editor, data persistence, and data filtering, just to name a few. The application was deployed to the AWS cloud and is ready to serve potential users.

Keywords: Amazon Web Service, AWS Amplify, Serverless, React

Contents

Appendices

Appendix 1: Program code - File CustomEditor.tsx

Appendix 2: Program code - File schema.graphql

List of Abbreviations

AWS:	Amazon Web Services
JSON:	JavaScript Object Notation
API:	Application Programming Interface
HTTP:	Hypertext Transfer Protocol
NPM:	Node Package Manager
IT:	Information Technology
ML:	Machine Learning
IoT:	Internet of Things
AI:	Artificial Intelligence
CLI:	Command Line Interface
HTML:	Hyper Text Markup Language
CSS:	Cascading Style Sheets
IDE:	Integrated Development Environment
SPA:	Single-page Application
UI:	User Interface

List of Figures

1 Introduction

Conventional data centers are made up of multiple hardware components, networking equipment, and servers. They often contain the applications and the data of a firm and are placed on-premises for exclusive usage. If a company uses this IT strategy, it must devote more money to maintaining and updating its infrastructure in order to attract more consumers. Also, compulsory software upgrades are entailed to guarantee the reliability of systems in the event of a hardware breakdown. In some companies, there would be a team of IT professionals to manage server infrastructure [1.]

Onsite data storage facilities will struggle to survive as information and data grow in volume due to rising storage needs and improvements in information systems [2]. As a means of reducing the footprint of data centres, promoting resource efficiency, and improving agility, cloud computing is being used in a greater number of institutions [3]. Cloud computing is defined as a method of operating computer services via the internet in order to assist the rapid deployment and scaling of application systems [4]. Because customers do not have to worry about providing the precise quantity of data they need, it also minimizes operating expenses, enhances system administration, and makes business scheduling simpler [4]. Amazon Web Services is the most comprehensive and widely used cloud provider in the world, with over 200 advanced featured services available from digital centres worldwide [5]. Millions of clients utilize it, including the world's largest corporations, most successful start-ups, and top government agencies [5]. It was thus chosen as the sole cloud provider for the final project.

The primary goal of the thesis was to create a serverless web application that would highlight the advantages of serverless architecture and show how simple it is to build apps using AWS Amplify. The major objectives of the thesis are to explain cloud services such as AWS Amplify, demonstrate Typescript and React software development, and build a sophisticated text editor with Draft.js. The thesis is organized into eight different sections: introduction, theoretical

framework, programming languages being used in the project, frontend development, project management, development project, result and conclusion.

2 Theoretical Framework

Managing architecture becomes increasingly difficult for enterprises as their technological ecosystems develop. Software developers would prefer to invest their time and resources in applications and development rather than becoming experts in system management. The emergence of serverless computing is motivated by the need to resolve this problem. [6.]

2.1 Serverless Architecture

Without needing to handle any underlying infrastructure, serverless computing offers a platform for effectively developing and releasing applications to the market. The major cloud providers have suggested a variety of serverless computing systems, including AWS Fargate, Azure Kubernetes Services, and Google Cloud Run. Because the software is designed to function on remote servers with the support of cloud service providers, such platforms allow development teams to focus on strategic approach rather than extending and providing infrastructure. Function-as-a-Service (FaaS), commonly known as “serverless function”, is the most well-known example of serverless computing. When utilizing FaaS, programmers just need to set triggers for their execution and deploy the source code for those functions. The FaaS provider then scales the execution of the functions and runs and bills them as separate instances as needed. [7.]

The entire serverless computing workflow can be seen in figure 1. It is divided into steps: function programming and function serving.

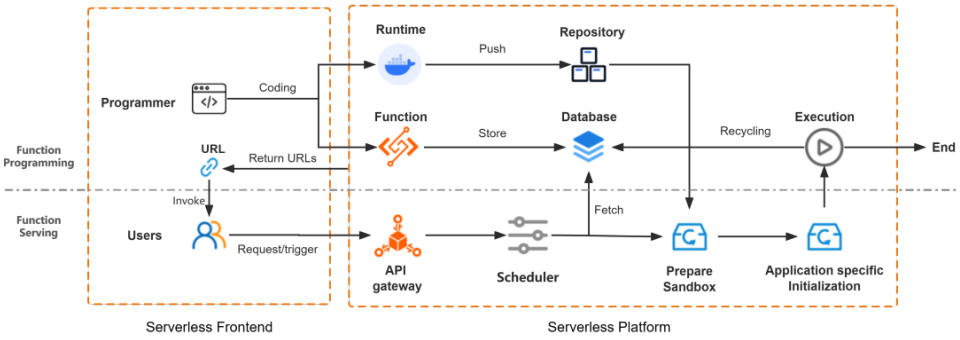

Figure 1: The Whole Workflow of Serverless Computing. Taken from Serverless Computing: State-of-the-Art, Challenges and Opportunities [8].

As shown in the above figure, in the first phase, programmers create functions by following the guidelines specified by cloud vendors. The functions are then deployed to the platform through the command line, which stores them in a database and pushes their processing performance into a storage system before returning the Hyperlinks to the programmers. In the subsequent phase, developers access the function's retrieved address or a pre-set trigger via an API gateway supplied by cloud vendors. The platform's scheduler then retrieves the function from a registry and creates a sandbox running environment by retrieving the execution speed of the function from a remote repository. Finally, the cloud vendor can create a suitable environment for the application and then perform the function. [8.]

2.2 Amazon Web Services

Amazon Web Services (AWS) is the world's most popular and desired cloud platform, providing more than 200 properly functioning products from locations around the globe [5]. Many diverse organizations from the largest enterprises such as Samsung, Unilever, and BMW to the fastest-growing start-ups such as Netflix, Airbnb are just a handful of the numerous customers AWS services globally that rely on it to save expenditure, promote innovation, and improve agility [5]. In the graphic below, AWS is positioned in the Leaders quadrant of Gartner Research's new 2021 Magic Quadrant for Cloud Infrastructure and Platform Services [9].

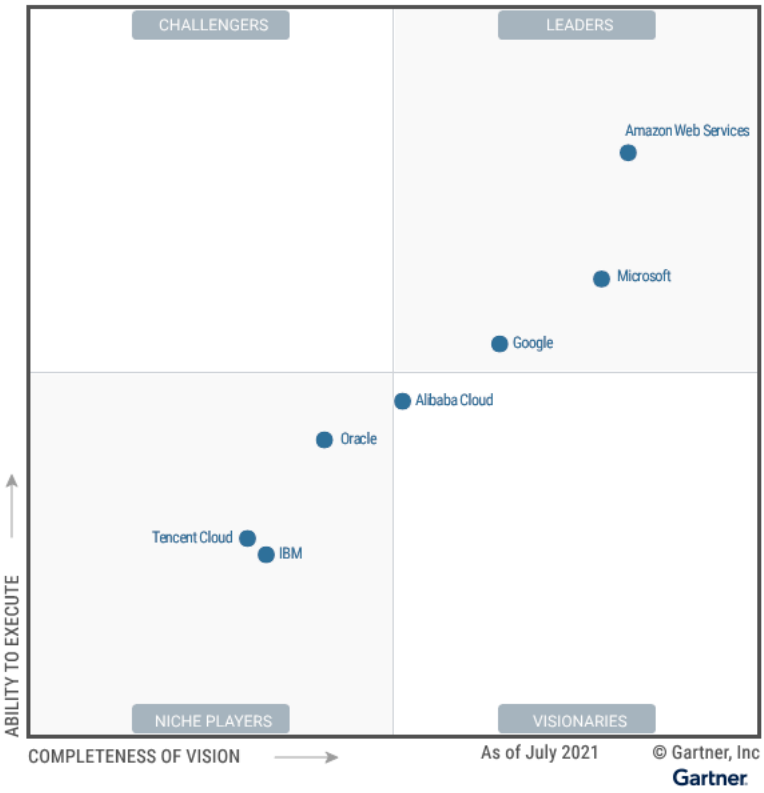

Figure 2: Magic Quadrant for Cloud Infrastructure and Platform Services. Taken from Magic Quadrant for Cloud Infrastructure and Platform Services [9].

Figure 2 shows that, in terms of market share, Amazon Web Services is in first place, followed by Microsoft, Google, and Alibaba. The widely known AWS applications include websites, mobile, gaming, and social apps, as well as storage and backup [10]. Figure 3 shows more of the AWS fundamental service elements.

Figure 3: The Main Component Services of AWS. Taken from AWS Fundamentals [10].

For almost all applications, whether old or new, compute, storage, and networking are fundamental and essential. When starting an application on AWS, the majority of clients should start with these. Many users wish to try out some of the fundamental features of these goods and services in order to explore and test out new solutions. AWS makes it simple for customers to create and experiment by providing such a broad selection of state-of-the-art technologies, including networking, artificial intelligence (AI), and content delivery, to mention a few. [11.]

2.3 AWS Amplify

AWS Amplify is a comprehensive revamp that was introduced in 2017 to simplify the building and distribution of apps. Additionally, there are code libraries, primed components, and embedded command line programs. The user experience, which is the most crucial element that must be considered while developing any application, is another factor in the introduction of AWS Amplify. AWS Amplify was created to unify the user experience across many platforms, including internet and mobile. Some advantages of AWS Amplify are easy and UI driven development, usage-based payment, backend support and web-based analytics. Building mobile and online applications can be done quickly, simply, and with a focus on the user interface thanks to AWS Amplify.

The functionalities that AWS Amplify provides that are most often used are authentication, store and sync data securely, quick backend updates, DataStore, deployment and app development. [12.]

2.3.1 Authentication

The primary authentication service provider for the Amplify Framework is Amazon Cognito. User sign-up, sign-in, and access control are all features that Amazon Cognito users can quickly and simply add to their online and mobile apps. Using SAML 2.0 and OpenID Connect, Amazon Cognito allows sign-in with social identity providers such as Amazon, Apple, and Facebook in addition to commercial identity providers. Users can easily create an authentication service from scratch or re-use their existing backend by Amplify commands such as amplify add auth or amplify auth update. Applications can implement authentication functionality in one of two ways: by leveraging pre-built UI components or by manually using authentication APIs. [13.]

2.3.2 DataStore

In order to assist clients build real-time and offline apps more rapidly, Amplify DataStore is an on-device storage engine that seamlessly synchronizes data between mobile and web apps and databases in the AWS cloud. Accessing distributed and shared data is feasible thanks to the programming language provided by Amplify DataStore, which eliminates the need to write distinct programs for each situation. By establishing a backend from their project, customers may begin synchronizing with the cloud once they are satisfied with their application. Using GraphQL as a data protocol, DataStore may connect to a remote backend and instantly sync all locally stored data. In addition, AWS Amplify users may limit which people or groups should be allowed to create, view, modify, or remove data by adding a @auth directive. [13.]

2.3.3 Hosting

AWS Amplify offers a fully managed solution with integrated CI/CD processes for the deployment and hosting of full-stack web apps, reducing the time between development and delivery. By simply linking the code repository for your application via the Amplify interface, modifications to users' frontend and backend are deployed in a single workflow whenever they contribute code. Amplify Hosting supports popular static site generators including Gatsby, Eleventy, Hugo, VuePress, and Jekyll in addition to SPA frameworks like React, Angular, Vue.js, Ionic, and Ember. In addition, it supports end to end testing and instant cache invalidations. Users can also connect their applications to custom domains. In addition to the features listed above, Amplify Hosting also supports several others. [14.]

3 Programming Languages Being Used in The Project

Programming languages are the instruments used to create computer-readable instructions. When choosing a programming language for a project, developers should consider many factors, including development speed, execution speed, ecosystem, and community. The programming languages JavaScript and TypeScript that were selected for the project will be further discussed in this section.

3.1 JavaScript

The internet grew rapidly in the early to mid-1990s. Some major companies such as Netscape and Microsoft were engaged in browser wars, including Netscape's Navigator against Internet Explorer [15]. In September 1995, Brandan Eich, a Netscape programmer, devised a new programming language in 10 days [16, p. 19]. It was first known as Mocha before being renamed LiveScript and later JavaScript [16, p. 19].

JavaScript is a scripting language that enables the integration of heterogeneous functions in a variety of contexts, including web development, mobile

development, and game creation among others. It may also be utilized to create a web application from scratch. Moreover, it facilitates the implementation of advanced features on web applications that are not possible with just Cascading Style Sheets (CSS) and Hyper Text Markup Language (HTML). [17.]

3.2 Typescript

TypeScript is an open-source programming language that compiles to JavaScript invented by Microsoft in 2012. Anders Hejlsberg, the inventor of C# and TurboPascal, is the most well-known person behind TypeScript. More precisely, a group of over 50 experts under the direction of Microsoft Technical Fellow Steve Lucco developed TypeScript. [18.]

The primary enhancement that TypeScript makes to JavaScript is the addition of a type system. Type is a term that refers to the purpose for which a piece of data is being used such as number, string, and array to name a few. In more technical terms, TypeScript provides the following:

- strict typing
- type annotations
- type inference. [19.]

Many mistakes are immediately caught as a result of stringent typing. By alerting programmers to the problems, TypeScript can help them avoid the most frequent mistakes, such as creating improper variable names and importing incorrect files. When substantial portions of the codebase are involved, this simplifies and expedites rewriting, which is good. However, TypeScript is much more than just a language for writing types, it introduces a slew of object-oriented programming-inspired syntax, such as inheritance, interfaces, classes, and more. [20.]

4 Frontend

The frontend part is a presentation layer of a website. It is something that nearly everyone encounters daily. It comprises everything a user sees and interacts

with immediately, from fonts and text to navigation bar and animations. As a result, frontend developers are primarily responsible for implementing interactive user interfaces. [21.]

4.1 React

There are more and more tools available for front-end development [22]. Certain instruments will be more suited for different tasks [22]. Simple apps might be made with just HTML, CSS, and JavaScript [22]. The availability of increasingly sophisticated tools for creating elaborate user interfaces increases as the program gets more complex [22]. Thus, React was chosen for this final project as an essential library to aid the developer in producing the application's pleasant and engaging design.

React is a well-known framework and open-source JavaScript project that focuses on helping programmers create user interfaces. Facebook initially released it in 2013 to address some of the issues that arise while creating a sizable, data-driven website [23, p. 1]. The primary reason why React was selected for the project is owing to its flexibility. Basically, components are the heart of React. Each component in React is a split chunk of code with its own logic and functions that may be reused across the program, significantly reducing development time [24]. In addition, instead of using normal Document Object Model (DOM), React uses virtual DOM, which is a lightweight copy of DOM, to improve its efficiency [25].

4.2 Redux

Redux is a predictable state container that is often used as a state management tool for applications [26]. The single source of truth, non-editable states, and updates being done with pure functions are three fundamental ideas that may be used to highlight the features of Redux [27]. In simple terms, Redux acts as a single store centralizing the state and logic of programs that aids in the creation of applications that are easy to test, operate consistently across multiple platforms (client, server, and native), and execute in a variety of

configurations [26]. The main advantage of combining Redux with React is that data will no longer be passed through multiple layers of components [28]. In addition, Redux’s states are consistently foreseeable [28]. A reducer, which is a pure function, will always provide the same result when given the same state and action [26].

5 Project Management

This section details the techniques and software tools that were applied to the project's management activities, including project planning, task distribution, issue tracking. Even though the thesis's sole author completed the final project, project management is still crucial because it ensures that the application's development is proceeding on schedule and increases productivity.

5.1 Source Code Version Control with Git and GitHub

Software code changes are tracked and managed using version control, commonly referred to as source control [29]. Development teams can manage source code changes over time with the help of version control systems [29]. Linus Torvalds, the well-known architect of the Linux operating system kernel, established the most well-liked contemporary version control system in the world, known as Git, in 2005 [30]. Git is used for version control in an incredible number of commercial and open-source software projects, and it functions nicely with a variety of IDEs and operating systems [28]. The primary attributes of Git are its code history repository, open-source nature, quick branching and merging, and distributed development. Git enables users to work remotely on a project from anywhere in the world. A user can choose any section of the project, do the necessary action, and then update the project again. This is possible thanks to Git's non-linear development tendency. Git also features a distributed mechanism that enables people to collaborate on the same project at the same time without interfering with one another's efforts. [31.]

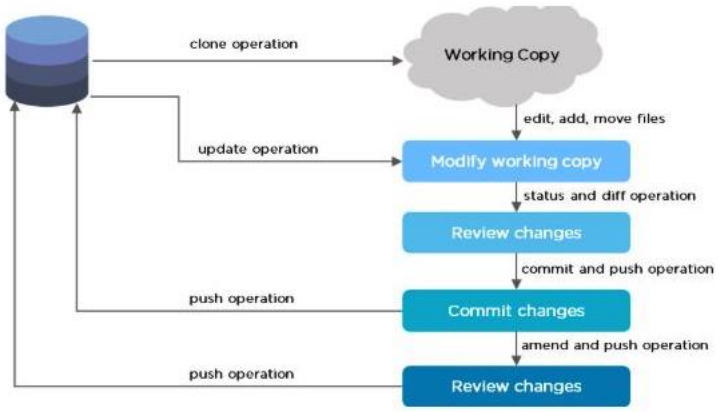

Figure 4: Git Workflow. Taken from What is Git: Features, Command and Workflow in Git [32].

Figure 4 depicts the fundamental Git workflow, which consists of the three stages of cloning, updating, and pushing. By cloning to your local computer, the repository will be downloaded in its entirety. Using commits, developers may intentionally and securely rewrite history. Pushing is the method used to transfer commits from your local repository to a remote repository.

GitHub is a platform that hosts code, allowing programmers and developers to post their work and collaborate with one another to make it better. Its strong version control system is one of GitHub's distinguishing characteristics. With the use of version control, programmers can make changes to any part of the project without being concerned about interfering with other people's work. These modifications to the software may be performed to address problems or boost performance. Following evaluation and approval, proposed improvements can be quickly incorporated into the current software. Forking, pull requests, and merging are three of the most important services that GitHub offers. [33.]

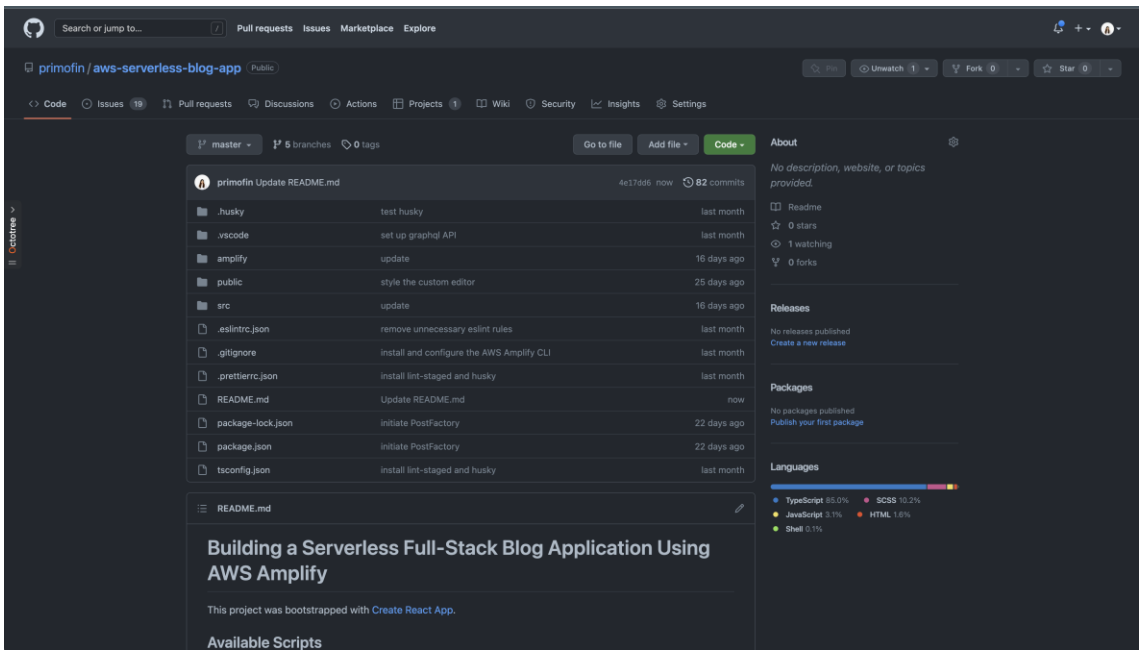

Figure 5: The Screenshot of the Project Repository Hosted in GitHub

Figure 5 depicts the project repository in GitHub in broad strokes. The repository page includes essential project elements such as code, code management procedures, project details, the languages being utilized in the project, and more.

5.2 Project Planning and Task Scheduling

The final product's workflow was managed using GitHub Projects, a tool for planning and organizing work on GitHub. It was decided to choose GitHub Projects over other management choices since it is completely compatible with other GitHub services and having everything in one location would be quite convenient. This section will offer an overview of the project planning phase and task management at a high level.

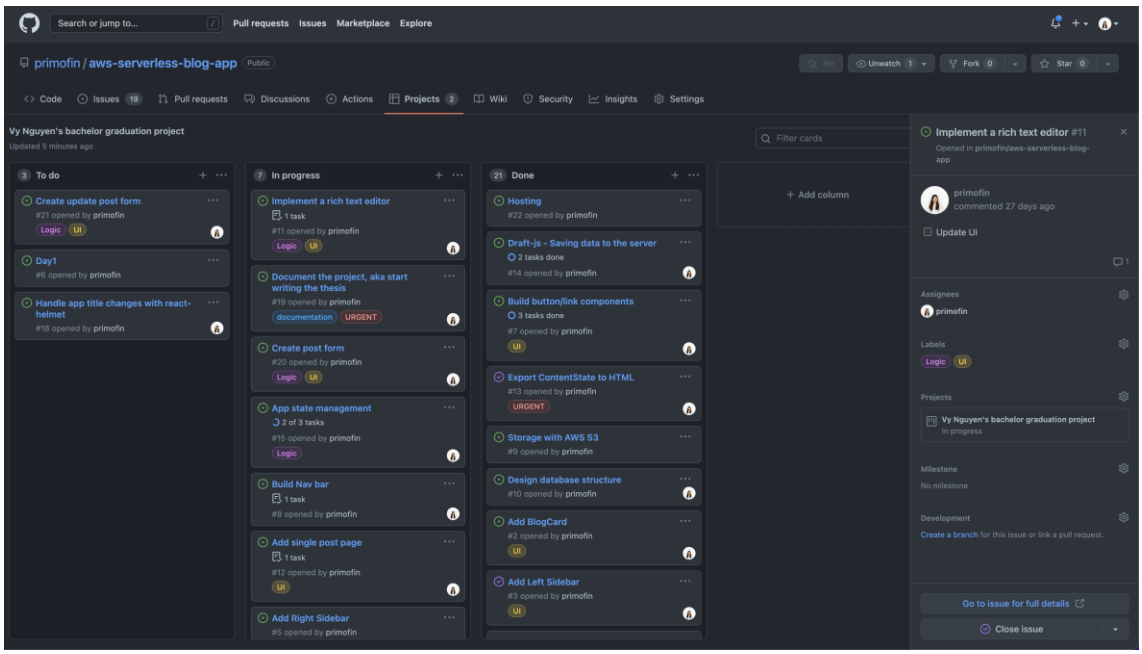

Figure 6: The Screenshot of the Projects Page

Figure 6 shows that the project board is divided into three columns: to do, in process, and done. The to do column may be viewed as the project's beginning point, where newly created tasks are stored. New tasks can be created by either clicking on the plus sign button on the to do column or clicking on new issue button in the issues page of the project. One of the most important features of the GitHub projects is the inclusion of triggers that automatically shift jobs around. For example, newly added issues can be moved automatically to the project board.

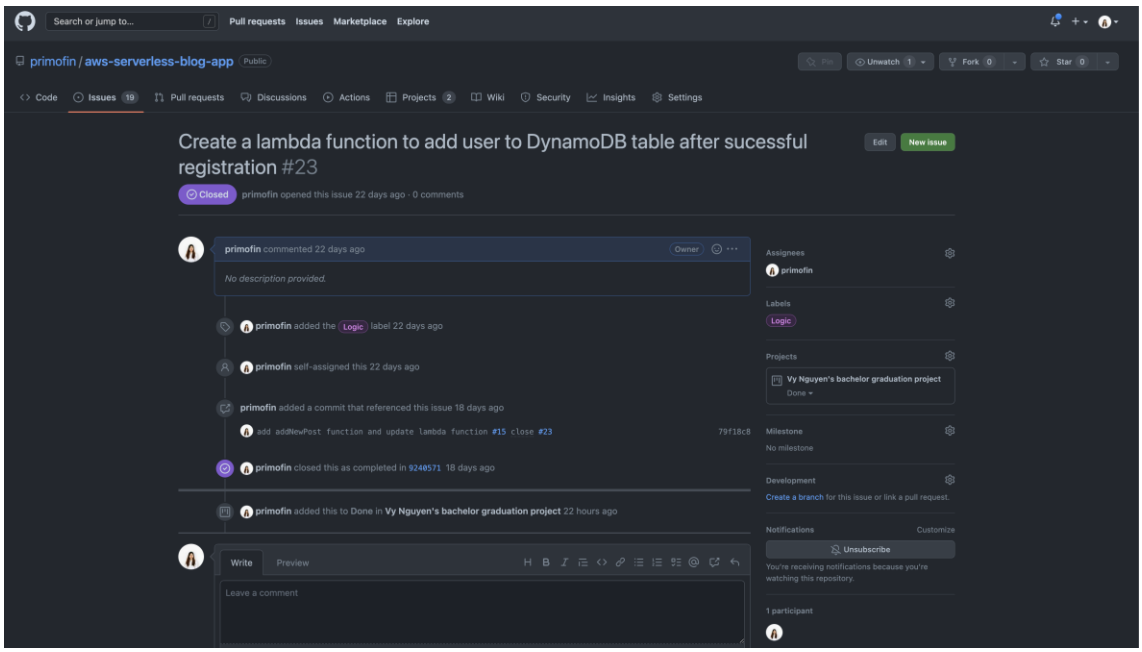

Figure 7: Screenshot of an Issue's Information page

A general description of an issue is provided in the figure above. An issue may be classified and allocated to one or more people by using certain descriptive labels. By specifying the issue's index in a commit and using keywords such as close, closed, and fixed, developers may also address issues in commits or automatically close issues [34].

6 Development Process

6.1 The Front-End Development

The client side of the application was bootstrapped using Create React App, which has the official support of the React team [33]. Create React App sets up a frontend build pipeline and preconfigures Webpack and Babel so developers may concentrate on the code [35]. To keep track of changes and roll back to a previous state in case of errors, a GitHub repository was set up at this stage of the project.

6.1.1 Styling

Instead of using plain CSS, SASS which is a CSS pre-processor was selected to add style to the application [34]. It makes it easier to share styles and keeps stylesheets organized since SASS allows developers access to a wide range of functionality, including functions, loops, mixins, mathematical operations, and variables [36]. See Figure 8 below.

Figure 8 The Screenshot of the Stylesheet Structure

The styling architecture of the project was structured based on a principle which is breaking the code into smaller chunks and separating them by scope and

mainly inspired by guidelines provided by Kitty Giraudel [37]. Thanks to those guidelines, codebase becomes more reusable, scalable, and maintainable.

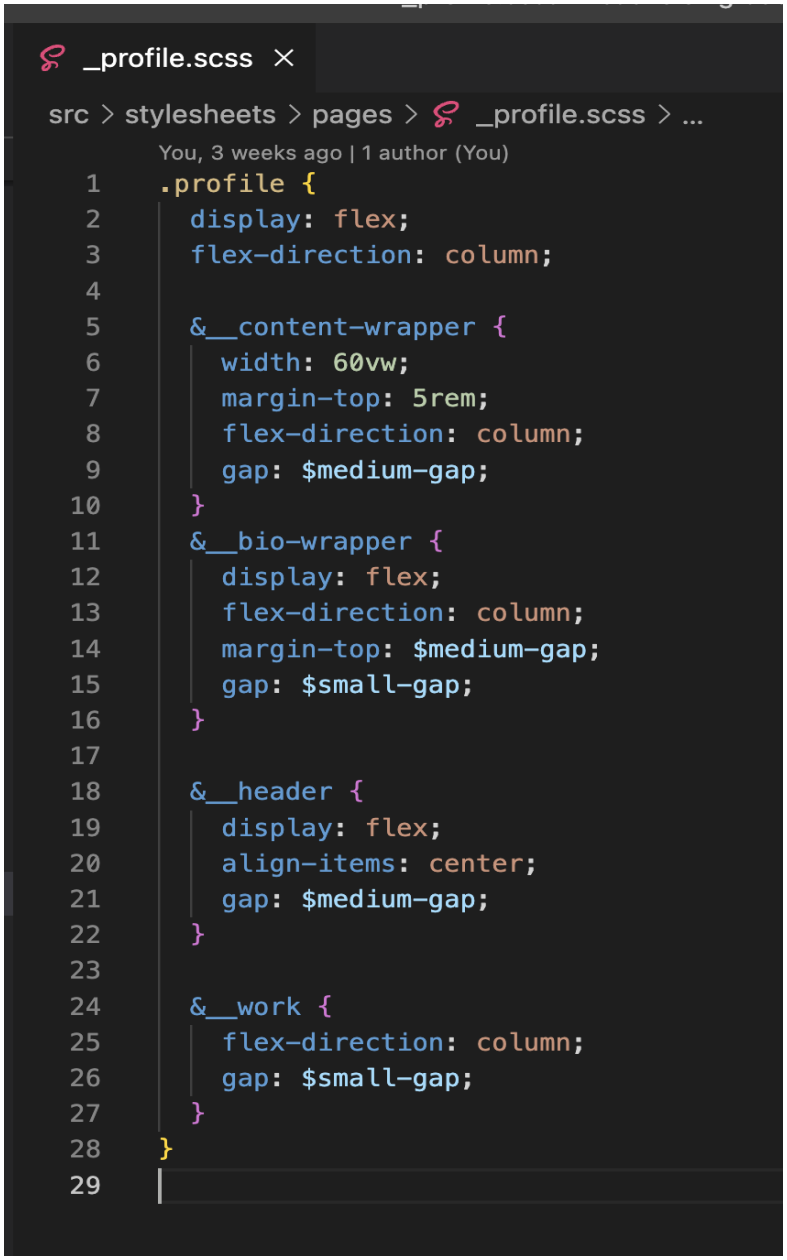

Figure 9: The Screenshot of the Profile Stylesheet. An Example of Using BEM Methodology.

Figure 9 gives an example of applying the BEM methodology in the project. BEM is a front-end naming method for organizing and naming CSS classes. The Block, Element, Modifier methodology is a popular naming convention for

class names in HTML and CSS [38]. With scoped classes, BEM may prevent developers from overriding classes accidentally.

6.1.2 Noteworthy UI Libraries Being Used in the Project

This section will go through some radical UI libraries being used in the project and their fundamental features. Those are Material UI, Formik, Amplify UI.

Following Google's 2014-launched Material Design, Material-UI (MUI) is a CSS framework that comes with pre-built React components [39]. MUI enables the usage of many components to build UIs for online and mobile apps for businesses [39]. To ensure that users have a uniform experience with its products regardless of how they engage with them, Google employs Material Design [39]. In addition to the React custom components, the project also employed the components Button and Popover from Material-UI. The primary purpose of Material UI was to accelerate the development cycle.

The project's whole form management was handled by Formik, the most widely used open-source form library for React and React Native in the world. Formik oversees the time-consuming and repetitive processes, including managing submission, managing validation, and monitoring values, errors, and visited fields. As a result, developers can focus more on the business logic while putting less work into setting up state and change handlers. [40.]

Amplify UI is an open-source design framework with virtualized elements and primitives can be integrated into any application and is completely customizable [41]. The project used Authenticator component from the library to add login functionality to the application.

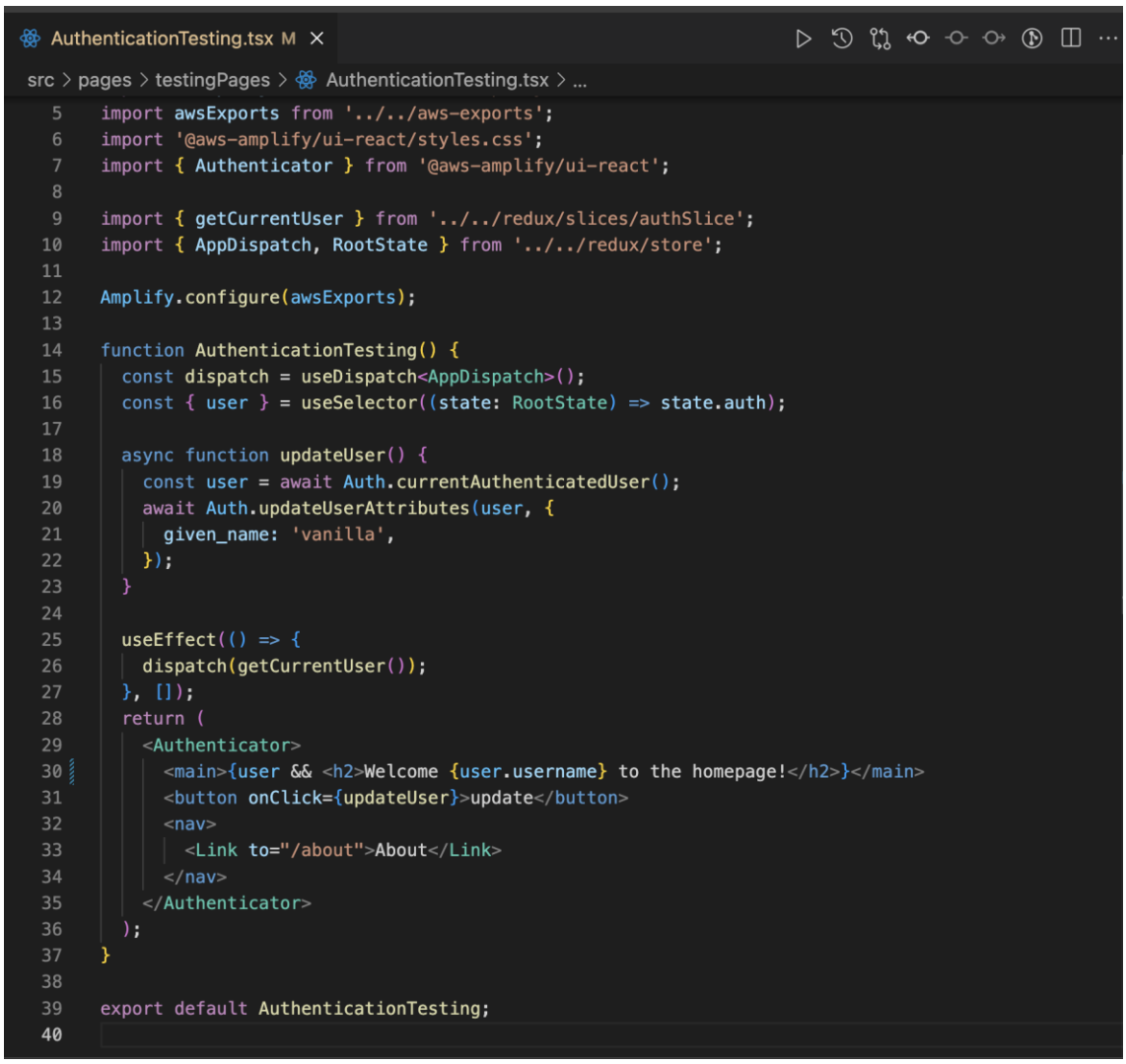

Figure 10: The Screenshot of the AuthenticationTesting Page. An example of the Usage of the Aws-amplify/ui-react.

As illustrated in Figure 10, the Authenticator component is quite straightforward to use, and it takes only a few lines of code to set up the authentication process.

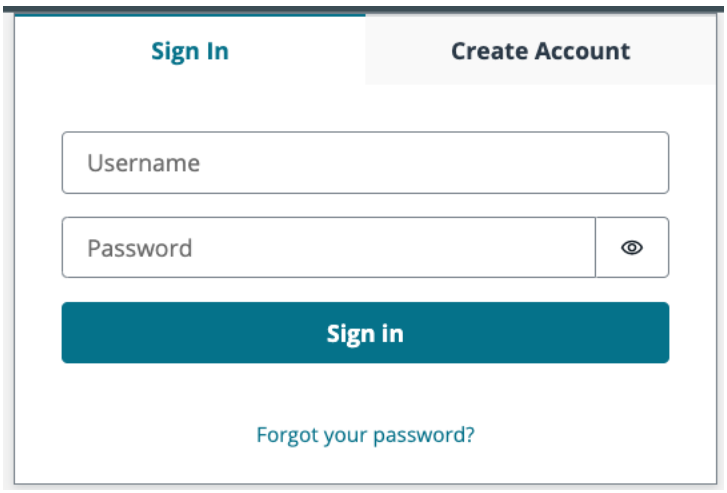

Figure 11: Screenshot of the Authentication Form.

Figure 11 shows how the Amplify UI-powered authentication form looks by default.

6.1.3 Developing a Rich Text Editor with Draft.js

Draft.js React officially introduced Draft.js in February 2006 at a conference. It is a framework for constructing sophisticated text editors in React, with an immutable model at its core and an abstraction layer that addresses cross-browser variants. Draft.js was built by Facebook to address several issues associated with recording user postings and comments, notably the use of text editing area and generic containers in HTML. In addition, Draft.js allows developers to construct any form of rich text input, from a few inline text styles to a full text editor for composing long entries. [42.]

Draft.js's two default properties, EditorState and onChange, serve as its cornerstone. The editor's current state, including its contents, cursor, and undo/redo history, is captured in EditorState. New EditorState objects are created for each change in content and selection made in the editor. The callback function onChange implements any changes made to the editor's DOM and communicates the modified state value to the editor core's top level using the current EditorState object. [43.]

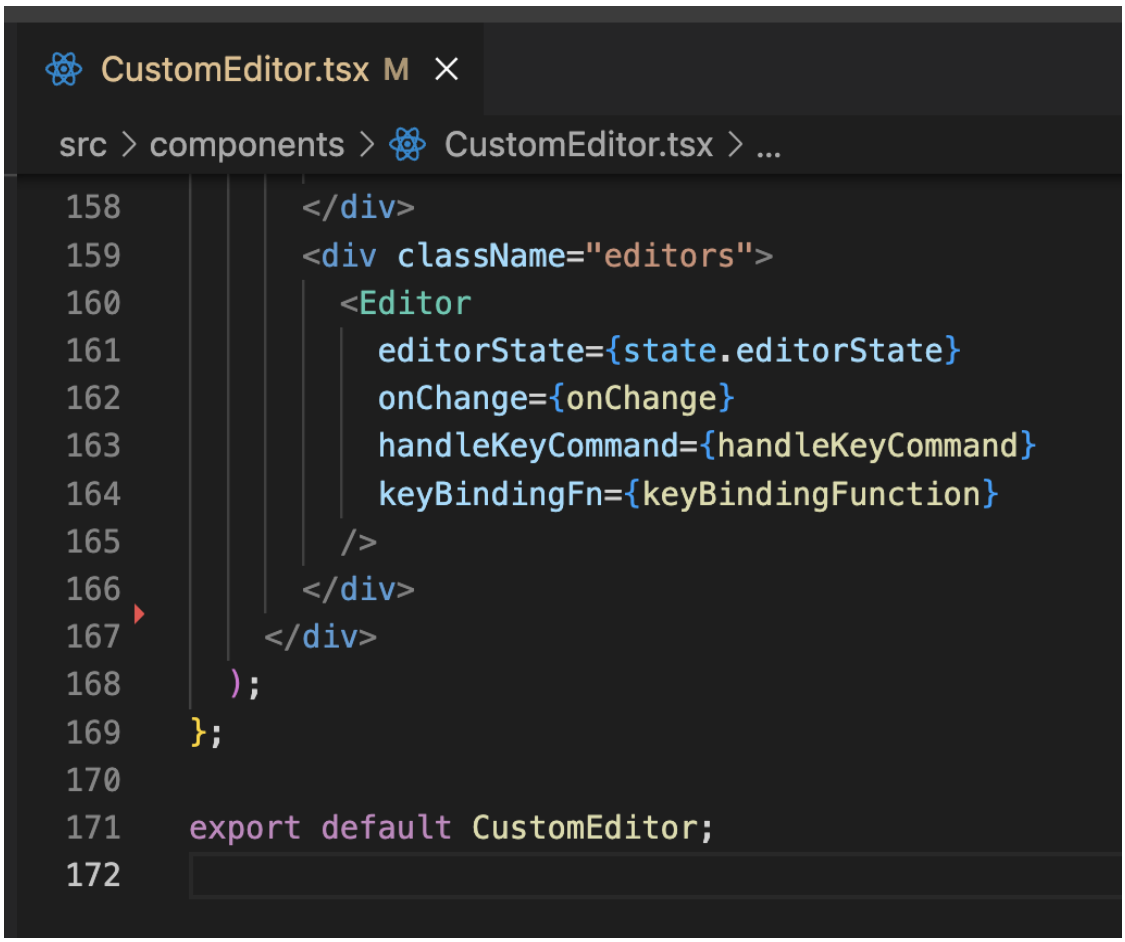

Figure 12: The Screenshot of a Part of the CustomEditor Component

An illustration of the use of Draft.js is shown in Figure 12. A fundamental Editor component is currently operational at this point in the project. In order to manage state, the project also made use of RichUtils, a tool supplied by Draft.js. Key combinations like Cmd+B (for bold), Cmd+I (for italics), and others are available to web editors [44]. Information about these key combinations is available from RichUtils [44]. Key presses may be detected and handled with the aid of the handleKeyCommand prop, which can then be connected to RichUtils to apply or remove the required style [44]. Moreover, users can alter styles within the editor by using a set of control buttons that have been added to the project. The whole source code for the CustomEditor file is included in Appendix 1.

6.2 The Serverless Backend Development

The backend of the project was mostly handled by AWS. The services from Amazon were used in the project are AWS Amplify, AWS Cognito, AWS Lambda, AWS AppSync, and Amazon DynamoDB. Following are the main steps to develop the backend of the application. Prior to using AWS Amplify, an AWS account had to be created. The Amplify CLI was then installed via the command line for the purpose of efficiency. A directory named amplify was created once the Amplify project was launched and it contained all the details about the backend of the application. The code templates in the folder will be updated as soon as new services are introduced.

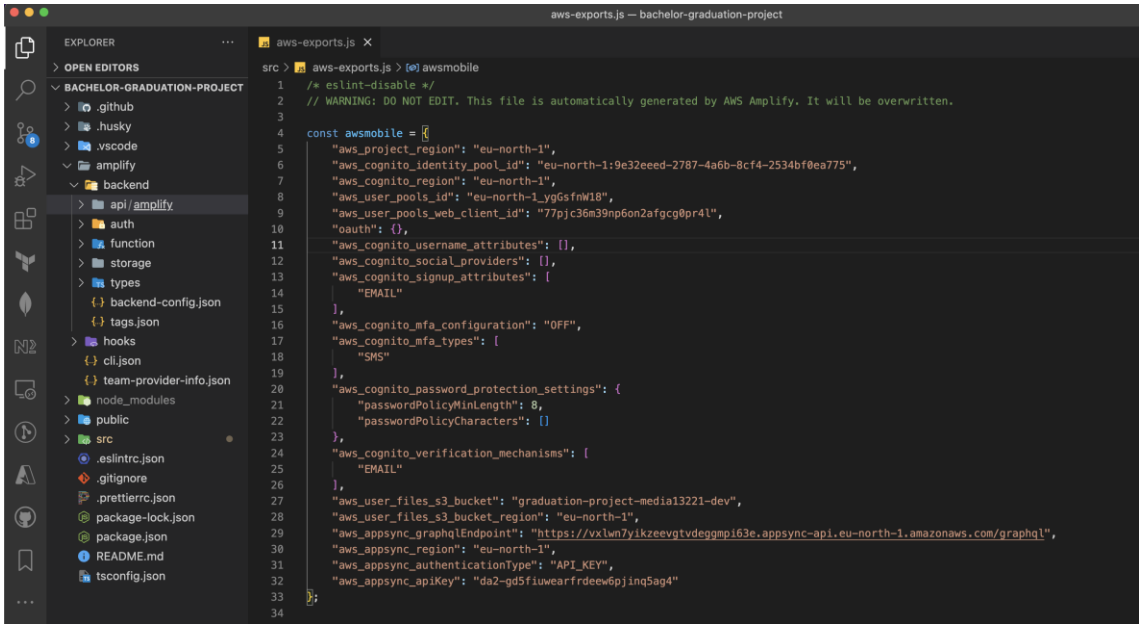

Figure 13: The Screenshot of the Amplify Directory and aws-export.js File.

The main backend modules, API, auth, function, and storage, are all located in the amplify folder, as seen in Figure 13. The configuration for the services built with Amplify was also stored in a file called AWS-exports, which also gives the frontend the information it needs on the backend on AWS Amplify.

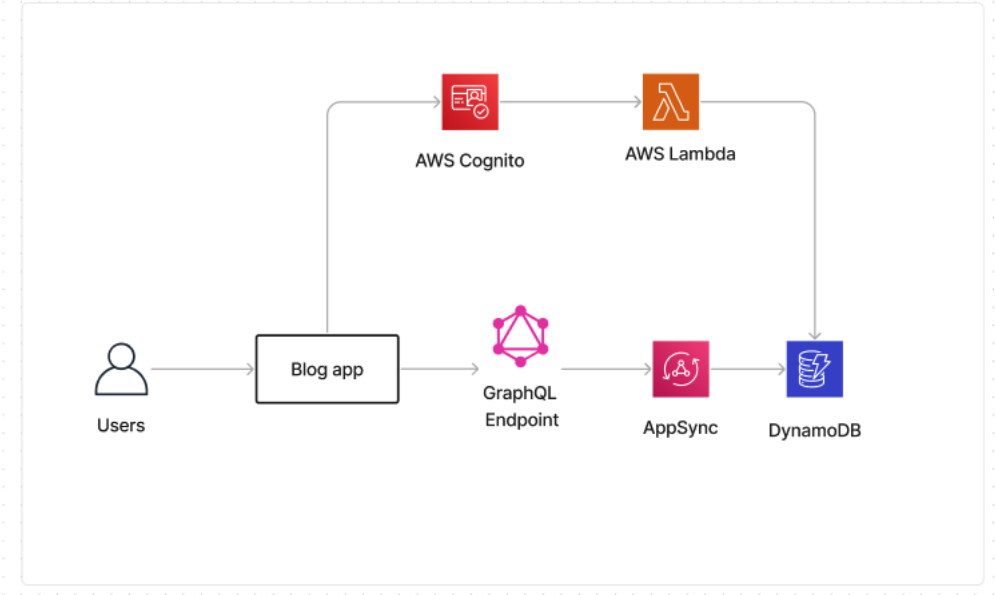

Figure 14: The Design of the Cloud Infrastructure of the Application. Screenshot Taken from the Design File.

Figure 14 shows the high-level architecture of the system. Firstly, authentication service was added to the project by running the command amplify add auth. Instead of manually communicating to the backend using AWS Amplify's authentication APIs, the application makes use of the Amplify UI components. Secondly, a new data storage was added through running the command amplify add storage. In this case, a NoSQL database DynamoDB was selected. Then, a GraphQL API was added by running amplify add api. Finally, all the services were deployed to the cloud with amplify push command. After the API deployment, the terminal would show a remote GraphQL endpoint address. Internally, for apps to easily get the precise data they want, the Amplify Framework makes advantage of AWS AppSync, a managed service that employs GraphQL [45]. AWS AppSync offers a powerful, scalable GraphQL interface that enables developers to mix data from many sources, such as HTTP APIs, Amazon DynamoDB, and AWS Lambda [46]. The implementation of GraphQL schema can be found in appendix 2.

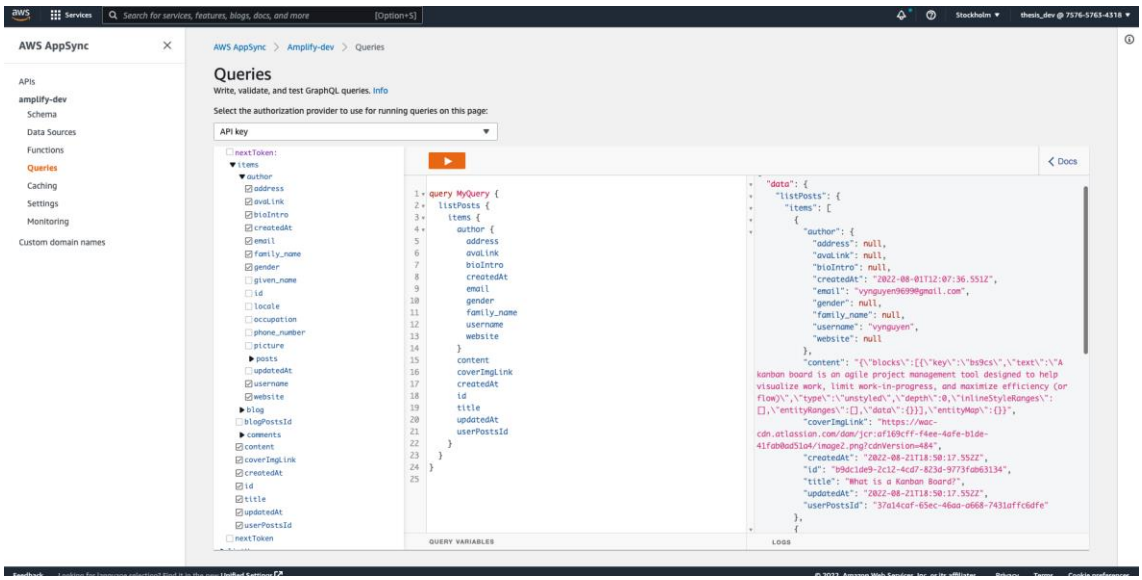

Figure 15: The Screenshot of the Interface of AWS AppSync of the Application.

AWS AppSync's visual is described in general in picture 15. Through AWS AppSync, users may do a variety of tasks, including examining or modifying schemas, obtaining data sources, and querying data, among other things.

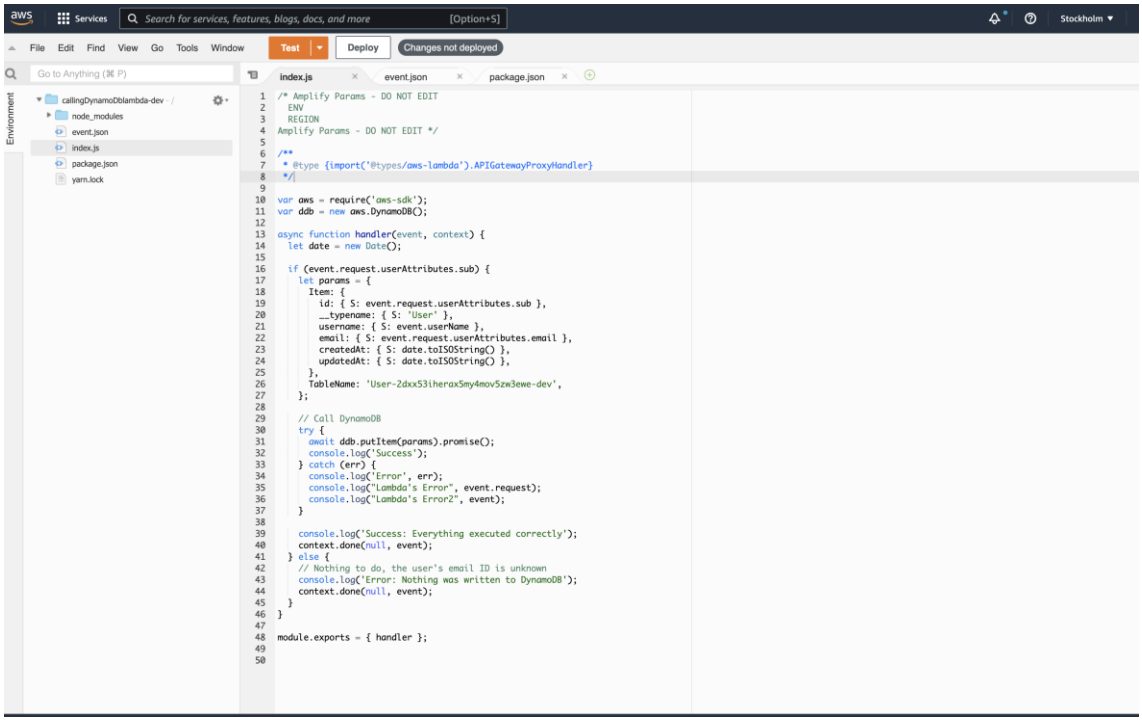

Figure 16: The Screenshot of the Content of callingDynamoDblamda Lambda Function.

Additionally, the application's DynamoDB table users' data was saved using the AWS Lambda service. Following the user's post confirmation, the function will be invoked. The intention was to keep the user pool from receiving an excessive number of special characteristics. More flexibility would also be available when handling user schema.

7 Results

The specific project outcomes are presented in detail in this chapter. The first and major accomplishment is the successful development and hosting of a blog application on the AWS cloud.

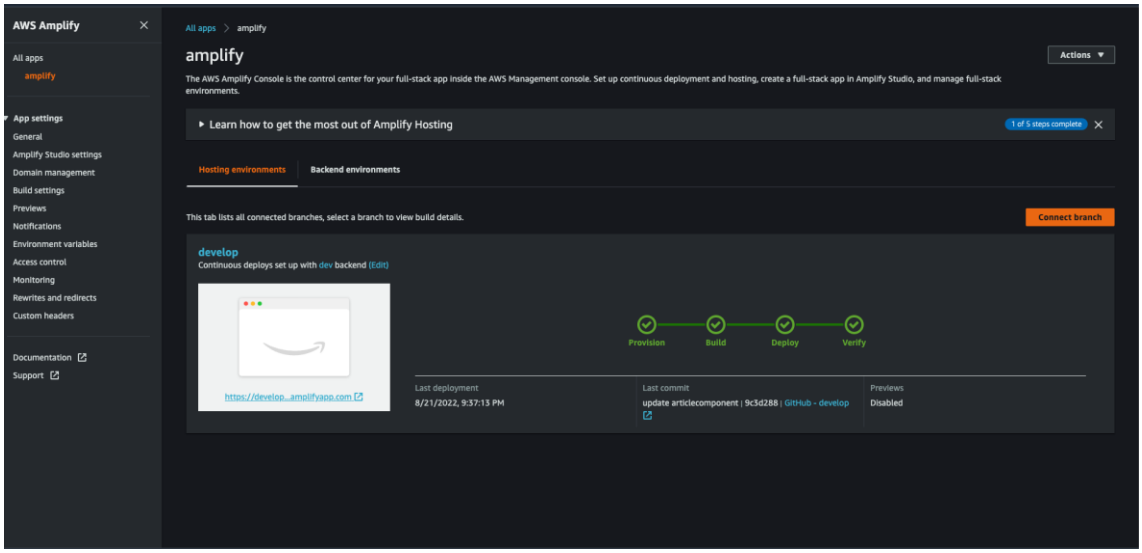

Figure 17: The Screenshot of the AWS Amplify.

Figure 17 shows that the application has been provisioned, created, deployed, and checked. Through the link supplied by AWS Amplify, users may visit the application, which is currently online.

The key features of the projects are:

- Users can view all public posts.
- Users can sign in, sign up or sign out.
- Users can filter posts by name, tag or author’s name.

- Authorized users can add a visualized post using a custom rich text editor.
- Authorized users can update or remove their own posts.

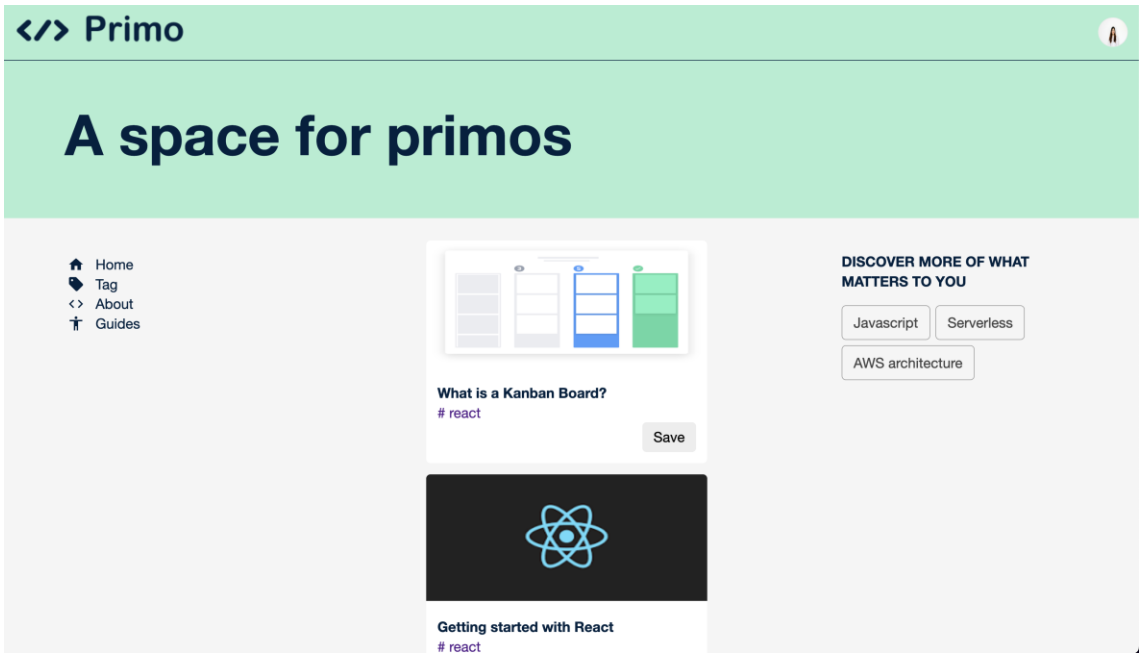

Figure 18: The Screenshot of the Main Page of the Application

Figure 18 shows a general look of the application's home page. The header, left sidebar, right sidebar, and app content are the page's four primary sections. Users may utilize the tags in the right sidebar to filter the material they want to see by clicking on them.

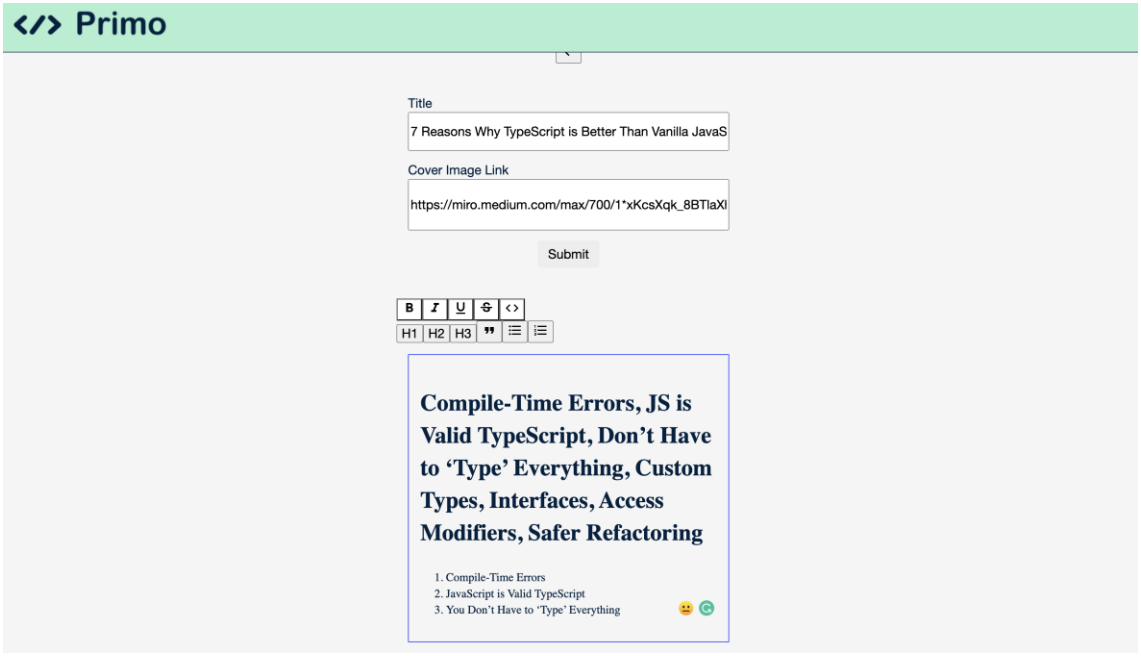

Figure 19: The Screenshot of the Application's Custom Editor

In addition to text, users have the option of contributing articles with organized material. A single post with a cover image, a heading-styled title, and an ordered list is constructed as an example in Figure 19.

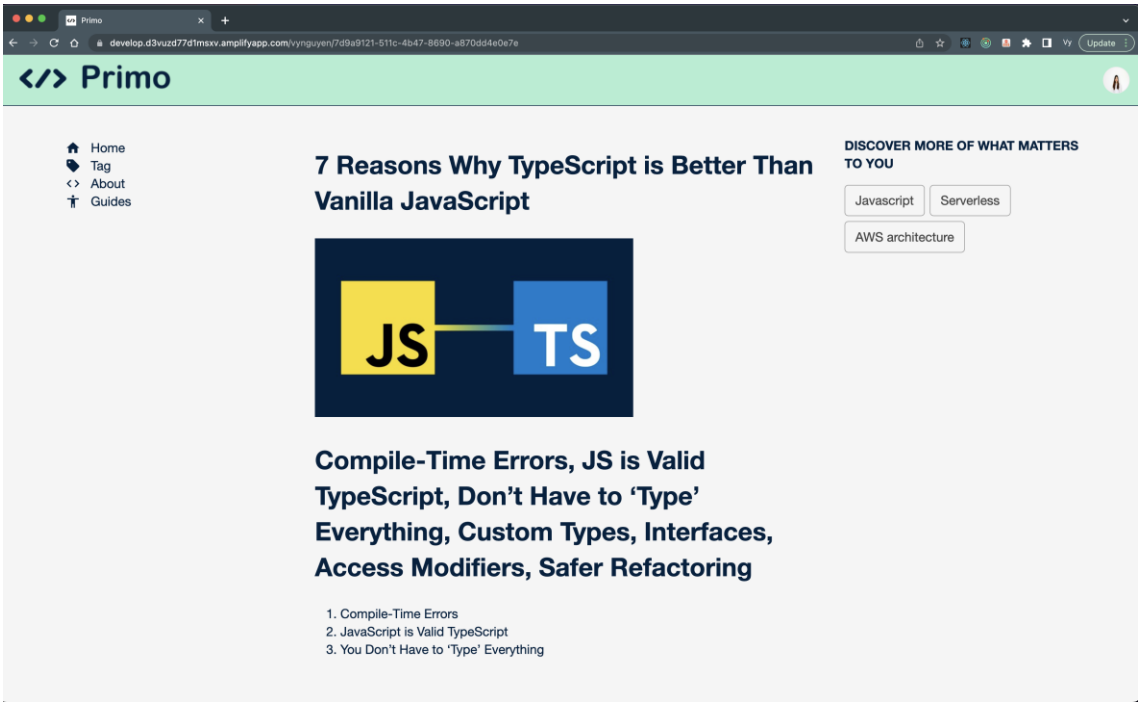

Figure 20: The Screenshot of a Dedicated Single Post Page.

An overview of a single post is shown in Figure 20. The post is organized according to user preferences.

8 Conclusion

The project's objective was to create a serverless blog platform leveraging AWS services, especially AWS Amplify. Such might be used to demonstrate the advantages of serverless architecture and its simplicity in development. The project resulted in the creation of a functioning application.

Many companies of all sizes and types operate their applications on AWS on a daily basis in this contemporary era of cloud computing. As a result, Amazon has made a large investment to encourage as many people as possible to utilize AWS. To make people's lives simpler, it offers a variety of tools and services. One such service for developers is AWS Amplify. Developers may quickly connect their applications, establish the app backends, and publish static web apps with the aid of AWS Amplify. Additionally, it enables the developer to handle content management outside of the AWS dashboard.

The application may add further capabilities such as commenting, reacting, sharing, and other features if the scope, resources, and time are available. Numerous cloud-based services may be employed to help manage the application more effectively. For instance, using AWS Budgets might make it easier to keep track of spending.

In conclusion, serverless computing offers a framework for swiftly creating and making marketable apps. AWS Amplify, in particular, is a simple-to-use tool for creating cloud-based web and mobile apps. Front-end web and mobile developers may create cutting-edge, feature-rich apps while using the strength of AWS services by utilizing a collection of tools and services.

References

1 IT Consulting. 2021. Traditional IT Infrastructure vs. Cloud Computing. Online. 6 December 2021. Centre Technologies. <https://www.forbes.com/sites/theyec/2020/02/03/why-every-business-needs-a-website>. Accessed 1 August 2022.

2 The Future of Data Center Growth. Online. 24 March 2022. DataSpan. <https://dataspan.com/blog/the-future-of-data-growth/>. Accessed 1 August 2022.

3 Macpherson, Jordan. 2022. Virtualization – Differences and Benefits. Online. 07 March 2022. Park Place Technologies. <https://www.parkplacetechnologies.com/blog/traditional-data-center-vs-virtualization-differences/>. Accessed 1 August 2022.

4 What is Cloud Computing? Online. Microsoft. <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing/#benefits/>. Accessed 1 August 2022.

5 Cloud Computing with AWS. Online. Amazon. <https://aws.amazon.com/what-is-aws/>. Accessed 1 August 2022.

6 Nočnica, Fee. 2020. What is Serverless Architecture? Key Benefits and Limitations. Online. 28 August 2020. New Relic. <https://newrelic.com/blog/best-practices/what-is-serverless-architecture>. Accessed 1 August 2022.

7 Nupponen, Jussi; Davide, Taibi. 2020. Serverless: What it Is, What to Do and What Not to Do. 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), May 2020, pp. 49-50. IEEE. Accessed 1 August 2022.

8 Yongkang, Li; Yanying, Lin; Yang, Wang; Kejiang, Ye; Cheng-Zhong, Xu. 2022. Serverless Computing: State-of-the-Art, Challenges and Opportunities. IEEE. Accessed 2 August 2022.

9 Raj, Bala; Bob, Gill; Dennis, Smith; David, Wright; Kevin, Ji. 2021. Magic Quadrant for Cloud Infrastructure and Platform Services. Online. 27 July 2021. Gartner. <https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802&st=sb/>. Accessed 5 August 2022.

10 AWS Fundamentals. Online. Simplilearn. <https://www.simplilearn.com/tutorials/aws-tutorial/aws-fundamentals/>. Accessed 5 August 2022.

11 Core Services and Additional Services. Online. Amazon. <https://docs.aws.amazon.com/whitepapers/latest/public-sector-cloud-transformation/core-services-and-additional-services.html/>. Accessed 6 August 2022.

12 Clark, Jessica. What is AWS Amplify? Online. Back4app. <https://blog.back4app.com/what-is-aws-amplify/#How_does_AWS_Amplify_work/>. Accessed 6 August 2022.

13 Amplify Libraries. Online. Amazon. <https://docs.amplify.aws/lib/q/platform/js/>. Accessed 6 August 2022.

14 Welcome to AWS Amplify Hosting. Online. Amazon. <https://docs.aws.amazon.com/amplify/latest/userguide/welcome.html/>. Accessed 6 August 2022.

15 DeGroat, T.J. 2019. The history of JavaScript: Everything you need to know. Online. 19 August 2019. Springboard. <https://www.springboard.com/blog/data-science/history-of-javascript/>. Accessed 6 August 2022.

16 Rauschmayer, Axel. 2021. JavaScript for impatient programmers. Electronic book. Exploring JS. <https://exploringjs.com/impatient-js/downloads/impatient-js-preview-book.pdf>. Accessed 6 August 2022.

17 Scott, Morris. Tech 101: What is JavaScript? Online. Skillcrush. <https://skillcrush.com/blog/javascript/>. Accessed 6 August 2022.

18 Foley, Mary. 2012. Who built Microsoft Typescript and why. Online. 5 October 2012. Zdnet. <https://www.zdnet.com/article/who-built-microsoft-typescript-and-why/>. Accessed 7 August 2022.

19 TypeScript for JavaScript Programmers. Online. 2022. Microsoft. <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>. Accessed 7 August 2022.

20 The Good and The Bad of Typescript. Online. 2020. Altexsoft. <https://www.altexsoft.com/blog/typescript-pros-and-cons/>. Accessed 7 August 2022.

21 Terra, John. How to Become a Front End Developer – Skills, Roles, Salary Explained. Online. Simplilearn. <https://www.simplilearn.com/how-to-become-a-front-end-developer-article#front_end_developer_salary/>. Accessed 8 August 2022.

22 Patel, Jeel. List of 10 Best Front End Frameworks to Use for Web Development. Online. 24 November 2021. Monocubed. <https://www.monocubed.com/blog/best-front-end-frameworks/>. Accessed 8 August 2022.

23 Banks, Alex & Porcello, Eve. 2017. Learning React – Functional Web Development with React and Redux. Sebastopol. O'Reilly Media.

24 Components and Props. Online. 2013. Facebook. <https://reactjs.org/docs/components-and-props.html/>. Accessed 10 August 2022.

25 Nyamador, Desmond. 2020. React's Virtual DOM Explained. Online. 23 October 2020. Pluralsight. <https://www.pluralsight.com/guides/react's-virtual-dom-explained>. Accessed 10 August 2022.

26 Abramov, Dan & the Redux documentation authors. 2022. Redux. Online. <https://redux.js.org/>. Accessed 10 August 2022.

27 Abramov, Dan & the Redux documentation authors. 2021. Three Principles. Online. 25 June 2021. <https://redux.js.org/understanding/thinking-in-redux/three-principles#single-source-of-truth/>. Accessed 10 August 2022.

28 Neo, Ighodaro. 2020. Why Use Redux? A Tutorial with Examples. Online. 12 October 2020. Pluralsight. <https://blog.logrocket.com/why-use-redux-reasons-with-clear-examples-d21bffd5835/#what/>. Accessed 10 August 2022.

29 What is Version Control? Online. Atlassian. <https://www.atlassian.com/git/tutorials/what-is-version-control/>. Accessed 12 August 2022.

30 What is Git. Online. Atlassian. <https://www.atlassian.com/git/tutorials/what-is-git/>. Accessed 12 August 2022.

31 Git Features. 2019. Online. GeekforGeeks. <https://www.geeksforgeeks.org/git-features/>. Accessed 12 August 2022.

32 Sayeda, Haifa Perveez. 2022. What is Git: Features, Command and Workflow in Git. Online. 12 July 2022. Simplilearn. <https://www.simplilearn.com/tutorials/git-tutorial/what-is-git/>. Accessed 14 August 2022.

33 Bradford, Laurence. 2020. What is GitHub. Online. 21 July 2020. The Balance Careers. <https://www.thebalancecareers.com/what-is-github-and-why-should-i-use-it-2071946/>. Accessed 14 August 2022.

34 Linking a Pull Request to an Issue. Online. GitHub. <https://docs.github.com/en/issues/tracking-your-work-with-issues/linking-a-pull-request-to-an-issue/>. Accessed 14 August 2022.

35 Create a New React App. 2013. Online. Facebook. <https://reactjs.org/docs/create-a-new-react-app.html/>. Accessed 15 August 2022.

36 Documentation. Online. Sass. <https://sass-lang.com/documentation/>. Accessed 18 August 2022.

37 Giraudel, Kitty. 2022. Sass Guidelines. Online. <https://sass-guidelin.es/>. Accessed 20 August 2022.

38 Robin, Rendle. 2015. BEM 101. Online. 2 April 2015. CSS-Tricks. <https://css-tricks.com/bem-101/>. Accessed 20 August 2022.

39 Sirotka, Alesia. 2022. What is Material UI. Online. 15 March 2022. FlatLogic. <https://flatlogic.com/blog/what-is-material-ui/>. Accessed 20 August 2022.

40 Build Forms in React, Without the Tears. Online. Formik. <https://formik.org/>. Accessed 20 August 2022.

41 Amplify UI. Online. NPM. <https://www.npmjs.com/package/@aws-amplify/ui-react/>. Accessed 20 August 2022.

42 Chavez, Kevin. 2020. Overview. Online. Draft.js. <https://draftjs.org/docs/getting-started/>. Accessed 20 August 2022.

43 Mahoney, Siobhan. 2018. Building a Rich Text Editor with React and Draft.js, Pt.1: Basic Set Up. Online. Medium. <https://medium.com/@siobhanpmahoney/building-a-rich-text-editor-with-draft-js-react-redux-and-rails-ef8d2e2897bf/>. Accessed 20 August 2022.

44 Tunde, Thomas. 2020. Rich Styling. Online. Draft.js. <https://draftjs.org/docs/quickstart-rich-styling/>. Accessed 20 August 2022.

45 Concepts. Online. Amazon. <https://docs.amplify.aws/lib/graphqlapi/concepts/q/platform/js/>. Accessed 10 September 2022.

46 What is AWS AppSync. Online. Amazon. <https://docs.aws.amazon.com/appsync/latest/devguide/what-is-appsync.html/>. Accessed 10 September 2022.

47 Robertson, Rose. Rich text editing on the web: Formatting text and keyboard shortcuts in Draft.js. Online. Dev.to. <https://dev.to/rose/rich-text-editing-on-the-web-formatting-text-and-keyboard-shortcuts-in-draft-js-4g9f />. Accessed 09 October 2022.

48 Client Code Generation. Online. Amazon. <https://docs.amplify.aws/cli-legacy/graphql-transformer/codegen/>. Accessed 09 October 2022.

Program Code – File CustomEditor.tsx

Certain parts of the file were constructed based on the instruction from the article Rich text editing on the web: Formatting text and keyboard shortcuts in Draft.js [47].

```
/**
 *
 * Metropolia University of Applied Sciences
 * Vy Nguyen Thanh 20 August 2022
 * Final Year Project
 * File:  CustomEditor.tsx
 *
 */

import { useEffect, useState } from 'react';
import {
  Editor,
  EditorState,
  RichUtils,
  getDefaultKeyBinding,
  KeyBindingUtil,
  convertToRaw,
  // ContentState,
  convertFromRaw,
} from 'draft-js';
import { stateToHTML } from 'draft-js-export-html';
import Icon from '@mui/material/Icon';
// import parse from 'html-react-parser';

import { blockTypeButtons, inlineStyleButtons } from '../data/editorData';

type CustomEditorState = {
  editorState: EditorState;
  editorContentHtml?: string;
};

type CustomEditorProps = {
  // eslint-disable-next-line @typescript-eslint/no-explicit-any
  handleContentChange: any;
};

function keyBindingFunction(event: React.KeyboardEvent<HTMLElement>): string |
null {
  if (KeyBindingUtil.hasCommandModifier(event) && event.shiftKey && event.key
=== 'x') {
    return 'strikethrough';
  }

  if (KeyBindingUtil.hasCommandModifier(event) && event.shiftKey && event.key
=== '7') {
    return 'ordered-list';
  }

  if (KeyBindingUtil.hasCommandModifier(event) && event.shiftKey && event.key
=== '8') {
    return 'unordered-list';
  }

  if (KeyBindingUtil.hasCommandModifier(event) && event.shiftKey && event.key
=== '9') {
    return 'blockquote';
  }

  return getDefaultKeyBinding(event);
}

const CustomEditor = ({ handleContentChange }: CustomEditorProps) => {
  const [state, setState] = useState<CustomEditorState>({
    editorState: EditorState.createEmpty(),
  });

  const onChange = (editorState: EditorState) => {
```

```
const contentState = editorState.getCurrentContent();
    // saveContent(contentState);
    setState({
      editorState,
      editorContentHtml: stateToHTML(contentState),
    });
    handleContentChange(JSON.stringify(convertToRaw(contentState)));
  };

  const handleKeyCommand = (command: string) => {
    // inline formatting key commands handles bold, italic, code, underline
    const editorState = RichUtils.handleKeyCommand(state.editorState,
command);

    // CMD + Shift + X
    if (!editorState && command === 'strikethrough') {
      onChange(RichUtils.toggleInlineStyle(state.editorState,
'STRIKETHROUGH'));
    }
    // CMD + Shift + 9
    else if (!editorState && command === 'blockquote') {
      onChange(RichUtils.toggleBlockType(state.editorState, 'blockquote'));
    }
    // CMD + Shift + 7
    else if (!editorState && command === 'ordered-list') {
      onChange(RichUtils.toggleBlockType(state.editorState, 'ordered-list-
item'));
    }
    // CMD + Shift + 8
    else if (!editorState && command === 'unordered-list') {
      onChange(RichUtils.toggleBlockType(state.editorState, 'unordered-list-
item'));
    }

    if (editorState) {
      setState({ editorState });
      return 'handled';
    }

    return 'not-handled';
  };

  const toggleInlineStyle = (event: React.MouseEvent<HTMLButtonElement>) => {
    event.preventDefault();

    const style = event.currentTarget.getAttribute('data-style');
    onChange(RichUtils.toggleInlineStyle(state.editorState, style!));
  };

  const toggleBlockType = (event: React.MouseEvent<HTMLButtonElement>) => {
    event.preventDefault();

    const block = event.currentTarget.getAttribute('data-block');
    onChange(RichUtils.toggleBlockType(state.editorState, block!));
  };

  const renderBlockButton = (value: string, block: string, icon?: string) => {
    return (
      <button type="button" key={block} data-block={block}
onMouseDown={toggleBlockType}>
        {icon ? <Icon>{icon}</Icon> : value}
      </button>
    );
  };

  const renderInlineStyleButton = (style: string, icon: string) => {
```

```
    const isActive = state.editorState.getCurrentInlineStyle().has(style);
    return (
      <button
        style={{
          backgroundColor: isActive ? 'black' : 'white',
        }}
        type="button"
        key={style}
        data-style={style}
        onMouseDown={toggleInlineStyle}
      >
        <Icon>{icon}</Icon>
      </button>
    );
  };

  // const saveContent = (content: ContentState) => {
  //   window.localStorage.setItem('content',
JSON.stringify(convertToRaw(content)));
  // };

  useEffect(() => {
    const content = window.localStorage.getItem('content');
    if (content) {
      setState({
        editorState:
EditorState.createWithContent(convertFromRaw(JSON.parse(content))),
      });
    } else {
      setState({ editorState: EditorState.createEmpty() });
    }
  }, []);

  return (
    <div>
      <div>
        {inlineStyleButtons.map((button) => {
          return renderInlineStyleButton(button.style, button.icon);
        })}
      </div>

      <div>
        {blockTypeButtons.map((button) => {
          return renderBlockButton(button.value, button.block, button.icon);
        })}
      </div>
      <div className="editors">
        <Editor
          editorState={state.editorState}
          onChange={onChange}
          handleKeyCommand={handleKeyCommand}
          keyBindingFn={keyBindingFunction}
        />
      </div>
    </div>
  );
};

export default CustomEditor;
```

Program Code – File schema.graphql

Several parts of the file include preset templates provided by AWS Amplify [48].

```
# /**
#  *
#  * Metropolia University of Applied Sciences
#  * Vy Nguyen Thanh 10 August 2022
#  * Final Year Project: Building a Serverless Full-Stack Blog Application
Using AWS Amplify
#  * GraphQL Schema
#  *
#  */
input AMPLIFY { globalAuthRule: AuthRule = { allow: public } }

type User @model {
  id: ID!
  username: String!
  email: String!
  posts: [Post] @hasMany
  picture: String
  given_name: String
  family_name: String
  gender: String
  address: String
  phone_number: String
  website: String
  locale: String
  occupation: String
  bioIntro: String
  avaLink: String
}

type Blog @model {
  id: ID!
  name: String!
  posts: [Post] @hasMany
}

type Post @model {
  id: ID!
  title: String!
  content: String
  coverImgLink: String
  blog: Blog @belongsTo
  author: User @belongsTo
  comments: [Comment] @hasMany
}

type Comment @model {
  id: ID!
  post: Post @belongsTo
  content: String!
}
```