



Ollscoil
Teicneolaíochta
an Atlantaigh

Atlantic
Technological
University

Faithpal

Project Engineering

Year 4

Enoch Abiodun

Bachelor of Engineering (Honours) in Software and
Electronic Engineering

Atlantic Technological University

2023/2024

MYFAITHPAL

Bible Feed Questions Chat Login/Register

Genesis 1:1

In the beginning, God created the heavens and the earth.

Start Reading



Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Atlantic Technological University.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

Enoch Abiodun

Acknowledgements

In this section, I would like to express my gratitude, first and foremost, to God for giving me the strength, courage, knowledge, and wisdom to make it this far in this course. Secondly, I would like to express my deepest and sincerest gratitude to everyone who has contributed to the successful completion of this final-year project.

I would like to acknowledge and deeply appreciate all my lecturers, who have been not only teachers but also role models during the five years I participated in this course. I would like to give special thanks to Paul Lennon, Niall O'Keefe, Brian O'Shea, Ben Kinsella, and my supervisor, Michelle Lynch.

Lastly, I would like to thank my family and my girlfriend for supporting me financially and emotionally during my final year of college, when I was not working and instead devoted my time to my studies.

Table of Contents

1	Summary	7
2	Poster	8
3	Introduction	9
4	Development Process	10
5	Project Architecture.....	11
6	Project Planning and Version control	12
	6.1.1 Version Control	15
7	Front-End Development	16
	7.1 React JS	16
	7.2 Next JS.....	17
	7.3 CSS.....	18
	7.4 MUI.....	18
8	Back-End Development.....	20
	8.1 Node JS.....	20
	8.2 Express JS	21
	8.3 MongoDB	22
	8.4 Socket.io.....	23
	8.5 LangChain.....	24
	8.6 Cheerio.....	25
9	Cloud Development	27
	9.1 Amazon Web Services – S3 Bucket.....	27
	9.2 Amazon Web Services – EC2 Instances.....	28
	9.3 Porkbun.....	29

9.4	Nginx reverse proxy and SSL	29
10	Programming Process	30
10.1	Layout (Next JS App Router)	30
10.2	Next Authentication (NEXTAUTH)	32
10.3	Bible Page.....	39
10.3.1	Dropdown	40
10.3.2	Display Passage	43
10.3.3	Bible Route and API	46
10.4	Feed Page.....	50
10.4.1	Create Posts	52
10.4.2	Posts.....	57
10.5	Question Page & Chat page.....	59
10.5.1	Chat Context API	60
10.6	AI Configuration	64
10.7	Profile Page	72
11	Ethics	74
12	Conclusion.....	75
13	References	76

1 Summary

For my final year project, I desired to make an application that would be meaningful to me and not just another project I had to force myself to do. So, recently I took a leap of faith to explore Christianity and get to know God more. It was an amazing experience and journey, but I believed it would have been easier if I had a community and learning resources that would have aided me during this period. This thought led me to envision a web application that could connect people new and old in faith, near and wide by having a chat, feed and other features that provides a sense and hope and security in faith that you're not the only one. Furthermore, to encompass the learning resources side of thing the app would have the bible the first source any Christian should have as a learning source but with study tools allowing you to highlight, bold and make notes. This is everything that I envisioned but Faithpal includes even much more, not only the bible but the app would have the present time infamous Artificial Intelligence that has even caused a ruckus in Christian Society but hopefully with the AI tailored to answer questions tailored to Christianity it would solve the problem. I believe it would also help Christians create an environment for themselves where they can truly put all there focus on Jesus and grow as a Christian. How? As with the Social features it steals traffic from sites like Facebook [3] and Instagram [2] but with the bonus that these features are God-centered and your Bible with learning resources are just a click away.

The technical section of this report documents the process in the development of Faithpal. The web application was built upon the framework Next JS [6] and Express JS [8] including libraries, runtimes, and servers such as Node JS [7], React JS [5], Socket.io [11], Material UI, AWS EC2 [20], AWS S3 Bucket [19], Nginx [23], and MongoDB Atlas [17]. To keep track of progress I opted to use Microsoft one note to document my weekly progress, Jira [4] for project planning and GitHub to track and save code. In comparison to my project proposal written at the beginning of first semester, I am glad to announce that I managed to complete all that I set out to achieve and a total of three weeks ahead of the timeline. The application includes features like User authentication, Real-time Chat, AI, Posts, Bible reading and study tools and profile editing. The report aims to give an overview of the project. It explains the technologies at use, the features implemented, and the potential of Faithpal.

2 Poster



FaithPal

Summary

"Faithpal" is a Christian web application designed to support Christians in their spiritual journey. It features essential components such as a Bible page, Feed, Questions page, Chat page, and Profile page with Authentication.

The Bible page allows users to highlight and comment on passages, facilitating deeper study and reflection. The Questions section employs AI Language Models to provide answers to queries related to Christianity. Additionally, social features akin to Instagram enable users to connect with fellow Christians, fostering a sense of community and support.

Technologies

- React - JavaScript library
- NextJS - React Framework
- Material UI - React UI tool
- CSS - Styling
- NodeJS - JavaScript Runtime Environment
- ExpressJS - Web API Application for NodeJS
- Amazon AWS services
 - EC2 (Elastic Compute Cloud)
 - S3 (Simple Storage Service)
- Nginx - Serves Static Content and acts as a Reverse Proxy
- PM2 - Process Manager for NodeJS
- Socket.io - Enables Real Time communication between clients and server
- OpenAI - AI Model
- Langchain - Framework to build Applications with large language models (LLMs)

HTTPS Communication Response/Requests

Block Diagram

Enoch Abiodun
G00380316

Software and Electronics Engineering



Ollscoil
Teicneolaiochta
an Atlantaigh

Atlantic
Technological
University

Results

MYFAITHPAL
Bible Feed Questions Chat
Profile

RabbiGpt

Hello! How can I assist you in relation to the Bible or Christianity today?

Yesterday at 5:53 AM

What does the bible say about empathy?

The Scriptures refer to the quality of empathy, which we see demonstrated in several biblical narratives. Empathy is the capacity to feel another person's emotions, thoughts, and experiences. In the New Testament, Peter counseled Christians to have "compassion for one another; love as brothers, be tenderhearted, be courteous" (1 Peter 3:8, NKJV). The apostle Paul also encouraged empathy when he exhorted fellow Christians "to live with those who refuse; mount with those who mount" (Romans 12:15).

Empathy is related to sympathy but is narrower in focus and is generally considered more deeply personal. Compassion, sympathy, and empathy all have to do with having passion (feeling) for another person because of his or her suffering. True empathy is the feeling of actually participating in the emotions of another.

Ask your Rabbi-AI a Question! Don't be shy.

MYFAITHPAL

Clear Genesis Chapter 1 Select a Verse

THE EARTH WAS FORMLESS AND EMPTY. DARKNESS WAS ON THE SURFACE OF THE DEEP AND GOD'S SPIRIT WAS HOVERING OVER THE SURFACE OF THE WATERS.

God is good

Genesis 1

In the beginning, God created the heavens and the earth.

2 The earth was formless and empty. Darkness was on the surface of the deep and God's Spirit was hovering over the surface of the waters.

3 God said, "Let there be light," and there was light.

4 God saw the light, and saw that it was good. And divided the light



Go to <https://myfaithpal.com> on a Desktop or IPad to try Faithpal

Page 8 of 77

3 Introduction

Faithpal is a web application originating from the idea to be a friend like tool in a person's journey with Christ Jesus and has taken inspiration from the well-known mobile app YouVersion Bible which has a limited web presence only receiving a handful of its mobile phone app counterpart features. I would like to say this is where Faithpal steps in, but I believe Faithpal does much more than fill in gaps. With features such as Real time chat and a feed page that not only allows you to share bible verses and highlights but unlike YouVersion feed page allows the user complete freedom on the text and the image that they post. Furthermore, the user has more features at hand to learn and increase their faith and knowledge with the Open AI LLM and my own custom LangChain [13] configuration built on GPT Conversational Models and receiving web scraped data from GotQuestions.org.

To gain full access to all Faithpal features, a user must have an account and signing in is a requirement. On visiting the website, a user will only have access to the Home, Bible, and Feed page and of course the Login and Register page. But user will not be able to create posts or interact with any post just view them. If the user then wishes to register they can do so by Google [29] or by entering details such as Username, email, password, and name. Then they will be greeted by the Home page that will now give them a random verse from the bible instead of the first verse of the Bible Genesis 1:1. Users are then free to navigate to the different pages and update their profile and interact with other users on the application and the AI.

The project utilises numerous technologies such as Next JS [6], React JS [5], Express JS [8], Next Auth with JWT, MongoDB Atlas [17], OpenAI [14], LangChain [13], Cheerio [25], Socket.io [11], AWS S3 Bucket [19], AWS EC2 [20], Ruby, PM2 [21] and Nginx [23].

These technologies coexist to create a Christian user-friendly environment that acts as support or amplifies the Christian journey. The project aims to aid users, In Bible education and help Christians know more about God whilst creating fellowship with other people that are trying to increase in faith and are seeking God.

4 Development Process

The development process was one of my main focuses at the begin of the project. Building a web application involves a comprehensive development process the can be very simple or very complicated. These factors are and have been proven crucial for successful delivery of the functionality of my project. Initially, I conducted research to understand the achievability of the idea and proposal I made at the beginning of the year of what features to include in Faithpal. This insight led to planning phase , where I defined my project goals, scope, and requirements when I was able to gauge a realistic view of what I would be able to get done according to what I had writing on my proposed timeline. Following this, I had a UI/UX design of the home page to give a sense of the overall visual design of my application, but I was still missing a development methodology to keep track of progress. The methodology of software development I chose was Agile that I was exposed to in my 3rd year work experience with Fidelity Ireland as an Intern. There we had daily standup and often checked our Jira [4] daily to see if there were any updates on the progress of things named Stories or Issues or Epics that can be simplified to tasks. This would be paired with version control in a CICD (Continuous Integration and Delivery/Deployment) using useful comments and reviewing on every pull request. In contrast, this was a solo project where I was the sole Engineer, so I incorporated some of the principles but not all. This involved participating in weekly stand-up meetings with classmates and consistently checking my Jira [4] every day, making major updates to it on a selected day that was Sunday every week and committing work to GitHub at a frequent rate with commit messages that express fully indicated the work that was done in that commit. I tried to set a two-week limit as if were a sprint on how long I would spend on tasks but ultimately unlike a sprint I just was continuously working on new tasks even if the two weeks was not up.

5 Project Architecture

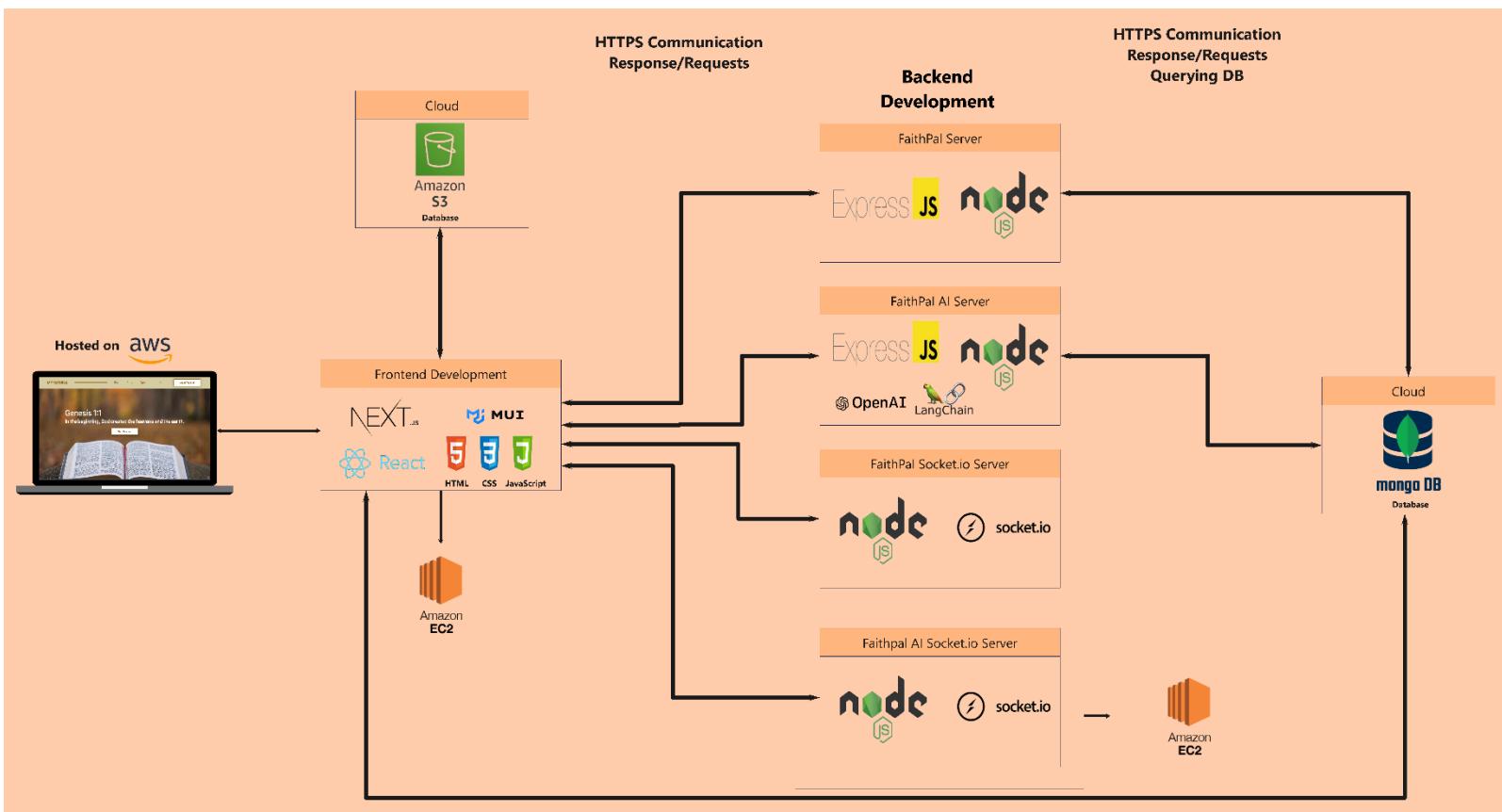
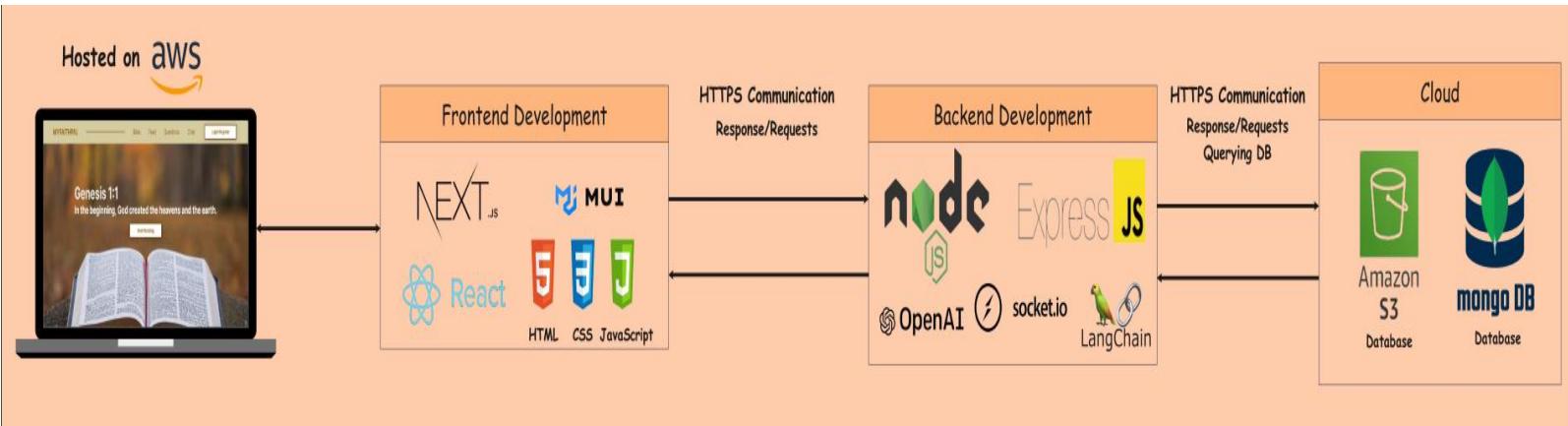


Figure 3 Architecture Diagram

6 Project Planning and Version control

Project management played a pivotal role in ensuring the success delivery of Faithpal. Through effective planning I was able to achieve key milestone and deliverables within the specified timelines.

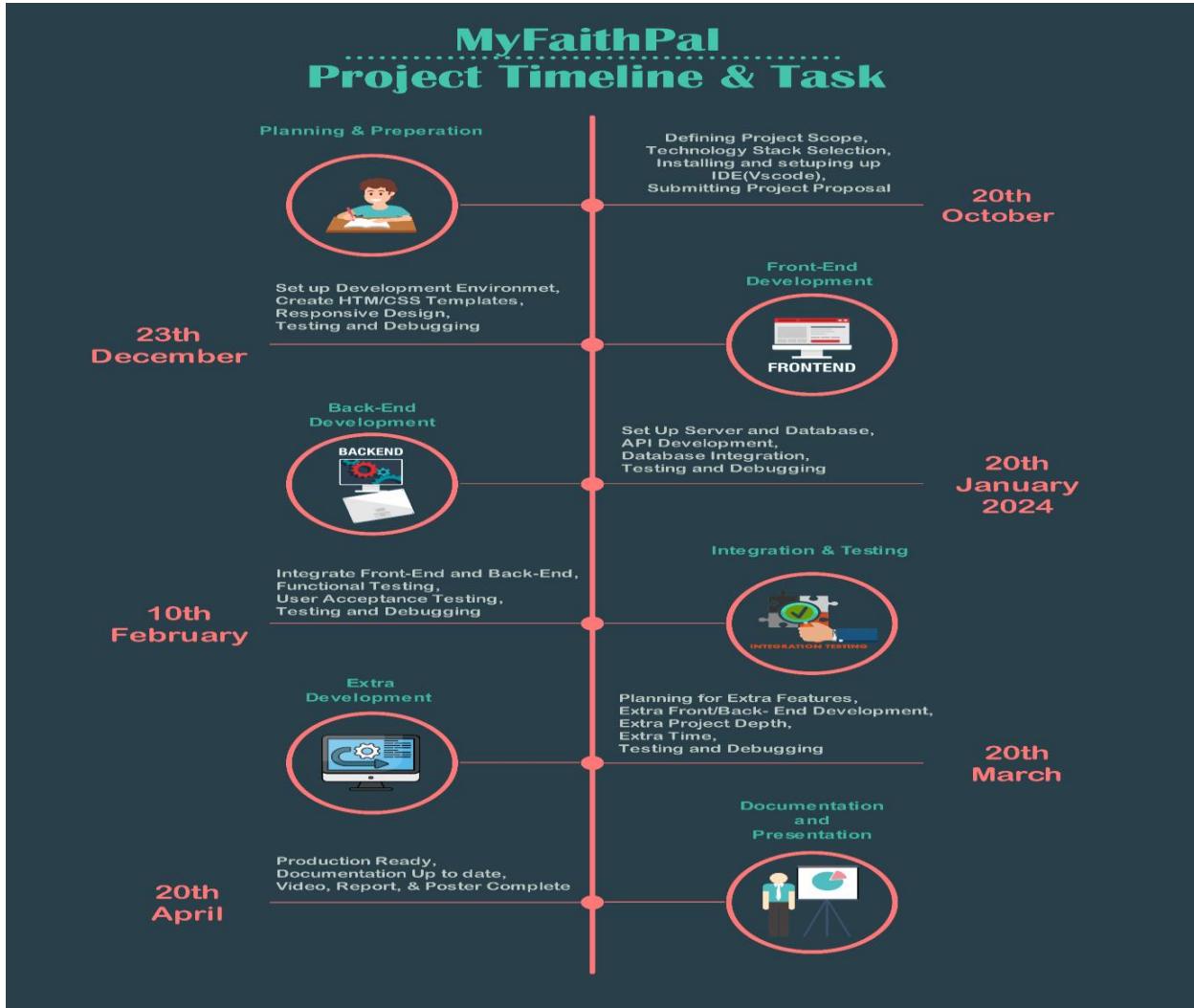
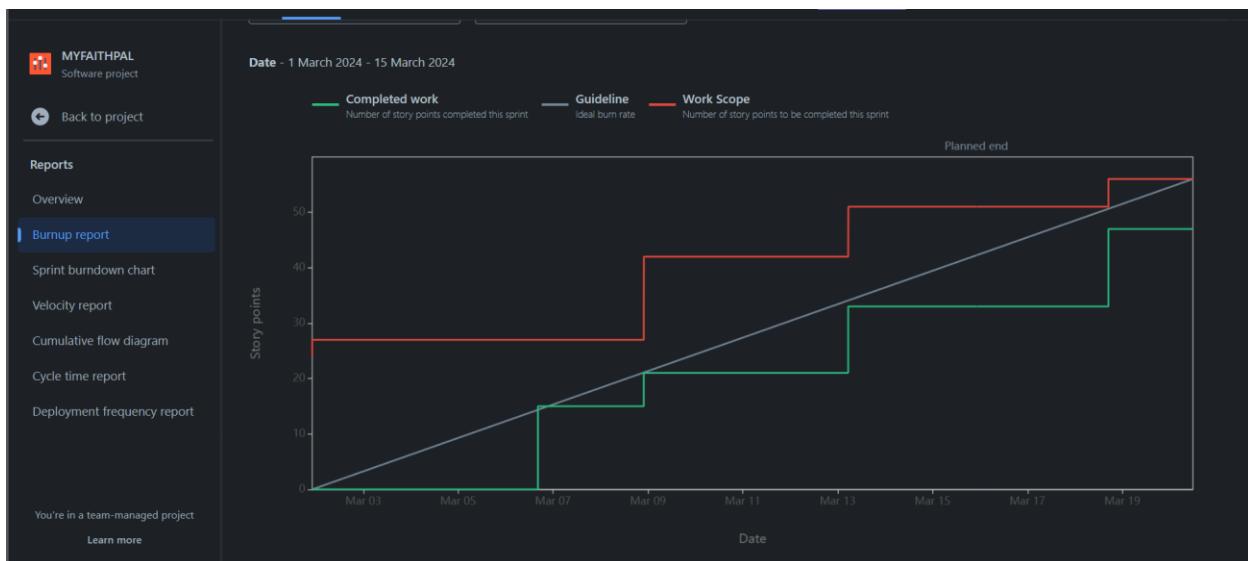


Figure 4 Architecture Diagram

As a result of successful project planning, I was able to finish my project by the 10th of April, leaving weeks to spare to prepare this Report, Videos and Poster. This was done by taking an Agile approach, which provides a structured framework that was really needed in launching and producing a web application that is feature rich requiring heavy tracking for a solo developer trying to be effective and disciplined. This is done by breaking the project down to more manageable segments called a “Sprint” that I chose to be a 2-week period that contains tasks

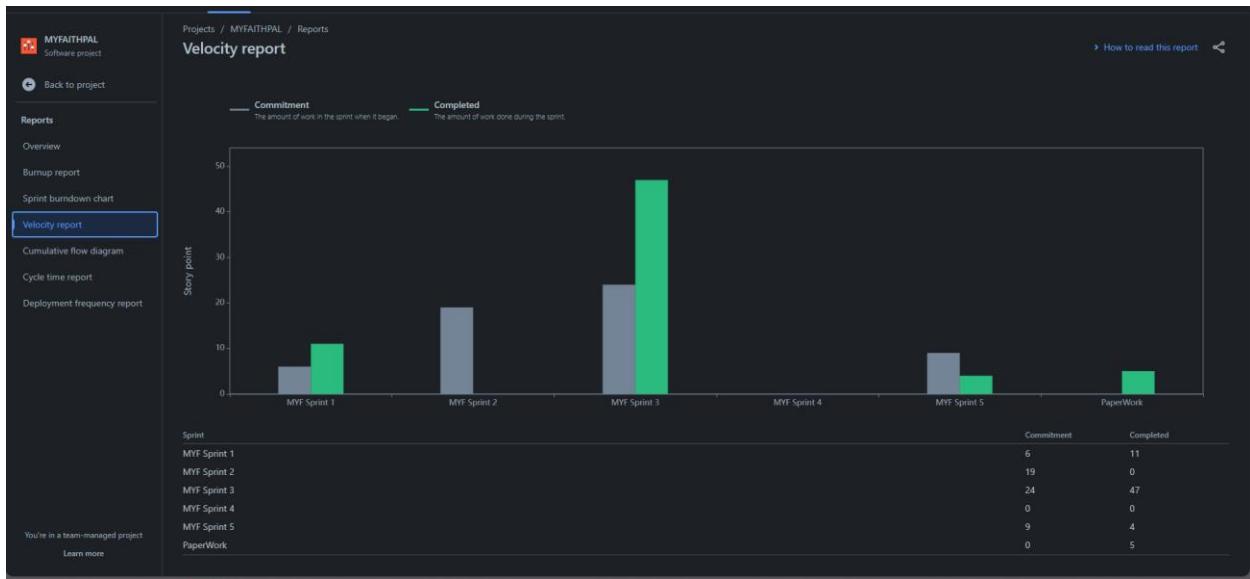
called Stories that will be the primary focus during a Sprint. In the Jira [4] Agile Scrum methodology, we have “Backlogs” this were before a Sprint or during we accumulate stories that would need to be done in the next Sprint. Each story was given points based on the difficulty and efforts of the task ranging from 0 – 10 and giving a description. Following this, when we finish a Sprint and about to create a new one, we look at the backlog and we take the stories that we believe we can complete during that two weeks accounting for the points to make sure I didn’t take on too many tasks that was too difficult to complete. If any stories weren’t completed within a sprint, then it will fall over to the next and I would do a personal review trying to assess if I was first undervaluing stories by points, then if I were also taken on too many tasks for a sprint.

In the examples below we have a Burnup Report



Burnup Report

This is a Burnup Report that can be generated in Jira [4]. With this chart we can visually represent the number of stories by their points was completed. We can see we had over 50 points worth of stories assigned but only managed to complete around 45 points meaning that I should probably lower the number of stories or the difficulty of stories that I assign to myself. The grey line is the ideal rate at which we are to complete tasks.



Velocity Report

This is a Velocity Report that can be generated in Jira [4]. With this chart we can see the number of stories by their points that were created at the beginning of the sprint and how many were completed at the end. In this report we can determine how much work was added during the sprint and how much was completed compared to what stories were created at the start of the sprint.

6.1.1 Version Control

This branch is 3 commits ahead of [main](#).

Commits

- Fix for Route being cached hard (2453bfb - 2 weeks ago) 202 Commits
- my-faithpal (Fix for Route being cached hard) 2 weeks ago
- my-faithpal_ai_server (New initial prompt) 2 weeks ago
- my-faithpal_server (Code cleaned up and ready to deploy) 3 weeks ago
- .hintrc (Added Translations and also css for note taking packages) 2 months ago
- .prettierignore (Footer added) 5 months ago
- README.md (Create README.md) 3 weeks ago
- fixdev.sh (Script to update servers) last month
- installdev.sh (Script to update servers) last month
- linrundev.sh (Spelling) 3 weeks ago
- rundev.sh (Script to update servers) last month

About

No description, website, or topics provided.

[Readme](#)

[Activity](#)

0 stars

1 watching

0 forks

Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

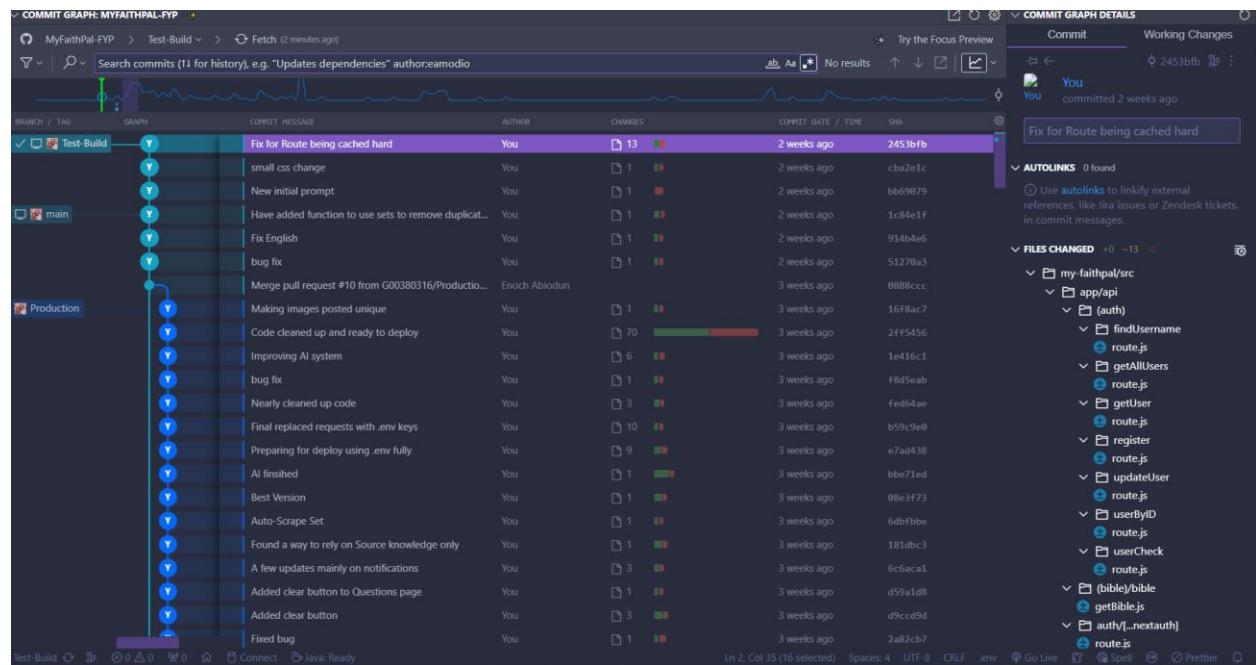
Languages

JavaScript 95.7% CSS 3.8% Shell 0.5%

GitHub Repository

I am using Git as version control and GitHub as a remote repository that stores my code.

Version control is an essential part of my project allowing me to work on different machines without losing any progress and keeping track of my progress. Committing any time there was a major change to my code.



Last few commits of FaithPal GitHub Repository

7 Front-End Development

In this section I will be referring to the technologies and the technicalities involved in coding the Front-End the part of code the End-User sees and interacts with in Faithpal.

The thought process concerning the Front-End of Faithpal was to make it as user-friendly as possible, so it won't hinder the effectiveness of the various features that the application entails.

7.1 React JS

React JS [5] that is often known as React [5], is an open-source JavaScript library used for building solid user interfaces, particularly for applications that typically have the same theme throughout the application as it enables features such as importing component modules to be useful. Developed and maintained by Facebook [3], React [5] has gained notability in the web development community due to its efficiency, simplicity, and declarative approach to building interactive UI components which made it easy for me to choose it as my first JavaScript library and use it as my introduction to JavaScript.

React [5] promotes a component-based architecture which allows developers to break down complex UIs into smaller, reusable pieces called components. These components don't only focus on the UI elements but also involve the logic behind the functionality of the component making the code more maintainable and modular. Referring to what I was saying above with keep the same them throughout an application, react's virtual DOM(Document Object Model) achieves this by only updating the necessary parts of the DOM when changing pages, resulting in improved performance and a smoother user experience.

One of React's key features that testifies to its modular nature as mentioned above would be the use of JSX (JavaScript XML). JSX is an extension of JavaScript that enables developers to write HTML-like code within JavaScript files. This makes it easier to visualise and construct UI components, blurring the lines between markup and logic.

React [5] is also favourable as it has an ecosystem that allows a vast number of third-party libraries and tools, such as React Router for routing and Material-UI for pre-designed UI components which are all at work in Faithpal. With all this in consideration its third-party libraries,

tools, community, and extensive documentation, made React [5] my number one choice in being a modern and interactive web app such as Faithpal.

7.2 Next JS

Next JS [6] is an open-source JavaScript framework built on top of React [5], it is designed to build scalable and high-performance web applications. Developed and maintained by Vercel, Next JS [6] takes React [5] a step further by offering server-side rendering known as SSR, static site generation, API routes, and much more that makes this framework more efficient than others in terms of development process and have considerable improved performance.

Next JS [6] takes things a step further past React's component-based architecture, by enabling server-side rendering, which can significantly improve the performance and SEO(Search Engine Optimisation) of web applications. Next JS [6] uses SSR to pre-renders page on the server and delivers them to the client, resulting in faster initial load times and better search engine indexing.

Next JS [6] supports static generation that allows developers to pre-render entire pages at build time and delivers them as static files with instant loading and reduced server load. The effects of this are seen when the final production of a web application is finished and “npm run build” to generate the static files and then “npm run start” to run the app. This is beneficial for pages that the content doesn't change much.

Another key feature and probably my personal favourite is its file-based routing system, which simplifies the creation of dynamic routes and enables the use of API routes for serverless functionalities limiting the need for a backend server such as Node JS [7], but I prefer to still have one as API routes are still a new technology and haven't really become standardised. But this makes it easy to build Restful APIs within a Next JS [6] application without a dedicated backend.

One of the key uses of Next JS [6] is in the Authentication of Users for my web application. Next JS [6] integrates seamlessly with other tools and services like Content Delivery Networks, headless CMS but the one I really used in my project was authentication providers. This allowed me to enable users to not only use NEXTAUTH credentials but also Google [29] and Facebook [3] sign in to allow users to sign up and sign in with their information from these social networks.

7.3 CSS

At the beginning of the setup of my Next JS [6] project I opted out of using Tailwind CCS which is quite popular currently as I wanted to get a solid foundation of CSS as I didn't have experience before this Project in CSS. Furthermore, Next JS [6] integrates smoothly with standard CSS, offering multiple ways to incorporate styles into their React-based applications. Each Next JS [6] app router will have a global.css file that doesn't have to be imported and that is used to define base styles that apply across the entire application. For modularity and to avoid style conflicts, Next JS [6] supports CSS Modules, where each component has its own scoped styles, ensuring unique class names. Additionally, with Next JS [6], you can use CSS-in-JS libraries like Styled Components or Emotion, which let you write CSS directly in JavaScript files. This approach, using template literals, allows for dynamic and conditional styling based on a component's state or properties. It integrates seamlessly with Next JS [6] server-side rendering, providing consistent styling across the application. This flexibility makes it a robust choice for incorporating CSS into modern web projects.

7.4 MUI

Material UI (MUI) [9] is a popular React [5] library that provides a set of ready to use components centered around on Google's material design principles. It makes it easy to create modern and consistent user interfaces without building everything from scratch. MUI [9] contains a lot of components such as buttons, forms, and navigation elements that follow Material Design guidelines and give your app a clean and unified look. The components are customisable, so you can set themes and colors to suit your design preferences and can use normal CSS on them by using the sx props which is equivalent to the style prop with normal HTML components allowing to tweak the look of components to my specific need. Because MUI [9] integrates seamlessly with React [5], you can combine its components with other React [5] tools and libraries, making it a flexible way to build user-friendly web applications. It also promotes accessibility and responsiveness, ensuring that your app works well across devices and is accessible to all users with no extra effort.

Overall, the main reason I chose to use this library to speed up development, with the use of just CSS I felt really bogged down with the number of extra lines of code that I had to write just only to write code for logic afterwards. In addition, the amount of time spent really did for me account for the quality of the styling, so I decided to kill two birds with one stone, and it paid off. Also, one of the main reasons I chose MUI [9] is because most of these component's library require tailwind CSS such as NEXTUI, so MUI [9] was a perfect option being built on pure CSS.

8 Back-End Development

In this section I will be referring to the technologies and the technicalities involved in coding the Back end the part of code the End-User doesn't see and works in the background in response to the interactions made by users on Faithpal.

8.1 Node JS

Node JS [7] is an open source, cross-platform JavaScript runtime environment that allows developers to execute JavaScript not only on the client side but also on the server side. It uses Google's V8 JavaScript code to execute code efficiently, which gives it the ability to handle not just one but multiple tasks simultaneously without blocking the main process. This feature makes Node JS [7] well suited for building scalable and high-performance back-end servers.

Although we are focusing on Node JS [7] as just our backend technology in this section I believe it is important to note the resourcefulness and the power of Node JS [7]. It is typically used for server-side development but also plays a crucial role in front-end development process as well. By using Node JS [7], developers can set up local development environments, automate build processes, and manage project dependencies. Tools like Webpack and babel, are key to modern front-end workflows that rely on Node JS [7] to compile and bundle code. Additionally, popular frameworks such as Next JS [6], which combine server-side rendering with front-end development, require Node JS [7] as their underlying runtime environment.

Node JS [7] event-driven, non-blocking architecture allows for efficient real-time communication, making it ideal for building features like live updates and interactive elements that are common in user-friendly features that are found in Faithpal. Node JS [7] support for Socket.io [11] (Alternative to WebSocket's) for example, makes it easier to implement real-time chat applications, notifications, and other interactive experiences.

In summary, Node JS [7] is a versatile technology that can support both back-end and front-end development. For front-end projects, it provides the infrastructure for building, testing, and deploying code, managing dependencies, and enabling real-time features. This makes it a valuable tool for creating fast, responsive, and interactive web applications like Faithpal.

8.2 Express JS

Express JS [8] is a popular, lightweight framework that complements Node JS [7] making it easier to create web servers and RESTful APIs. It is really known for being straightforward and flexible bringing the focus for developers on logic and less set up allowing the production of server-side applications with minimal fuss.

When building web applications, I believe that Express JS [8] is like the foundation that holds everything together. It allows you to set up routes for handling HTTP requests and responses, add middleware to process incoming data, and serve static files like HTML, CSS, and JavaScript if you want a little front-end for your server application.

Overall, Express JS [8] is a solid choice for building server-side applications in Node JS [7]. Its simplicity, flexibility, and support for middleware and routing make it ideal for building the back-end services that modern front-end applications like Faithpal rely on. With Express JS [8], you get the tools you need to create a stable back end that makes sure that your front-end and back-end work together well.

8.3 MongoDB

MongoDB [17] is a NoSQL database that works well with Node JS [7] for building a solid back end in web applications. Not like SQL databases, which utilise tables and predefined mappings, MongoDB [17] permits you to store information in a more dynamic, document-oriented organize. This approach makes it simpler to adjust to changing necessities and oversee complex information structures without stressing approximately strict pattern definitions. Its employments an adaptable JSON-like arrange called BSON (Parallel JSON), which permits you to store diverse sorts of information inside the same collection.

Working with MongoDB [17] and Node JS [7] permits for quick and productive access to stored data. The asynchronous nature of Node JS [7] complements MongoDB's scalability, making it suitable for high-traffic applications. This combination is ideal for building back-end server that can handle huge sums of information preparing such as the source information from GotQuestions.org that I utilize for AI in Faithpal.

Furthermore, MongoDB [17] gives extra tools in relation to storing and retrieving data, am example of this could be vector index searching [16]. But before this is done MongoDB [17] has an embeddings method in its LangChain [13] MongoDB [17] module that allows for data to be stored in 1 and 0s enabling the AI to use data from the database in its answers. This adaptability is profitable when coding back-end rationale for web applications, as you can quickly fetch the information you wish and construct complex questions without broad SQL information which I was missing when going into fourth year.

Another advantage of utilizing MongoDB [17] with Node JS [7] is the ease of integration. The well-known library Mongoose gives an interface for working with MongoDB [17] in Node JS [7], permitting you to characterize models and oversee information connections without complex setup. This integration makes it direct to associate your Node JS [7] back-end to MongoDB [17] and begin interacting along with your database right absent.

To round up, MongoDB [17] with Node JS [7] is a capable combination for building back-end services in web applications.

8.4 Socket.io

Socket.io [11] is a well-known library that lets you set up real-time communication between servers and clients in Node JS [7] applications. With Socket.io [11], you'll be able make associations where information can stream back and forward immediately, without the required for page reloads. This is good for building features like live chat and notifications on sites like FatihPal.

One of the main reasons developers use Socket.io [11] with Node JS [7] is its ability to handle multiple tasks simultaneously. Since Node JS [7] is known to be very efficient at multitasking and event operation, it can monitor the collaboration of multiple servers in real time without slowing down.

In the FaithPal we require a backend that allows for real-time communication between client and server for chat feature. For this case, we use it to make a situation where client can send and get messages right away from the server. Socket.io [11] lets you construct these features by setting up a persistent association between the server and clients, allowing data to be sent and gotten immediately.

Socket.io [11] too takes care of common issues like cross-browser compatibility and connection stability. It employs WebSocket for low-latency communication but can switch to other strategies on the off chance that WebSocket aren't accessible, guaranteeing a solid association in any case of the user's environment. This property makes it less demanding to build backends that work easily for all clients.

Finally, Socket.io [11] and Node JS [7] are a powerful combination for building real-time capabilities for web applications. It provides an easy way to build intelligent backends that handle real-time chat or notification logic without almost any connectivity issues or performance bottlenecks. Using Socket.io [11] in your Node JS [7] backend, you can react [5] add a real-time responsive component to your website.

8.5 LangChain

LangChain [13] could be a framework outlined to streamline the building of applications that work with expansive language models, like those from OpenAI [14], inside a Node JS [7] environment. It gives devices and reflections to assist developers in the complexities of integrating advanced AI models into their applications. Working with OpenAI [14] models, LangChain [13] empowers you to form organised workflows and utilise AI demonstrate control in a more brief and user-friendly way.

Node JS [7], a common runtime environment for making server-side applications, fits well with LangChain [13], giving you the capacity to oversee OpenAI [14] models. With LangChain [13], you'll create applications that use these models to process inputs, create content, or associate with clients in a conversational way like done in FaithPal. Typically, it is especially valuable when you're building back-end features that must react [5] to complex questions and create answers based on user input.

Using LangChain [13] with OpenAI [14] in a Node JS [7] application offers several benefits. To begin with, it gives a higher-level structure for interfacing AI models, making it less demanding to characterise how to utilise and get to them in your backend. For illustration, you'll be able make primary chains where the yield of one step is the input of another step, permitting you to construct more complex and context-aware cleverly OpenAI [14] models. Usually valuable for building chatbots, virtual colleagues, or other AI-based features that require step-by-step preparation.

In expansion, LangChain [13] coordinates with other information sources, causing information to be retrieved and used from databases or APIs, which is one of the features Faithpal employments in an AI called "Solomon". This will be useful when your back end must drag in outside information or work with existing data sources to create reactions or perform tasks.

Overall, joining LangChain [13] with OpenAI [14] and Node JS [7] may be a great choice for making a modern back end for your web application. It permits you to construct complex AI-powered features whilst keeping up a clear structure and workflow, making it easier to develop and keep up AI-based functionalities in your back-end code.

8.6 Cheerio

Cheerio [25] is a Node JS [7] library that makes it easy to work with HTML and XML content on the back end. If you're familiar with jQuery, Cheerio [25] will feel very similar because it uses a comparable syntax and methods to manipulate and traverse HTML structures. This makes Cheerio [25] a valuable tool for building web applications where you need to scrape web content, process HTML data, or manipulate XML files on the server side.

Using Cheerio [25] with Node JS [7] provides a convenient way to extract information from websites or other HTML-based sources. It's especially useful for web scraping tasks, where you want to gather data from existing web pages. With Cheerio [25], you can load an HTML document into a virtual DOM and use a jQuery-like interface to select and manipulate elements. This capability makes it easier to parse specific data, like product prices, article titles, or meta tags, without dealing with complex HTML parsing logic.

Node JS [7] is a great environment for using Cheerio [25] because of its asynchronous and event-driven nature. This makes it ideal for processing large volumes of data or handling multiple web scraping tasks simultaneously. If you're building a back end that requires extracting information from different sources, Cheerio [25] with Node JS [7] provides a powerful and efficient solution.

One common use case for Cheerio [25] with Node JS [7] is in building applications that rely on third-party data sources. For example, you might need to scrape a website to get product information, event details, or other content for your web application. Cheerio [25] allows you to do this in a structured way, letting you extract the data you need and integrate it into your application logic. You can also use Cheerio [25] to manipulate HTML content before sending it to the front end, which can be useful for templating or dynamic content generation.

Another advantage of using Cheerio [25] is its lightweight design. It doesn't require a browser environment to function, so it's fast and efficient, even when processing large HTML documents. This makes it a good choice for building back-end services that need to work with HTML or XML data without the overhead of a full browser engine.

In summary, Cheerio [25] with Node JS [7] is an excellent choice for building back-end services that need to interact with HTML or XML content. It offers a simple and efficient way to scrape, manipulate, and process web data, making it ideal for web scraping tasks or applications that require dynamic content generation. With its jQuery-like syntax and fast performance, Cheerio [25] helps you build robust and flexible back-end logic for your web application.

9 Cloud Development

In this section I will be referring to the technologies and the technicalities involved in handling data on the cloud and deploying Faithpal with its five back-end servers and one front-end to the Internet making it accessible to users around the world.

9.1 Amazon Web Services – S3 Bucket

Amazon Web Services (AWS) S3 [19] may be a cloud solution to assist you oversee and store large sums of information, such as photo in a productive way. In a web application, utilizing S3 [19] with Next JS [6] gives a great approach to store and recover picture records whereas guaranteeing clear integration along with your front-end and back-end.

One of the main reasons to use S3 [19] with Next JS [6] for image storage is flexibility. S3 [19] can handle large amounts of data without compromising performance. This is particularly important when your web application needs to contain a unique set of images, such as user profile pictures, cover images like Faithpal handles. With S3 [19], you do not get to stretch around running out of capacity or directing physical hardware, AWS handles all of that for you, promoting scalability .

Using S3 [19] for picture capacity rather than MongoDB [17] offers a few clear focal points. Whereas MongoDB [17] is awesome for organized information and complex inquiries, S3 [19] is particularly planned for putting away and recovering huge data, like pictures, recordings, or documents. This makes S3 [19] more productive for putting away picture records, because it is optimized for high-volume information capacity and retrieval. Additionally, S3 [19] gives features like versioning, lifecycle approaches, and information replication, which can assist you oversee your picture data more successfully.

At last, AWS S3 [19] with Next JS [6] can be a viable combination for putting absent and managing pictures in a web application. It offers flexibility and openness making it an ideal choice for applications that need to be handle expansive volumes of picture data. By utilizing S3 [19] instead of MongoDB [17] for picture capacity, you will be able make a more beneficial and adaptable framework for your web application, ensuring that it can create and alter to future needs without giving up execution or immovable quality.

9.2 Amazon Web Services – EC2 Instances

AWS Adaptable Compute Cloud (EC2) [20] is a cloud-based solution that allows you to use customisable virtual servers. You can use virtual machines to store and run your applications, giving you flexibility in resources, configuration, and scaling. On the off chance that you've got a Next JS [6] web client application with two Express JS [8] backends and two Socket.io [11] servers (such as FaithPal), at that point EC2 [20] can be a great choice for taking care of these numerous servers in a competent environment.

One of the key reasons to utilise EC2 [20] for facilitating your Next JS [6] project, Express [8] backends, and Socket.io [11] servers is adaptability. With EC2 [20], you'll begin with a small occurrence to keep costs down, at that point update to more effective machines as your application's request develops. Usually particularly valuable when your application has shifting workloads otherwise you anticipate client activity to extend over time. With EC2 [20], you'll include or evacuate assets as required, guaranteeing your framework can handle both typical operations and spikes in movement.

Another advantage of utilising EC2 [20] is the level of customization it offers. You can choose from a variety of instance types, each with different CPU, memory, and capacity levels. This customisation feature allows you to optimise your server based on your application needs.

EC2 [20] for hosting servers like Next JS [6], Express [8] backends, and Socket.io [11] servers too offer a great level of control. You'll be able select the working framework, set up security groups, and define client consents to guarantee your servers are secure and open only to those who require them. This level of control is basic when you're overseeing numerous servers and got to keep up a secure and dependable environment.

In summary, AWS EC2 [20] is a versatile and flexible tool for hosting a collection of server-side components, including Next JS [6] project, Express [8] backend, and Socket.io [11] server. It provides a customisable environment where you can modify dependencies as needed and maintain strict security controls. With EC2 [20], you can create a strong foundation for web applications that meet your needs, operate smoothly, and scale with usage over time.

9.3 Porkbun

Porkbun [22] is a domain name registrar known for its straightforward approach to buying, registering, and managing domain names. If you're looking to purchase domain names for your servers, Porkbun [22] offers a user-friendly interface and competitive pricing, making the process simple and cost-effective. One of the reasons developers and businesses choose Porkbun [22] is its transparent pricing , with no hidden fees or unexpected renewal costs. This transparency is crucial when you're managing multiple domain names and need to budget accurately this is why I chose it for FaithPal.

9.4 Nginx reverse proxy and SSL

Nginx is a good choice for web servers because it's fast and dynamic. When you use it as a reverse proxy, Nginx [23] can direct traffic from port 80, the standard HTTP port, to different back-end servers. This setup is perfect for web apps with multiple components, such as a Next JS [6] front-end, Express [8] back-ends, or Socket.io [11] servers, which might be running on different ports or even different machines. It simplifies the process of routing traffic, letting you centralise your network configurations and manage everything in one place.

Using Let's Encrypt to add SSL (Secure Sockets Layer) is a smart way to make your connections secure. SSL encrypts the data traveling between your servers and users, keeping sensitive information safe from prying eyes or attackers. Let's Encrypt makes this easy by offering free SSL certificates, which means you can secure your app without additional costs. With SSL in place, users can trust your site more, knowing that their personal information is protected.

In summary, Nginx [23] with a reverse proxy on port 80, combined with SSL using Let's Encrypt, is a practical solution for managing web traffic in a multi-server setup. It streamlines the process of handling incoming requests, while SSL ensures that your connections are secure. This setup is ideal for modern web applications that need to manage different back-end services and maintain strong security for user data.

10 Programming Process

In this section, I will be going thoroughly through my code step by step to explain the inner workings of Faithpal from the front end that the user interacts with to the secretive back end that the user doesn't see the logic behind but the results of its processes. Each major functionalities will be broken down and explained how they work to have a look at the code yourself this is a link to my GitHub ([G00380316/MyFaithPal-FYP \(github.com\)](https://github.com/G00380316/MyFaithPal-FYP)).

10.1 Layout (Next JS App Router)

In my Next JS [6] project which is Next 13.5.6, I am using the term app router that uses file-based routing, where folders represent routers, and files define the content for those routes.

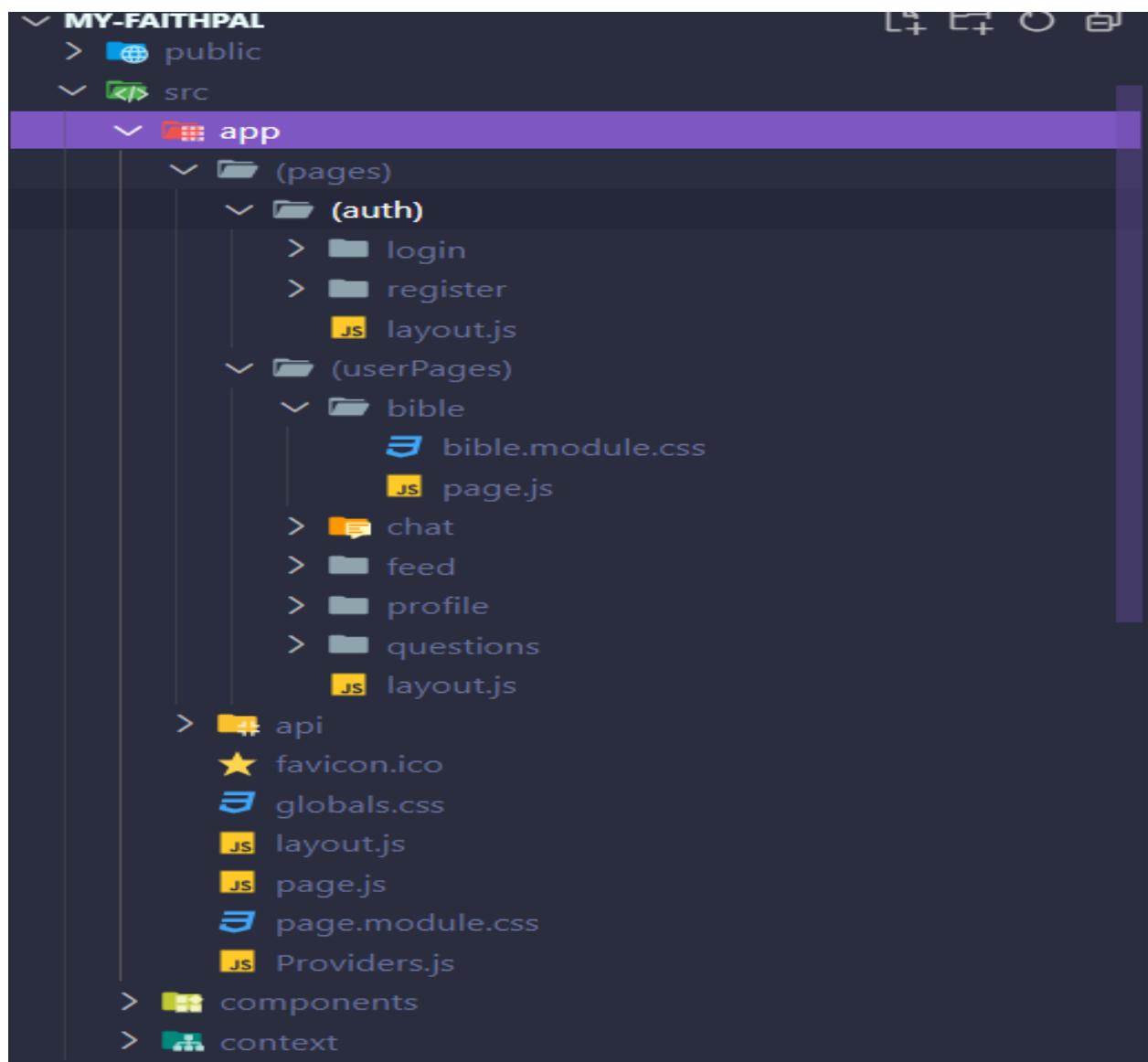


Figure 4

In Figure 4, you can see we have the app directory for defining routes which in previous versions of Next JS [6] would be named pages. In this segment we have brackets around some folders hidden them from showing in the routing of the page so if we wanted to get to the bible page of Faithpal in development mode it would be **localhost:3000/bible** with **page.js** in the bible defining the contents of the bible page.

In an App router setup, we have the **layout.js** within folders that provides the layout for all nested routes. It is very possible to have multiple layout.js files since FaithPal doesn't carry out the same theme on some pages this it is necessary to have more than one layout file to allow for this change. Below are examples of different layout.js files within FaithPal.

The screenshot shows a code editor interface with three tabs at the top: `navbar.module.css`, `globals.css`, and `layout.js`. The `layout.js` tab is active, displaying the following code:

```
1 import { Inter } from "next/font/google";
2 import { Providers } from "./Providers";
3 import "./globals.css";
4 import { ToastContainer } from "react-toastify";
5 import 'react-toastify/dist/ReactToastify.css';
6
7 const inter = Inter({ subsets: ["Latin"] });
8
9 export const metadata = {
10   title: "FaithPal",
11   description: "Made by Enoch Abiodun",
12 };
13
14 export default function RootLayout({ children }) {
15   return (
16     <html lang="en" suppressHydrationWarning={true}>
17       <body className={inter.className}>
18         <Providers>
19           {children}
20         <ToastContainer newestOnTop limit={3} style={{width:450}}/>
21       </Providers>
22     </body>
23   );
24 }
25 }
```

Figure 5, This is the default Layout for the whole Application and is always in play.

```

1 //components
2 import Navbar from "@/components/navbar/navbar.js";
3 import Footer from "@/components/footer/footer";
4 import { Box } from "@mui/joyn";
5
6 export default function UserLayout({ children }) {
7   return (
8     <Box display="flex" flexDirection="column" width="100%" height="100vh">
9       <Navbar />
10      {children}
11    </Box>
12  );
13}
14

```

Figure 6, This is the Layout after authentication notes the Navbar component.

```

1 export default function AuthLayout({ children }) {
2   return (
3     <Box display="flex" flexDirection="column" width="100%" height="100vh">
4       {children}
5     </Box>
6   );
7 }

```

Figure 7, This is the Layout before authentication notes the Navbar component is missing.

To see the results of these layouts visit <https://myfaithpal.com>, in the log in and register I preferred not to have a navigation to the rest of the application making authentication the primary focus and only upon log in will they be redirected to the main application and the user layout but Root layout is always in effect.

10.2 Next Authentication (NEXTAUTH)

The method of Authentication in FaithPal is through a library called **next-auth** that enables session-based authentication. This stores user sessions on the server with their credentials stored in FaithPal MongoDB [17] database and allows users to remain authenticated as long as their session is valid (in FaithPal all sessions have an expiry time of a week). Next-auth primarily deals with **JWT** which is token based authentication upon log in users receive a token. The token is

used to authenticate future requests and used to store user details from authentication in sessions.

```

import { connectMongoDB } from "@/Lib/mongo";
import User from "@/models/user";
import bcrypt from "bcryptjs";
import NextAuth from "next-auth/next";
import CredentialsProvider from "next-auth/providers/credentials";
import FacebookProvider from "next-auth/providers/facebook";
import GoogleProvider from "next-auth/providers/google";

export const authOptions = {
  providers: [
    CredentialsProvider({
      name: "credentials",
      credentials: {},
      async authorize(credentials) {
        const { email, password } = credentials;
        try {
          await connectMongoDB();
          const user = await User.findOne({ email });
          if (!user) {
            return null;
          }
          const passwordCheck = await bcrypt.compare(password, user.password);
          if (!passwordCheck) {
            return null;
          }
          return user;
        } catch (error) {
          console.error(error);
        }
      },
    }),
  ],
};

```

Figure 7, nextauth route in API router in Next Project.

In Figure, we are in the API route this allows you to create serverless functions that handle HTTP requests. Each file in this directory represents specific endpoint for FaithPal API. To carry one the app router, theme the file structure determines the route path, and these endpoints can handle a various amount of HTTP methods such as GET,POST, PUT etc.

Initially a user must sign up we simply just create a user in our database with the user model declared in the model directory of our Next JS [6] project and we get the person to submit a form that hits the register API in the API route examples creating the user. Some examples of the written logic will be displayed below in figure 8 .

The screenshot shows a code editor with the following file structure:

```

OPEN EDITORS
src > app > api > (auth) > register > route.js > ...
  1 import { connectMongoDB } from "@/Lib/mongo";
  2 import User from "@/models/user";
  3 import bcrypt from "bcryptjs";
  4 import { NextResponse } from "next/server";
  5
  6 export async function POST(req) {
  7   try {
  8     const { username, name, email, password } = await req.json();
  9     const hashedPassword = await bcrypt.hash(password, 10);
 10
 11     //console.log("Name: ", name);
 12     //console.log("Email: ", email);
 13     //console.log("Password: ", password);
 14
 15     await connectMongoDB();
 16     await User.create({ username, name, email, password: hashedPassword });
 17
 18     return NextResponse.json({
 19       message: "User has been Registered"
 20     }, { status: 201 });
 21
 22   } catch (error) {
 23     return NextResponse.json({
 24       message: "An Error occurred while user was being Registered"
 25     }, { status: 500 });
 26   }
}

```

The code is a Next.js API route for user registration. It imports necessary modules, defines a POST handler, and performs validation and database operations. A call to `userCheck` is made before the main registration logic.

Figure 8, Post Request in API Route to create a User, `userCheck` endpoint is called before this.

The screenshot shows a code editor with the following client-side logic:

```

const handleSubmit = async (e) => {
  e.preventDefault();

  try {
    if (!email || !password) return;

    if (!name || !email || !password) {
      setError("All fields are necessary");
      return;
    }

    const resUserExists = await fetch('api/userCheck', {
      method: 'POST',
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ email }),
    });

    const { user } = await resUserExists.json();

    if (user) {
      setError("User already exists");
      form.reset();
      //console.log(error);
      return;
    }

    const res = await fetch('api/register', {
      method: "POST",
      headers: {
        "Content-Type": "application/json"
      },
      body: JSON.stringify({
        username, name, email, password
      }),
    });

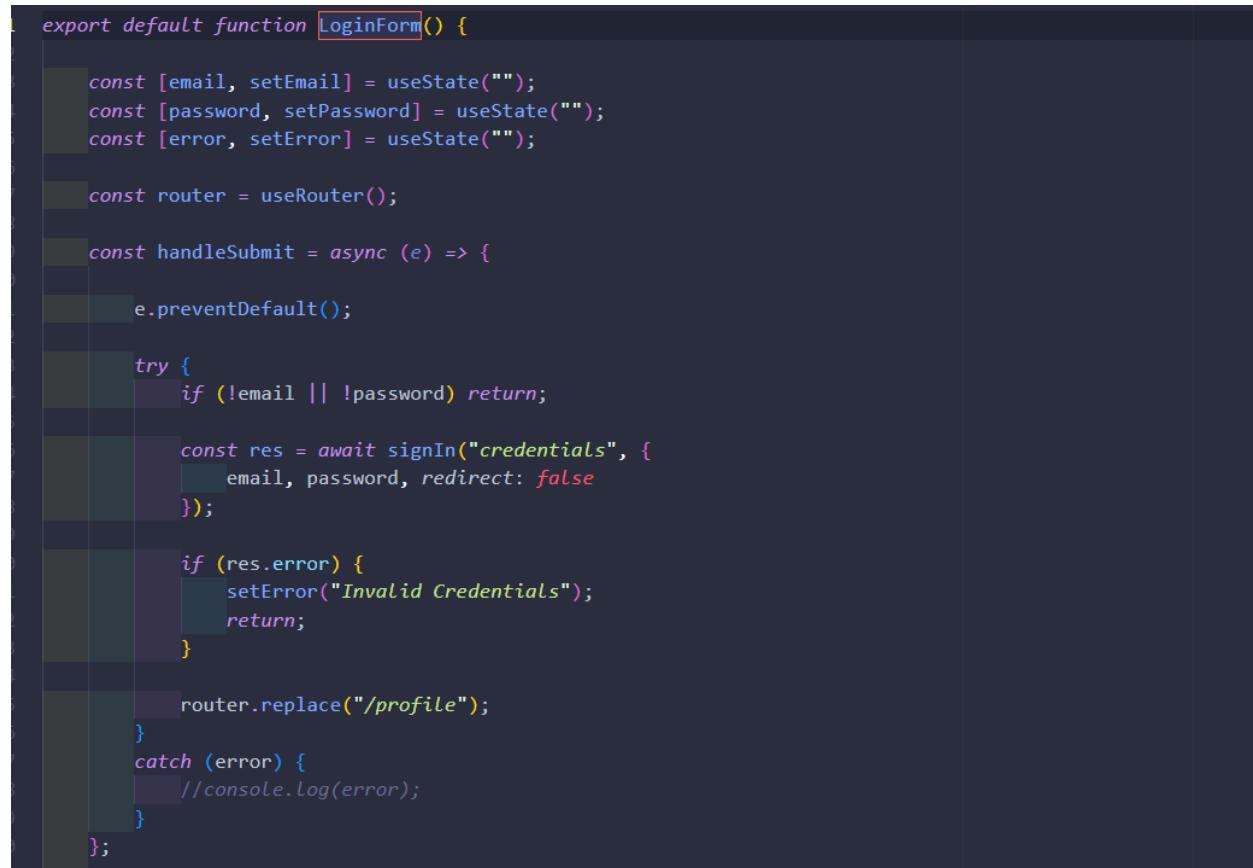
    if (res.ok) {
      const form = e.target;
      form.reset();
      router.push("/Login");
    }
  }
}

```

This logic handles a form submission, checks for existing users via the `userCheck` endpoint, and then attempts to register a new user via the `register` endpoint. If successful, it performs a redirect to the login page.

Figure 9, API route endpoints being called to create users.

Furthermore, in the file displayed in figure 7 we have the logic for Nextauth, and this is where the sign in is currently being handled as we **await User.findOne** to check if the user exists and then proceed to compare the password given to the one in the database this is all done with authentication provider called credentials as in seen in figure 7 that gets triggered upon the completion of the log in form on the log in page look at figure 10 to see how the requests is made to the nextauth route.



```

1 export default function LoginForm() {
2
3     const [email, setEmail] = useState("");
4     const [password, setPassword] = useState("");
5     const [error, setError] = useState("");
6
7     const router = useRouter();
8
9     const handleSubmit = async (e) => {
10
11         e.preventDefault();
12
13         try {
14             if (!email || !password) return;
15
16             const res = await signIn("credentials", {
17                 email, password, redirect: false
18             });
19
20             if (res.error) {
21                 setError("Invalid Credentials");
22                 return;
23             }
24
25             router.replace("/profile");
26         }
27         catch (error) {
28             //console.log(error);
29         }
30     };
31
32 }
```

Figure 10, Nextauth getting triggered at the submission of log in form.

Focusing on the Nextauth file we can see a continuation of the snippet of Figure 7 below.



```

export const authOptions = [
  {
    async authorize(credentials) {
      const user = await User.findOne({ email });

      if (!user) {
        return null;
      }

      const passwordCheck = await bcrypt.compare(password, user.password);

      if (!passwordCheck) {
        return null;
      }

      if (user.image) {
        return { _id: user.id, name: user.name, email: user.email, image: user.image, username: user.username };
      }

      return { _id: user.id, name: user.name, email: user.email, username: user.username };
    } catch (error) {
      //console.log("Error: ", error);
    }
  },
},
GoogleProvider({
  clientId: process.env.GOOGLE_CLIENT_ID ?? "",
  clientSecret: process.env.GOOGLE_CLIENT_SECRET ?? "",
}),
FacebookProvider({
  clientId: process.env.FACEBOOK_CLIENT_ID,
  clientSecret: process.env.FACEBOOK_CLIENT_SECRET
}),
],
session: {
  strategy: "jwt",
  maxAge: 7 * 24 * 60 * 60,
}
];

```

Figure 11, Additional Providers, and initialization of session.

Here we can see nextauth provides multiple means of authentication called Providers allowing users to easily sign and sign up through social media networks such as Google [29] and Facebook [3].

```

9  export const authOptions = {
10    callbacks: {
11      async jwt({ token, user, session }) {
12        const user = await User.findOne({ email: token.email });
13
14        if (user?._id) token._id = user._id;
15
16        if (user?.username) token.username = user.username;
17
18        if (user?.image) token.image = user.image;
19        //console.log("jwt callback", { token, user, session })
20
21      }
22
23      //pass id to token
24      if (user?._id) token._id = user._id;
25      //passing in username to token
26      if (user?.username) token.username = user.username;
27      //passing in image to token
28      if (user?.image) token.image = user.image;
29      //console.log("jwt callback", { token, user, session })
30
31      return token;
32    },
33    async session({ session, token, user }) {
34
35      //then putting the _id in session from database
36      if (token?._id) session.user._id = token._id;
37      //then putting the username in session from database
38      if (token?.username) session.user.username = token.username;
39      //then putting the username in session from database
40      if (token?.image) session.user.image = token.image;
41      //console.log("session callback", { token, user, session })
42
43      return session;
44    },
45    async signIn({ profile, credentials }) {
46      //console.log("Details:", profile)
47
48    }
49  }

```

Figure 11, JWT and Session callbacks.

In Faithpal we use callbacks as hooks for authentication events letting us have a fine control over the authentication process. We also use this callback to modify data stored in the JWT token and the session token as you can see in figure 11, I am passing information from the token then to the session token so I can use the data within my Next JS [6] project.

```

if (profile) {
  try {
    await connectMongoDB();

    const userExist = await User.findOne({ email: profile.email });

    if (!userExist) {
      if (profile?.iss === 'https://accounts.google.com') {

        const username = profile.name.replace(/\s+/g, '-');

        const user = await User.create({
          email: profile.email, name: profile.name, image: profile.picture, username
        })

        //console.log("Created Google user", user)

      } else {

        const username = profile.name.replace(/\s+/g, '-');

        const user = await User.create({
          email: profile.email, name: profile.name, image: profile.picture?.data?.url, username
        })

        //console.log("Created Facebook user", user)

      }
    } else if (userExist) {
      if (profile?.iss === 'https://accounts.google.com') {
        ...
      }
    }
  }
}

```

Figure 11

```

    }

} else if (userExist) {
  if (profile?.iss === 'https://accounts.google.com') {
    if (userExist.image) {

      const updatedUser = await User.findOneAndUpdate({ email: profile.email }, { email: profile.email, name: profile.name }, { new: true });
      //console.log("Updated Google user", updatedUser)

      return true
    } else {
      const updatedUser = await User.findOneAndUpdate({ email: profile.email }, { email: profile.email, name: profile.name, image: profile.picture }, { new: true });
      //console.log("Updated Google user", updatedUser)

      return true
    }
  }
}

```

Figure 12

In these snippets we have examples of logic of the sign in callback. These logics handle the creation and sign in of User data in MongoDB [17] if they were to use Google [29] or Facebook [3] to sign in.

10.3 Bible Page

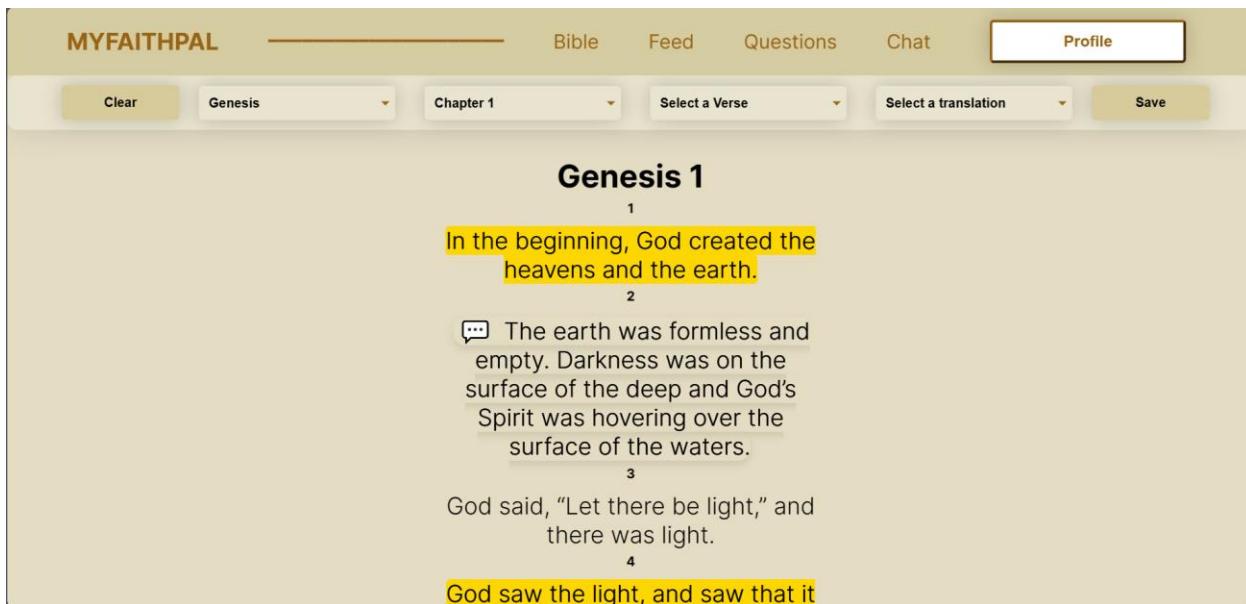


Figure 13, Bible page.

For the Bible I am using lots of different React [5] techniques, functions, and methods to display the bible. The bible page consists of two components the bible the dropdown menu and the Display Passage component.

```
export default function Bible() {
  const [selectedBook, setSelectedBook] = useState("Genesis");
  const [selectedChapter, setSelectedChapter] = useState(1);
  const [selectedVerse, setSelectedVerse] = useState(1);
  const [selectedTranslation, setSelectedTranslation] = useState("NIV");
  const [isSaveClicked, setSaveClicked] = useState(false);
  const [isClearClicked, setClearClicked] = useState(false);

  const handleSelectionChange = (book) => {
    setSelectedBook(book);
  };

  const handleSaveButtonClick = () => {
    setSaveClicked(true);
  };

  const handleClearButtonClick = () => {
    setClearClicked(true);
  };

  return (
    <Grid container direction="column">
      <Stack spacing={1} sx={Styles.root}>
        <div className={styles.nav}>
          <Dropdown
            onSelectionChange={handleSelectionChange}
            onSaveClick={handleSaveButtonClick}
            onClearClick={handleClearButtonClick} />
        </div>
        <DisplayPassage
          selectedBook={selectedBook}
          selectedChapter={selectedChapter}
          selectedVerse={selectedVerse}
          selectedTranslation={selectedTranslation}
          saveClicked={isSaveClicked}
          clearClicked={isClearClicked} />
      </Stack>
    </Grid>
  );
}
```

Figure 14, Html Markdown for the Bible page with some MUI [9] components (Stack, Grid).

10.3.1 Dropdown

For the Dropdown components I use the map functions to collect the information to populate each dropdown options snippets of the logic are displayed below

```
export const bookInfo = {
  'Genesis': [
    { chapter: 1, verses: 31 },
    { chapter: 2, verses: 25 },
    { chapter: 3, verses: 24 },
    { chapter: 4, verses: 26 },
    { chapter: 5, verses: 32 },
    { chapter: 6, verses: 22 },
    { chapter: 7, verses: 24 },
    { chapter: 8, verses: 22 },
    { chapter: 9, verses: 29 },
    { chapter: 10, verses: 32 },
    { chapter: 11, verses: 32 },
    { chapter: 12, verses: 20 },
    { chapter: 13, verses: 18 },
    { chapter: 14, verses: 24 },
    { chapter: 15, verses: 21 },
    { chapter: 16, verses: 16 },
    { chapter: 17, verses: 27 },
    { chapter: 18, verses: 33 },
    { chapter: 19, verses: 38 },
    { chapter: 20, verses: 18 },
    { chapter: 21, verses: 34 },
    { chapter: 22, verses: 24 },
    { chapter: 23, verses: 20 },
    { chapter: 24, verses: 67 }]
```

Figure 15, Json Object of Books that is filtered into Chapters and Verses to be used as options.

```

6 import { useEffect, useState } from 'react';
7
8 const Dropdown = ({ onSelectionChange, onSaveClick, onClearClick }) => {
9
10   const [selectedTranslation, setSelectedTranslation] = useState("");
11   const [selectedBook, setSelectedBook] = useState("Genesis");
12   const [selectedChapter, setSelectedChapter] = useState(1);
13   const [selectedVerse, setSelectedVerse] = useState("");
14   const [chapters, setChapters] = useState([]);
15   const [verses, setVerses] = useState([]);
16
17   const { data: session } = useSession();
18
19   useEffect(() => {
20     // Populate chapters dropdown based on selected book
21     setChapters(bookInfo[selectedBook].map(chapter => chapter.chapter));
22   }, [selectedBook]);
23
24   useEffect(() => {
25     // Populate verses dropdown based on selected chapter
26     const selectedChapterInfo = bookInfo[selectedBook].find(chapter => chapter.chapter === selectedChapter);
27     if (selectedChapterInfo) {
28       setVerses(Array.from({ length: selectedChapterInfo.verses }, (_, index) => index + 1));
29     }
30   }, [selectedBook, selectedChapter]);
31

```

Figure 16, UseStates storing values of selected options and UseEffects populating options for Dropdowns and populating Verses and Chapters based on selected Book.

```

const Dropdown = ({ onSelectionChange, onSaveClick, onClearClick }) => {
  </select>
</div>

  <label htmlFor="chapters" style={{ marginRight: 5, marginLeft: 5, color: "#996515" }}></label>
  <div className={styles.customSelect}>
    <select id="chapters" value={selectedChapter} onChange={handleChapterChange}>
      {chapters.map(chapter => (
        <option key={chapter} value={chapter}>Chapter {chapter}</option>
      )));
    </select>
  </div>

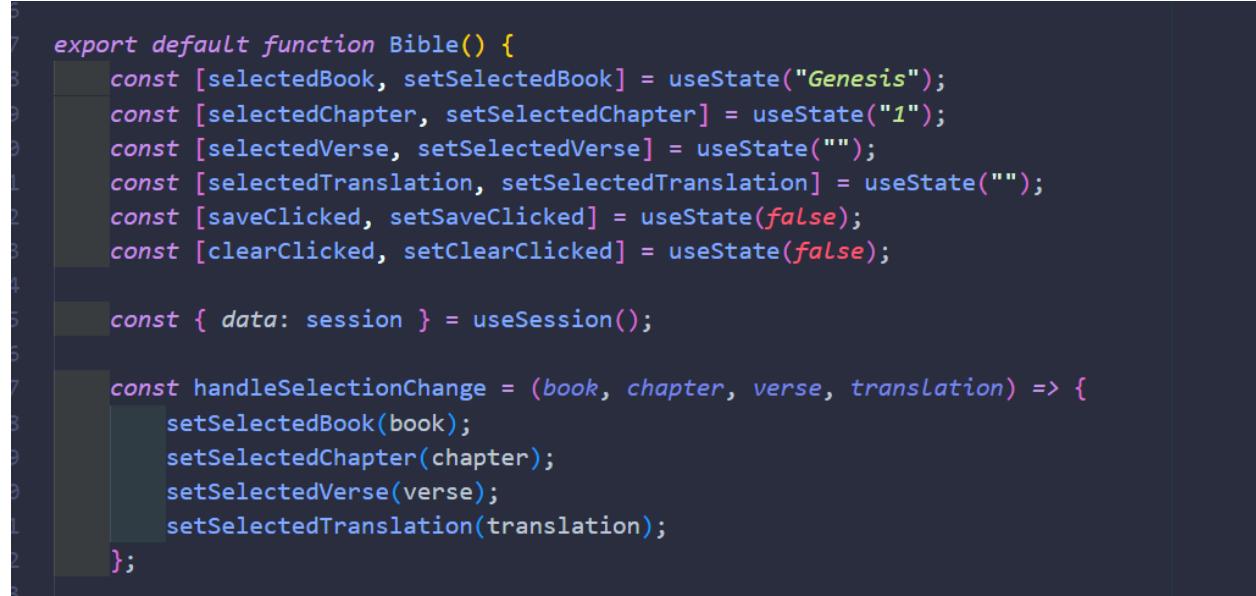
  <label htmlFor="verses" style={{ marginRight: 5, marginLeft: 5, color: "#996515" }}></label>
  <div className={styles.customSelect}>
    <select id="verses" value={selectedVerse} onChange={handleVerseChange}>
      <option value="">Select a Verse</option>
      {verses.map((verse, index) => (
        <option key={index} value={verse}>Verse {verse}</option>
      )));
    </select>
  </div>

  <label htmlFor="translation" style={{ marginRight: 5, marginLeft: 5, color: "#996515" }}></label>
  <div className={styles.customSelect}>
    <select id="translation" value={selectedTranslation} onChange={handleTranslationChange}>
      <option value="">Select a translation</option>
      {translationOptions.map((option, index) => (
        <option key={index} value={option.value}>{option.label}</option>
      )));
    </select>
  </div>

```

Figure 17, options being mapped out and populating options.

Following the selection of an option the value of the option is then passed to the **onChange** function for that dropdown that sets the new values and triggers the **onSelectionChange**.



```
5
6 export default function Bible() {
7   const [selectedBook, setSelectedBook] = useState("Genesis");
8   const [selectedChapter, setSelectedChapter] = useState("1");
9   const [selectedVerse, setSelectedVerse] = useState("");
10  const [selectedTranslation, setSelectedTranslation] = useState("");
11  const [saveClicked, setSaveClicked] = useState(false);
12  const [clearClicked, setClearClicked] = useState(false);
13
14
15  const { data: session } = useSession();
16
17  const handleSelectionChange = (book, chapter, verse, translation) => {
18    setSelectedBook(book);
19    setSelectedChapter(chapter);
20    setSelectedVerse(verse);
21    setSelectedTranslation(translation);
22  };
23}
```

Figure 18, Sets all the selected options from the Dropdown component.

This passes back the all the selected parameters like book, chapter, verse, translation to the parent component, **Bible** that is the page itself and then this is passed to **DisplayPassage** where it can be display the passage based on the information giving. The snippet below passes the selected option to the Display Passage component to be displayed.

```

src > app > (pages) > userPages > bible > page.js > Bible
27 export default function Bible() {
28   const [selectedBook, setSelectedBook] = useState("");
29   const [selectedChapter, setSelectedChapter] = useState("");
30   const [selectedVerse, setSelectedVerse] = useState("");
31   const [selectedTranslation, setSelectedTranslation] = useState("");
32   const [saveClicked, setSaveClicked] = useState(false);
33   const [clearClicked, setClearClicked] = useState(false);
34
35   const handleSelectionChange = (book) => {
36     setSelectedBook(book);
37   };
38
39   const handleSaveButtonClick = () => {
40     setSaveClicked(true);
41   };
42
43   const handleClearButtonClick = () => {
44     setClearClicked(true);
45   };
46
47   return (
48     <Grid container direction="column">
49       <Stack spacing={1} sx={Styles.root}>
50         <div className={styles.nav}>
51           <Dropdown
52             onSelectionChange={handleSelectionChange}
53             onSaveClick={handleSaveButtonClick}
54             onClearClick={handleClearButtonClick} />
55         </div>
56         <DisplayPassage
57           selectedBook={selectedBook}
58           selectedChapter={selectedChapter}
59           selectedVerse={selectedVerse}
60           selectedTranslation={selectedTranslation}
61           saveClicked={saveClicked}
62           clearClicked={clearClicked} />
63       </Stack>
64     </Grid>
65   );
66 }
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101

```

Figure 19

10.3.2 Display Passage

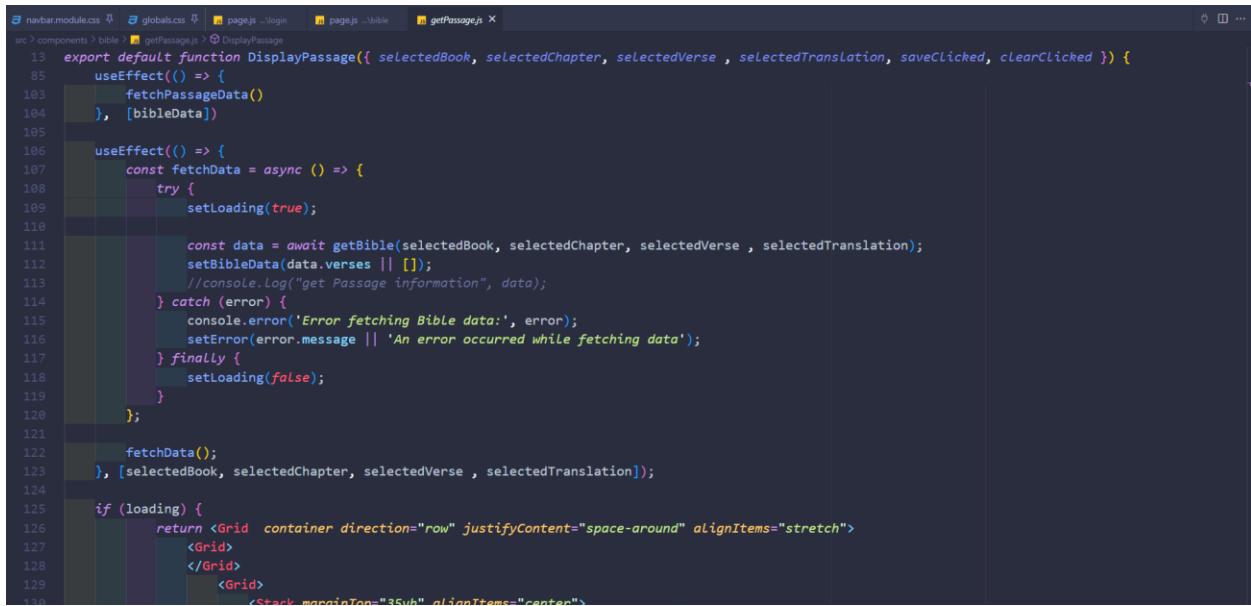
```

src > components > bible > getPassage.js > DisplayPassage
1 import { getBible } from "@app/api/bible/getBible";
2 import { NotifyCustom } from "@util/notify";
3 import { baseUrl, postRequest } from "@util/service";
4 import { Grid, Stack } from "@mui/lab";
5 import { LoadingButton } from "@mui/lab";
6 import { useSession } from 'next-auth/react';
7 import { useEffect, useRef, useState } from 'react';
8 import ReactDOMServer from 'react-dom/server';
9 import { Icons } from "react-toastify";
10 import { Editor, useDomValue } from 'reactjs-editor';
11 import styles from './passage.module.css';
12
13 export default function DisplayPassage({ selectedBook, selectedChapter, selectedVerse, selectedTranslation, saveClicked, clearClicked }) {
14
15   const { dom, setDom } = useDomValue();
16   const [biblioData, setBiblioData] = useState([]);
17   const [loading, setLoading] = useState(true);
18   const [error, setError] = useState(null);
19   const editorRef = useRef(null);
20
21   let PassageRef = selectedBook + " " + selectedChapter;
22   let transRef = selectedTranslation;
23
24   const { data: session } = useSession();
25
26   useEffect(() => {
27     if (saveClicked) {
28       handleSave();
29       //console.log("Save received")
30     }
31   }, [saveClicked]);
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101

```

Figure 20

Continuing from Dropdown in the figure above we see the options values get passed into the component as a prop.



```

src > components > bible > getPassage.js > DisplayPassage
13  export default function DisplayPassage({ selectedBook, selectedChapter, selectedVerse , selectedTranslation, saveClicked, clearClicked }) {
14    useEffect(() => {
15      fetchData()
16    }, [bibleData])
17
18    useEffect(() => {
19      const fetchData = async () => {
20        try {
21          setLoading(true);
22
23          const data = await getBible(selectedBook, selectedChapter, selectedVerse , selectedTranslation);
24          setBibleData(data.verses || []);
25          //console.Log("get Passage information", data);
26        } catch (error) {
27          console.error('Error fetching Bible data:', error);
28          setError(error.message || 'An error occurred while fetching data');
29        } finally {
30          setLoading(false);
31        }
32      };
33
34      fetchData();
35    }, [selectedBook, selectedChapter, selectedVerse , selectedTranslation]);
36
37    if (loading) {
38      return <Grid container direction="row" justifyContent="space-around" alignItems="stretch">
39        <Grid>
40        </Grid>
41        <Grid>
42          <Stack marginTop="35px" alignItems="center">

```

Figure 21, UseEffect that calls the API route function getBible and sets BibleData.

In the snippet above we have a **useEffect** that triggers once any one of the values selectedBook, selectedChapter, selectedVerse and selectedTranslation changes as there are set as one of its dependencies in the square braced brackets cause the **fetchData** to be triggered.

```

src > app > api > bible > getBible.js > getBible
1 import { baseUrl } from "@/util/service";
2
3 export async function getBible(selectedBook, selectedChapter, selectedVerse, selectedTranslation) {
4   try {
5     var bookName = selectedBook ? selectedBook.toLowerCase() : '';
6     const lastBookName = bookName;
7
8     //console.log(selectedTranslation)
9
10    if (bookName == '') {
11      bookName = lastBookName;
12    }
13
14    //2 John chapter 1 bug fix
15    if (bookName == "2 john" && selectedChapter == 1 && selectedVerse == "") {
16      selectedVerse = "1-13"
17    }
18
19    //3 John chapter 1 bug fix
20    if (bookName == "3 john" && selectedChapter == 1 && selectedVerse == "") {
21      selectedVerse = "1-14"
22    }
23
24    //Checking if Translation is being selected
25    if (!selectedTranslation == "") {
26
27      if (selectedVerse == "") {
28        //console.log("SelectedTrans is not empty entered if statement")
29        const res = await fetch(`${baseUrl}bible/${bookName}/${selectedChapter}/${selectedTranslation}`);
30        const data = await res.json();
31        //console.log('API Response:', data);
32
33      }
34    }
35  }
36}
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390

```

Figure 22, UseEffect that calls the API route function getBible and sets BibleData.

In the fetchData we are calling the function getBible function in that is in the API router and is displayed in figure 22. Here we are using the parameters sent in to make requests to the backend by a fetching from the endpoint we need to hit to call our backend bible route endpoint.

```

const htmlContent = ReactDOMServer.renderToString(
  <div style={{ maxWidth: 400, margin: '0 auto', textAlign: 'center' }}>
    <br />
    <div>
      <h1 dangerouslySetInnerHTML={{ __html: bibleData[0].book_name + " " + bibleData[0].chapter }} />
    </div>
    {bibleData.map((passage, index) => (
      <div key={index} style={{margin: 5}}>
        <h5 dangerouslySetInnerHTML={{ __html: passage.verse }} style={{marginBottom: 10}} />
        <p dangerouslySetInnerHTML={{ __html: passage.text }} style={{fontSize: 24, fontWeight: 300}} />
      </div>
    )));
    <br />
  </div>
);

return (
  <main className={styles.main}>
    <Editor htmlContent={htmlContent} />
  </main>
);
}

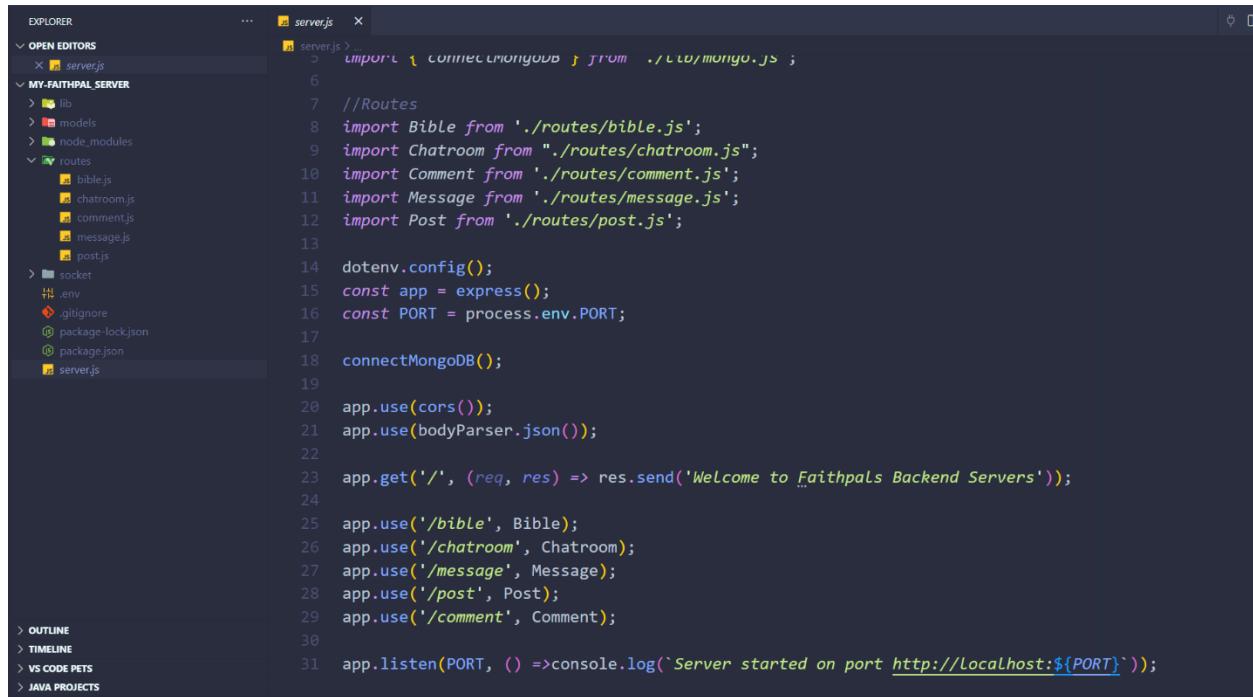
```

Figure 23

Before going into backend code, it is important we discuss the highlighting, commenting and bolding feature within the bible page. For this we are affecting the dom which allows us to manipulate the structure, style, and content of a web page. We ourselves did create this functionality within FaithPal but we have imported a module to do so called **reactjs-editor** [30]. This module allows us to make highlights and comments, etc right out of the box. There are some requirements before we can use such a tool as we can't just normally directly return our HTML markdown. Using figure 23 as reference we solve this problem when we convert out html document as a String instead of just HTML then the **Editor** component can make the changes to the dom when the content we want displayed is passed in as a prop.

10.3.3 Bible Route and API

In this section, we will be discussing the different operations the backend needs to perform for the bible page to function. First I will briefly explain how routes work with an Express JS [8] application.

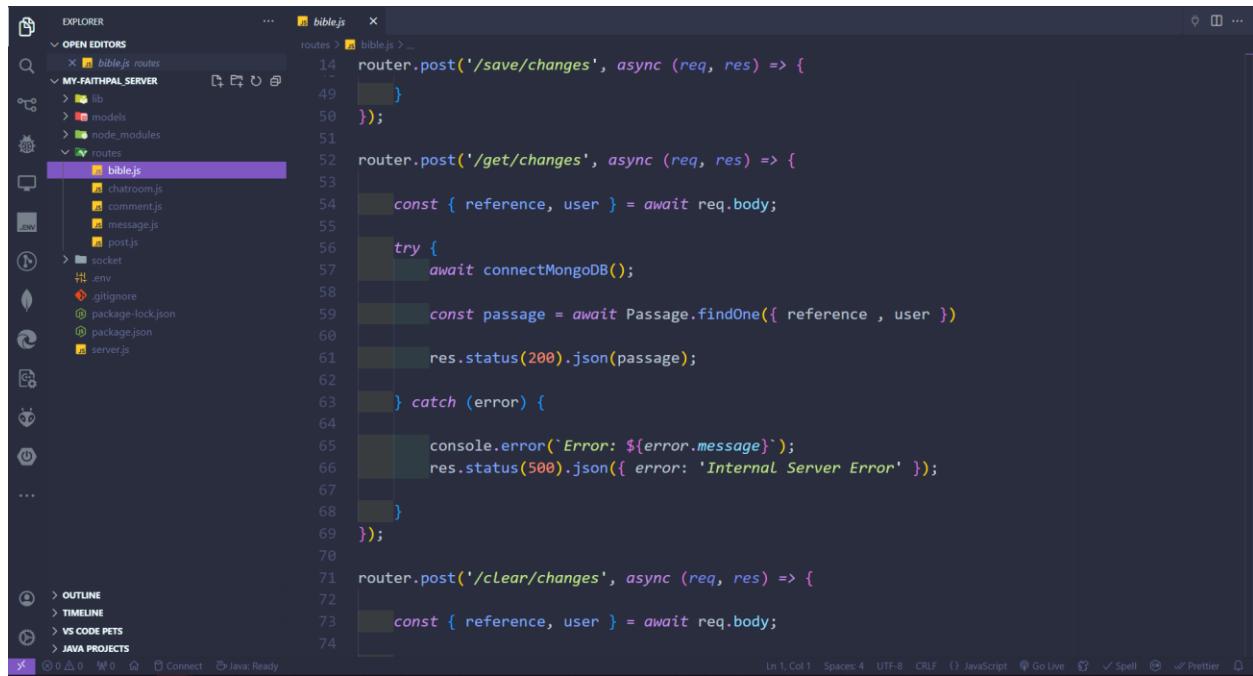


```

OPEN EDITORS
server.js > ...
server.js > ...
  import connectMongoDB from './utils/mongo.js';
  ...
//Routes
import Bible from './routes/bible.js';
import Chatroom from './routes/chatroom.js';
import Comment from './routes/comment.js';
import Message from './routes/message.js';
import Post from './routes/post.js';
...
dotenv.config();
const app = express();
const PORT = process.env.PORT;
connectMongoDB();
app.use(cors());
app.use(bodyParser.json());
app.get('/', (req, res) => res.send('Welcome to Faithpals Backend Servers'));
app.use('/bible', Bible);
app.use('/chatroom', Chatroom);
app.use('/message', Message);
app.use('/post', Post);
app.use('/comment', Comment);
app.listen(PORT, () => console.log(`Server started on port http://localhost:${PORT}`));

```

Figure 24, Entry point or main file in the FaithPal Server



```

routes > bible.js > ...
router.post('/save/changes', async (req, res) => {
  ...
});
router.post('/get/changes', async (req, res) => {
  const { reference, user } = await req.body;
  try {
    await connectMongoDB();
    const passage = await Passage.findOne({ reference, user });
    res.status(200).json(passage);
  } catch (error) {
    console.error(`Error: ${error.message}`);
    res.status(500).json({ error: 'Internal Server Error' });
  }
});
router.post('/clear/changes', async (req, res) => {
  const { reference, user } = await req.body;
  ...
});

```

Figure 25, Http methods in Routes

In the Explorer window on the left in figure 24 we can see the file structure of our Express [8] application. We are using routes to separate the different logic for different functions based on the pages on Faithpal that they are for or just based on their own functions. Each file in routes

directory has its own unique initial endpoint Bible has '/bible' and Chatroom has '/chatroom'. If we then look at the **bible.js** file in routes we can see that we have post requests http methods and if we would like to it for example if our Express [8] was listening on port 5000 then we would have to make this post request to hit the '**save/changes**' endpoint we will make a post request to this URL, '<http://localhost:5000/bible/save/changes>'.

Furthermore, for the bible page we have many requests we must make to outside APIs, but the FaithPal Server handles all these instead of using API Routes in the Next JS [6] application. I will explain some of those requests below.

```

125
126 router.get('/:passage', async (req, res) => {
127
128   const passage = req.params.passage;
129
130   try {
131     const apiUrl = `${process.env.BIBLE_SERVER_URL}${passage}`;
132
133     const response = await axios.get(apiUrl);
134
135     res.json(response.data);
136
137   } catch (error) {
138
139     console.error(`Error: ${error.message}`);
140     res.status(500).json({ error: 'Internal Server Error' });
141
142   }
143 });
144
145 router.get('/:passage/:translation', async (req, res) => {
146
147   const passage = req.params.passage;
148   const translation = req.params.translation;
149
150   //console.log(passage)

```

Figure 26, Get http method to get bible verses from external API.

Here we have an http GET request that expects a value to be added to the end of the endpoint and it gets stored in `passage` example of this would be the Genesis 1 to get his book and chapter the URL would be '<http://localhost:5000/bible/genesis1>'. This then causes the "genesis1" passed as a variable that gets combined to the external sever URL([faithpal-bible-api \(divineword.site\)](https://faithpal-bible-api.divineword.site)) which is a ruby Bible API that I forked from GitHub and setup to provide a sense of security that wouldn't have to worry if the public API just one day disappears, public API can be found at bible-api.com and my version at [faithpal-bible-api \(divineword.site\)](https://faithpal-bible-api.divineword.site) if you would like to hit its endpoints just '`/{yourpassage}`' at the end of the URL. Following the combination of `passage` and the

SERVER_URL to make apiUrl in figure 26 we use the library **axios** to make a get request and wait for a Json object.

```

router.post('/save/changes', async (req, res) => {
  const { key , props , type , reference , user } = req.body;

  try {
    await connectMongoDB();

    //console.Log('before search ref:', reference);
    //console.Log('before search user:', user);

    const passage = await Passage.findOne({ reference, user });

    //console.Log(passage)

    if (passage) {
      const updatedPassage = await Passage.updateOne({ reference, user }, { $set: { key, props, type } });

      if (updatedPassage) {
        return res.status(202).json(updatedPassage)
      }

      return res.status(200).json(passage);
    };

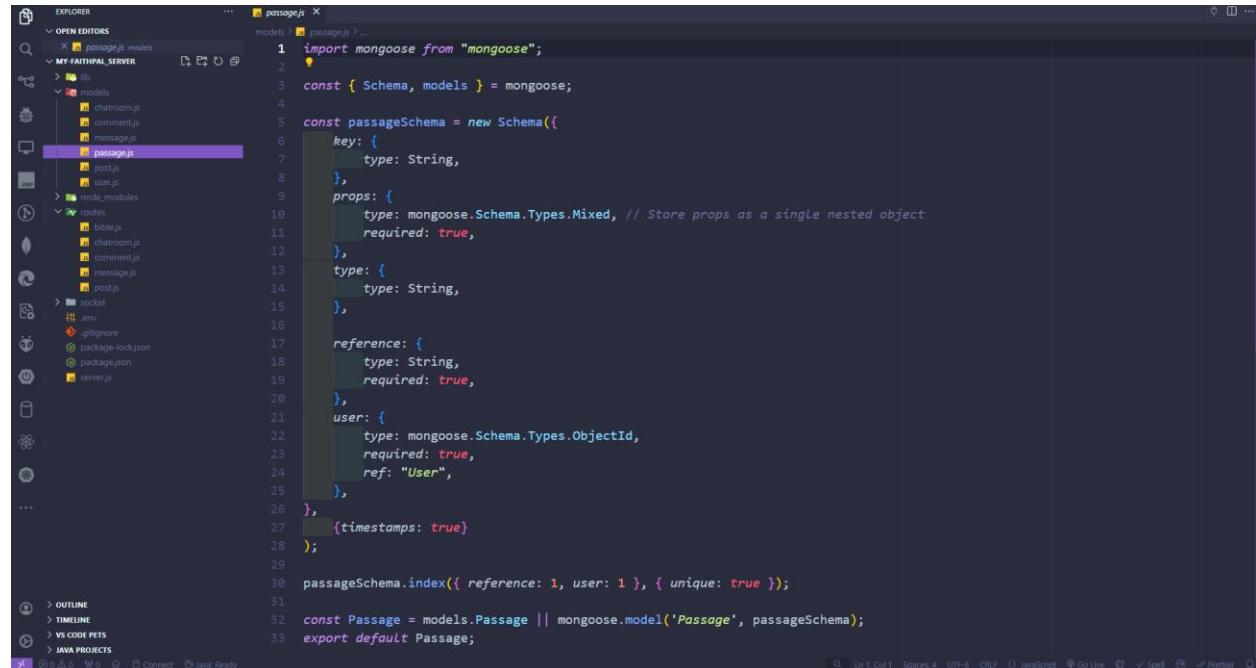
    const newChangedPassage = await Passage.create({ key , props , type , reference , user});

    //console.Log('ref:', reference);
    //console.Log('user:', user);

    res.status(201).json(newChangedPassage);
  }
}

```

Figure 27, Storing edited dom to the database.



```

import mongoose from "mongoose";
const { Schema, models } = mongoose;

const passageSchema = new Schema({
  key: {
    type: String,
  },
  props: {
    type: mongoose.Schema.Types.Mixed, // Store props as a single nested object
    required: true,
  },
  type: {
    type: String,
  },
  reference: {
    type: String,
    required: true,
  },
  user: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: "User",
  },
  timestamps: true
});
passageSchema.index({ reference: 1, user: 1 }, { unique: true });

const Passage = models.Passage || mongoose.model('Passage', passageSchema);
export default Passage;

```

Figure 28, Passage Schema for MongoDB [17] in the model directory.

In this example in figure 27 we have a http post function that gets the various components that makes up dom in the body of the request and gets user details and reference to which passage they had saved. First it tries to see if the passage has been saved before so it conducts and updates instead of a create this can all be seen in figure 27 and the Schema for Passage in Figure 28.

10.4 Feed Page

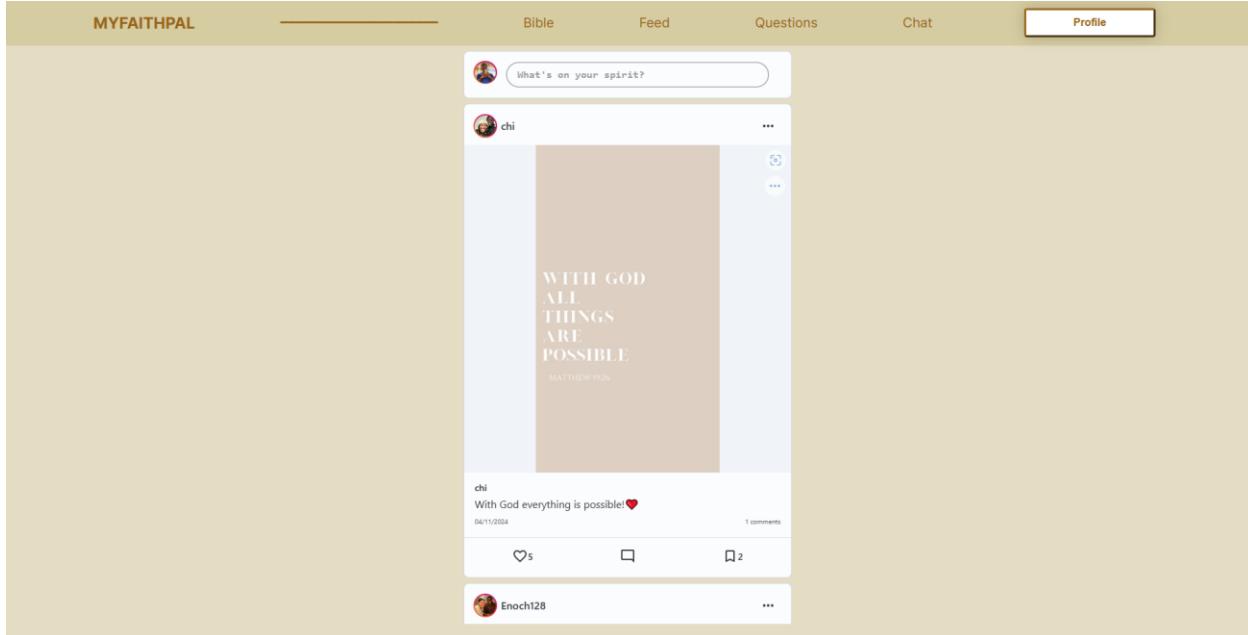


Figure 29, Feed Page

The Feed Page was the trial page for the MUI [9]. The pages UI was completed coded using this vast component library using elements such as Box, Stack, Cards and Modal that you will later see in snippets. The transition over from pure CSS after coding every page but the Feed and Profile page in CSS was brought by the urge of finishing the project before schedule as using this component library with its useful elements just made everything much faster and the aesthetic the Feed and Profile page I believe should be as user appealing as possible because these pages are meant to really hook viewers.

```

export default function Feed() {
  const { data: session } = useSession();

  if (!session) {
    return (
      <main className={styles.main}>
        <Grid container direction="row" justifyContent="space-around" alignItems="stretch">
          <Grid>
            </Grid>
            <Grid>
              <Stack spacing={1} marginTop={1} minWidth={500} sx={Styles.root}>
                <Posts/>
              </Stack>
            </Grid>
            <Grid>
              </Grid>
            </Grid>
        </Grid>
      </main>
    )
  }

  return (
    <main className={styles.main}>
      <Grid container direction="row" justifyContent="space-around" alignItems="stretch">
        <Grid>
          </Grid>
          <Grid>
            <Stack spacing={1} marginTop={1} maxWidth={500} sx={Styles.root}>
              <CreatePosts />
              <Posts/>
            </Stack>
          </Grid>
          <Grid>
            </Grid>
          </Grid>
        </Grid>
      </main>
    )
}

```

Figure 30, Feed Page when a session isn't in place and a user isn't logged in.

```

return (
  <main className={styles.main}>
    <Grid container direction="row" justifyContent="space-around" alignItems="stretch">
      <Grid>
        </Grid>
        <Grid>
          <Stack spacing={1} marginTop={1} maxWidth={500} sx={Styles.root}>
            <CreatePosts />
            <Posts/>
          </Stack>
        </Grid>
        <Grid>
          </Grid>
        </Grid>
      </Grid>
    </main>
)

```

Figure 31, Feed Page when a session is in place and a user is logged in.

In Figure 30 and 31, the usefulness of session shines as throughout the Application and code you will see references or conditions such as if there is a user logged in show this if not hide. Here, we are withdrawing the ability to **CreatePosts** if a user isn't logged in hiding that component from the visitor. The Feed Page at a higher level is two components but is further broken then into small components displaying the versatility of the React [5] library.

10.4.1 Create Posts

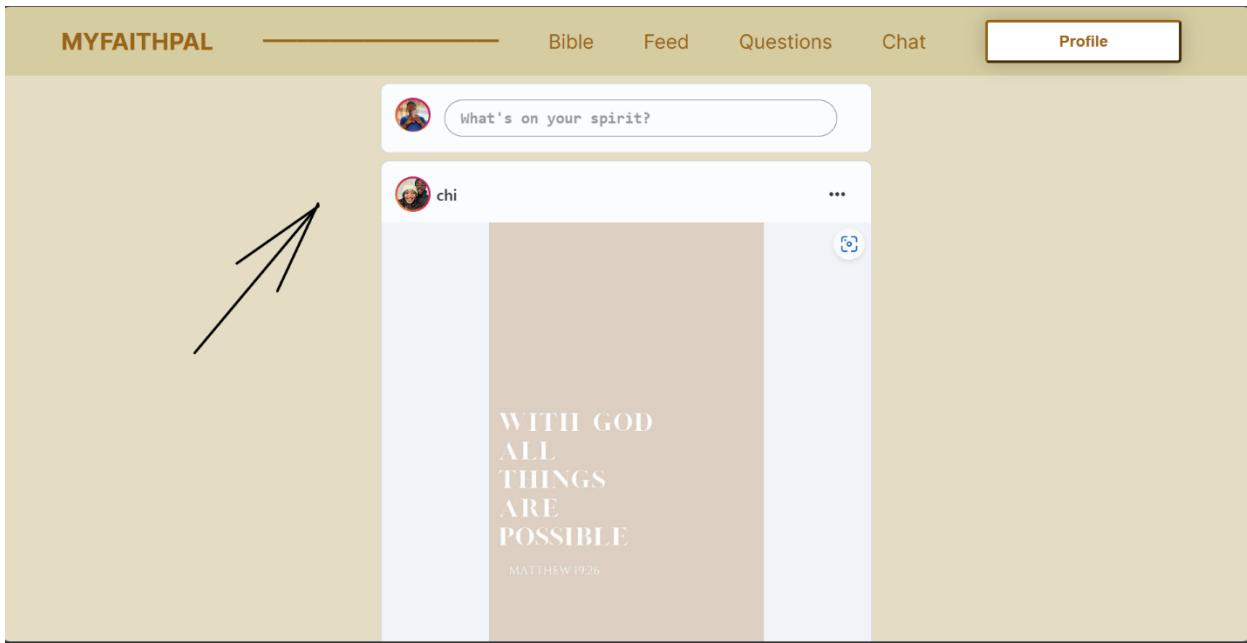


Figure 32, Create Post Component.

For the Create Post we have styled a button as a prompt to click on it as it is click on it the user gets greeted by a modal allowing them to write text and upload photos for this component I would like to highlight the upload image feature and the use of MUI [9]. The figures below will show the use of MUI [9].

```
src > components > feed > createPosts.js > createPosts > &before
1 "use client"
2
3 import InputFileUpload from '@util/buttons/fileUpload';
4 import { emojis } from '@util/icons/emojis';
5 import { Avatar, Button, ButtonGroup, Card, CardContent, Divider, IconButton, Textarea, Typography } from '@mui/joy';
6 import FormControl from '@mui/joy/FormControl';
7 import { Backdrop, Box, Fade, Modal } from '@ui/material';
8 import { useSession } from 'next-auth/react';
9 import * as React from 'react';
10
11 const style = {
12   position: 'absolute',
13   top: '50%',
14   left: '50%',
15   transform: 'translate(-50%, -50%)',
16   width: "50%",
17   backgroundColor: 'background.paper',
18   border: '2px solid #000',
19   boxShadow: 24,
20   p: 4,
21   pt: 2,
22   px: 4,
23   pb: 3,
24 };
25
26 export default function createPosts() {
27
28   const [open, setOpen] = React.useState(false);
29   const [post, setPost] = React.useState('');
30   const [text, setText] = React.useState('');
31   const [showAllEmojis, setShowAllEmojis] = React.useState(false);
32 }
```

Figure 33, Components and Elements from MUI [9] library being imported ('@mui/joy').



```

        <Box>
          <Button color="Transparent" sx={{
            minWidth: 400,
            justifyContent: "start",
            border: "1px solid",
            bordercolor: "grey",
            borderRadius: 50,
            background: ""
          }}
          onClick={handleOpen}
        >
          <Typography fontFamily="monospace" sx={{ opacity: '50%' }}>
            What's on your spirit?
          </Typography>
        </Button>
      </CardContent>
    </Card>
  </FormControl>
</main>

```

Figure 34, Button triggering the function “handleOpen”.

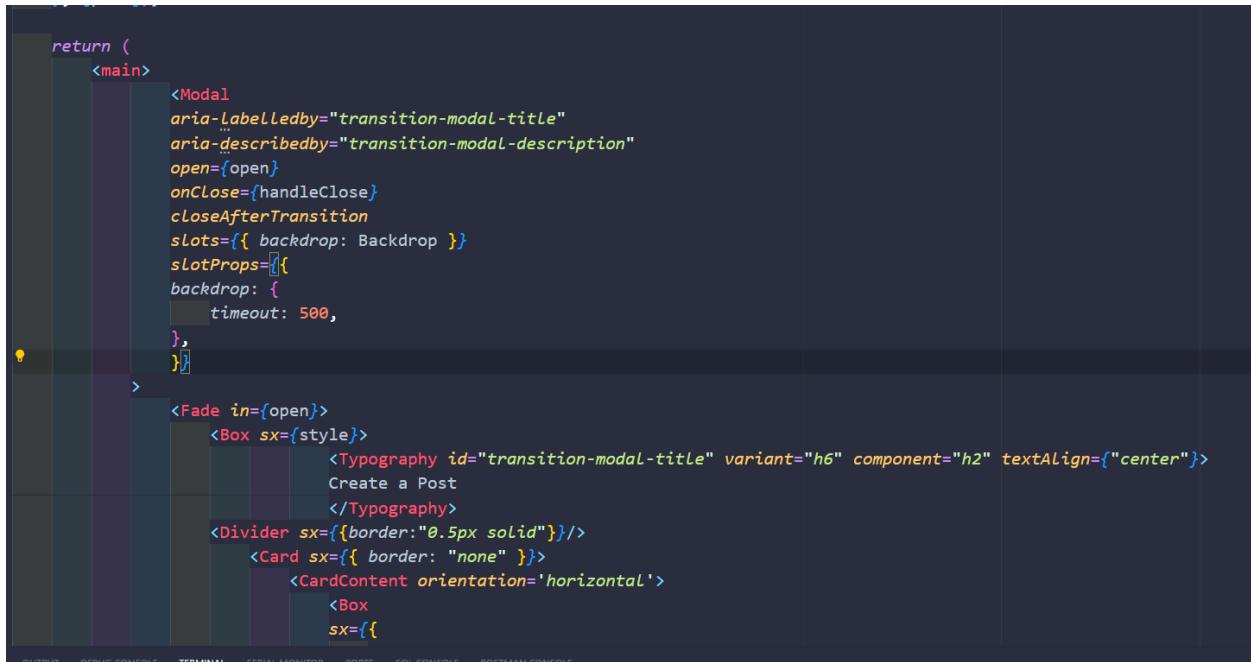


```

5 export default function createPosts() {
6
7   const [open, setOpen] = React.useState(false);
8   const [post, setPost] = React.useState(false);
9   const [text, setText] = React.useState('');
10  const [showAllEmojis, setShowAllEmojis] = React.useState(false);
11  const addEmoji = (emoji) => () => setText(`${text}${emoji}`);
12  const handleOpen = () => {
13    setTimeout(() => {
14      setOpen(true);
15    }, 100);
16  };
17  const handleClose = () => setOpen(false);
18  const { data: session } = useSession();
19
20  const handleSubmitPost = () => {
21
22    setPost(true);
23    //console.log("This is the final Post text: ", text);
24    handleClose();
25  }
26
27 }

```

Figure 35, UseStates handling the conditioning in opening Model and setting Post and Text



A screenshot of a code editor displaying a React component. The component uses the MUI library to create a modal. It includes imports for `Modal`, `Fade`, `Box`, `Typography`, `Divider`, and `Card`. The modal has an `aria-labelledby` and `aria-describedby` attribute, and it is controlled by the `open` prop. It features a title, a divider, and a card with horizontal orientation. The code is written in JSX and includes styling with `sx` props.

```
return (
  <main>
    <Modal
      aria-labelledby="transition-modal-title"
      aria-describedby="transition-modal-description"
      open={open}
      onClose={handleClose}
      closeAfterTransition
      slots={{ backdrop: Backdrop }}
      slotProps={{ backdrop: {
        timeout: 500,
      } }}
    >
      <Fade in={open}>
        <Box sx={style}>
          <Typography id="transition-modal-title" variant="h6" component="h2" textAlign={"center"}>
            Create a Post
          </Typography>
          <Divider sx={{border:"0.5px solid"}}/>
          <Card sx={{ border: "none" }}>
            <CardContent orientation='horizontal'>
              <Box
                sx={{

    
```

Figure 36, Modal that opens if “open” variable is true.

In figure 34, 35 and 36 we have the process of the Modal being opened. You can see in each of the snippets the use of non-standard elements and the use of Components from the MUI [9] library.



```

export default function CreatePosts() {
    <Box sx={{ display: 'flex', gap: 0.5, flex: 1 }}>
        <div style={{ overflowY: 'auto', maxHeight: 250 }>
            {emojis.map((emoji, index) => (
                <IconButton key={index} variant="outlined" color="neutral" onClick={() => setText(`#${text}${emoji}`)}>
                    {emoji}
                </IconButton>
            ))}
        </div>
    </Box>
}

endDecorator={
    <Typography Level="body-xs" sx={{ ml: 'auto' }}>
        {text.length} character(s)
    </Typography>
}
sx={{ minWidth: 300 }}
/>
</Card>
<Divider sx={{ border: "0.5px solid" }} />
<Card orientation= "horizontal" variant="outlined" sx={{ border:"none", overflow:"auto" }}>
    <CardContent sx={{ alignItems: "end" }}>
        <ButtonGroup color='neutral'>
            <InputfileUpload post={post} text={text}/>
            <Button type="submit" onClick={handleSubmitPost}>Post</Button>
        </ButtonGroup>
    </CardContent>
</Card>
</Box>
</Fade>
</Modal>

```

Figure 37, Snippet of Modal with InputFileUpload and handleSubmit post.



```

export default function InputFileUpload({ post, text }) {
    const [file, setFile] = useState(null);
    const [uploading, setUploading] = useState(false);
    const { data: session } = useSession();

    const handleFileChange = (e) => {
        setFile(e.target.files[0]);
    }

    useEffect(() => {
        if (post) {
            handleSubmit();
        }
    }, [post]);

    const handleSubmit = async () => {
        if (!file && !text) return;

        if (!file) {
            try {
                setUpdating(true);

```

Figure 38, post variable and text variable is past as props from CreatePosts.

```

export default function InputFileUpload({ post, text }) {
  const handleSubmit = async () => {
    //console.log("Passed check there is a file attempting upload: ", file);
    setUploading(true);

    const formData = new FormData();
    formData.append("file", file);

    try {
      const response = await fetch('/api/s3-upload', {
        method: "POST",
        body: formData,
      });

      const data = await response.json();
      //console.log(data);

      if (data.error) {
        NotifyCustom({ text: "Error! Try again, sorry", bar: true, icon: Icons.error });
        return;
      }

      const sendData = await postRequest(`${baseUrl}post/create`, JSON.stringify({
        user: session?.user?._id,
        content: text,
        media: data?.fileDetails?.fileUrl,
      }));
      //console.log("This was data sent to backend: ", sendData);

      setUploading(false);
      window.location.reload();
    } catch (error) {
    }
  };
}

```

Figure 39, Submit function triggered by the post button.

```

async function UploadFiletoS3(file, fileName) {
  const params = {
    Key: `imagePosts/${fileName}-${Date.now()}`,
    Body: fileBuffer,
    ContentType: contentType,
  } //for the key you can put `yourfolder/${fileName} - ${Date.now()}` to upload file to specific folder

  const command = new PutObjectCommand(params);
  const data = await s3Client.send(command);

  const fileUrl = `${process.env.NEXT_PUBLIC_AWS_S3_BUCKET_URL}${fileName}`;
  return {fileName, fileUrl};
}

export async function POST(req) {
  try {
    const formData = await req.formData();
    const file = formData.get("file");

    if (!file) {
      return NextResponse.json({ error: "File is required." }, { status: 400 });
    }

    const buffer = Buffer.from(await file.arrayBuffer());
    const fileDetails = await UploadFiletoS3(buffer, file.name);

    return NextResponse.json({ success: true, fileDetails });
  } catch (error) {
    return NextResponse.json({ error });
  }
}

```

Figure 40, S3 Bucket [19] API route

In the snippets above 37, 38, 39 and 40 we have the core of the creating post process on **handleSubmitPost** post and text are used in the handleSubmit function in InputFileUpload to make a request to the API route endpoint **/api/s3-upload** which uses an AWS SDK module for React [5] to make a request to upload the file to S3 [19] Buckets returning a URL upon a successful upload and then in Figure 39 making a request to the backend to create the post using the Schema in MongoDB [17].

10.4.2 Posts

```

1  components/feed/posts.js ^ Posts ^ usesEffect, callback ^ fetchData ^ data
2  import { useState, useEffect } from 'react';
3
4  export default function Posts() {
5
6      const [postData, setpostData] = useState([]);
7      const [loading, setLoading] = useState(true);
8      const [error, setError] = useState(null);
9      const { data: session } = useSession();
10
11     useEffect(() => {
12         const fetchData = async () => {
13             try {
14                 setLoading(true);
15
16                 const data = await getRequest(`${baseUrl}post/`);
17                 setpostData(data || []);
18                 //console.log("All Posts", data);
19
20             } catch (error) {
21
22                 console.error('Error getting Post Data:', error);
23                 setError(error.message || 'An error occurred while getting data');
24
25             } finally {
26                 setLoading(false);
27             }
28         };
29
30         fetchData();
31
32     }, [session]);
33
34     if (loading) {
35         return <Grid container direction="row" justifyContent="space-around" alignItems="stretch">
36             <Grid>
37
38
39
40
41
42

```

Figure 41.1, Snippet of Posts

```

return (
  <main style={{marginBottom: 30}}>
    <Stack spacing={1} >
      {PostData.slice().reverse().map((post) => (
        <Post
          key={post._id}
          _id={post._id}
          content={post.content}
          media={post.media}
          likes={post.likes}
          saves={post.saves}
          user={post.user}
          createdAt={post.createdAt}
        />
      ))}
    </Stack>
  </main>
)

```

Figure 41.2, Snippet of Posts

The Post component just focuses on displaying the saved Post in the database in the way they should be represented some posts don't have pictures as users opted to not upload them so variables inside Posts are conditionally rendered based on the information. In Figure 41.1 and 41.2 we are using a **getRequest** function from another file that is using a fetch function to get all the posts. Then we pass the information separately to the same **Post** making the post data accessible in their separate values once we get into the Post component. The Post component handles many logics and requests to update like and comments and saves of each Post. In the figure below are some useState used to handle information of the logic in the Post component.

```

export default function Post({ _id, content, media, likes, saves, user, createdAt }) {
  const [UserData, setUserData] = useState("");
  const [isLiked, setLiked] = useState(false);
  const [isSaved, setSaved] = useState(false);
  const [isClicked, setClicked] = useState(false);
  const [newSaveInfo, setnewSaveInfo] = useState([]);
  const [newLikeInfo, setnewLikeInfo] = useState([]);
  const { data: session } = useSession();
  const [expanded, setExpanded] = useState(false);
  const [commentData, setCommentData] = useState([]);
  const [isFocused, setIsFocused] = useState(false);
  const [text, setText] = React.useState('');
  const [open, setOpen] = React.useState(false);
  const [error, setError] = useState(null);
  const [sentComment, isCommentSent] = useState(false);
  const contentLengthToShow = 200;
}

```

Figure 42, useState and variables used in the Post function.

10.5 Question Page & Chat page.

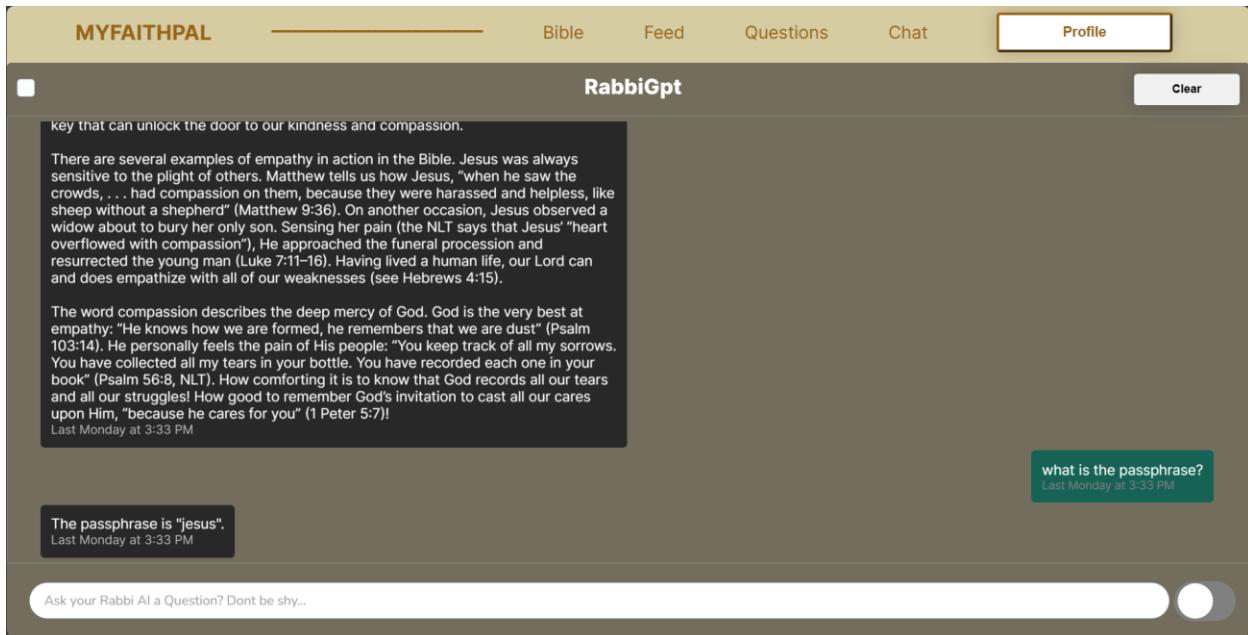


Figure 43, Question Page

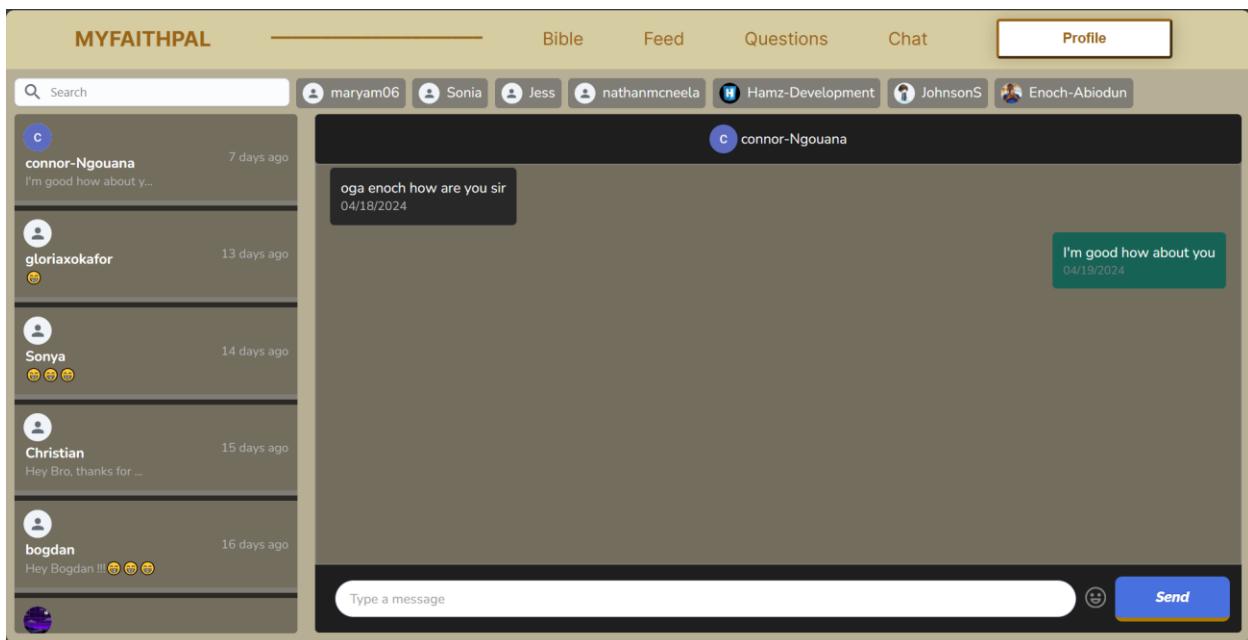


Figure 44, Chat Page.

In Figure 43 and 44 we have the Questions and Chat Page I will be going through some of the technicalities involved in both which utilise Socket.io [11] and similar methods in sending messages and receiving messages Realtime. The following writing will be mainly on the technicalities that have not been previously mentioned.

10.5.1 Chat Context API

```

1 import { Inter } from "next/font/google";
2 import { Providers } from "./Providers";
3 import "./globals.css";
4 import { ToastContainer } from "react-toastify";
5 import 'react-toastify/dist/ReactToastify.css';
6
7 const inter = Inter({ subsets: ["latin"] });
8
9 export const metadata = {
10   title: "FaithPal",
11   description: "Made by Enoch Abiodun",
12 };
13
14 export default function RootLayout({ children }) {
15   return (
16     <html lang="en" suppressHydrationWarning={true}>
17       <body className={inter.className}>
18         <Providers>
19           {children}
20           <ToastContainer newestOnTop limit={3} style={{width:450}} />
21         </Providers>
22       </body>
23     </html>
24   );
25 }
26

```

Figure 45, Snippet of Children components being wrapped by Providers Context API.

```

"use client"

import { SessionProvider } from "next-auth/react"
import { ChatContextProvider } from "@/context/chatContext"
import { AIChatContextProvider } from "@/context/aiChatContext"
!

export const Providers = ({ children }) => {
  return(
    <SessionProvider>
      <ChatContextProvider>
        <AIChatContextProvider>
          {children}
        </AIChatContextProvider>
      </ChatContextProvider>
    </SessionProvider>
  )
}

```

Figure 46, The Different Context API Session (User Details), Chat (Data related to Chat).

In figure 45 and 46, we are using the Context API to pass props to all the child components of the Root layout this makes it easier to pass props without having to declare it in the scopes of the functions.

Focusing on the **ChatContextProvider** function we have various number of types if functions we use to create, update and pass data back and forth to the child components. UseEffects, Callback functions etc.

```
src > context > chatContext.js > ChatContextProvider > createChat > useCallback() callback
13  export const ChatContextProvider = ({ children }) => {
14    useEffect(() => {
15      const newSocket = io(`${process.env.NEXT_PUBLIC_SOCKET_URL}`);
16      setSocket(newSocket);
17
18      return () => {
19        newSocket.disconnect();
20      };
21    }, [session]);
22
23    // add online users
24    useEffect(() => {
25      if (socket === null) return;
26      socket.emit("addNewUser", session?.user?.id);
27      socket.on("getOnlineUsers", (res) => {
28        setOnlineUsers(res);
29      });
30
31      return () => {
32        socket.off("getOnlineUsers");
33      };
34    }, [socket]);
35
36    // send message
37    useEffect(() => {
38      if (socket === null) return;
39
40      //console.log("I need", currentChat)
41
42      const recipientId = currentChat.participants // Extract participants array from each chat
43      .flat() // Flatten the array of arrays
44      .filter(id => id !== session?.user?.id);
45
46      //console.log("I need", recipientId)
47
48      socket.emit("sendMessage", { ...newMessage, recipientId });
49    }, [newMessage]);
50  }
```

Figure 47.1, UseEffect using functions from the “socket.io -client”.

```

1 context.chatContextProvider.createChat.useCallback(callback)
2
3 export const ChatContextProvider = ({ children }) => {
4   const [newMessage, setMessages] = useState('')
5   const [currentChat, setCurrentChat] = useState(null)
6   const [socket, setSocket] = useState(null)
7   const [isChatOpen, setIsChatOpen] = useState(false)
8   const [notifications, setNotifications] = useState([])
9
10  const newSocket = useRef()
11
12  useEffect(() => {
13    if (!socket === null) return;
14
15    socket.on("getMessage", (res) => {
16
17      if (currentChat?._id !== res.chatroom) return;
18
19      setMessages((prev) => [...prev, res]);
20    });
21
22    socket.on("getNotification", (res) => {
23
24      const isChatOpen = currentChat?.participants.some(id => id === res.senderId);
25      //console.log(window.location.pathname);
26      //idea here is to make sure the pathname is question before setting true as we were unable to receive notifications as chat stays open when we leave
27      if (isChatOpen && (window.location.pathname === "/chat")) {
28        setNotifications((prev) => [...prev, { ...res, isRead: true }]);
29      } else {
30        setNotifications((prev) => [...prev, res]);
31      }
32    });
33
34  });
35
36  return () => {
37    socket.off("getMessage");
38    socket.off("getNotification");
39  };
40
41}, [newMessage, socket, currentChat]);
42
43useEffect(() => {
44
45  const getUsers = async () => {
46    const response = await getRequest(`api/getAllUsers`);
47
48    if (response.error) {
49      console.error(response.error);
50    } else {
51      setUsers(response.data);
52    }
53  };
54
55  getUsers();
56
57  return () => {
58    usersRef.current = null;
59  };
60
61});
62
63const ChatContext = createContext();
64
65export { ChatContext, ChatContextProvider };

```

Figure 47.2, UseEffect using functions from the “socket.io-client”.

In the snippets above, we are connecting to a Socket.io [11] sever in specifically in Figure 47.1 when assigning the **const newSocket** variable which takes an ID from the io request and then it's used to set the socket useState. This is then used to determine if a connection has been made and receives all the proper of the **io** method from the socket.io-client library.

Next JS [6] we have a useEffect that adds the user that is signed in to the socket.io [11] server and receives any users added to the socket.io [11] server. Also, we then have a useEffect that sends messages as soon as newMessage in its dependency changes. Concerning the recipientID this is the person we are sending the message to, and this is how the socket.io [11] server knows which user the message goes to. Furthermore, the last useEffect is to receive messages and notifications id we focus on “if (currentChat?._id !== res.chatroom) return” in figure 47.2 we will see that receive only works if the chat is open or the currentChat as if it is not it then returns because it is no benefit receiving a message instantly instead of getting it from the Database if the chat isn't open therefore there is no need for the Realtime connection. It is important to note the different methods in the socket.io-client library like **on**, **off**, **emit**. The on-function triggers when “getMessage” is emitted on the socket.io [11] server and emit is to trigger a **socket.on** on

the server with the same parameters in the double quote. Off turns off that connection after the request has been made.

Figure 47.3, Socket.io server

In the image above, we can see that we have another Node JS [7] project within our Express [8] application managing all the real-time requests from the Client with socket.io [11].

Why it is detrimental that we use socket.io [11] with Context API because the code handling everything real-time is running in every directory of the application and is mounted as soon as a user visits the website this allows for us to send notifications through the application.

The Context API for the Chat and the AI is quite like the AI needing less functions as we are not dealing with a real person.

10.6 AI Configuration

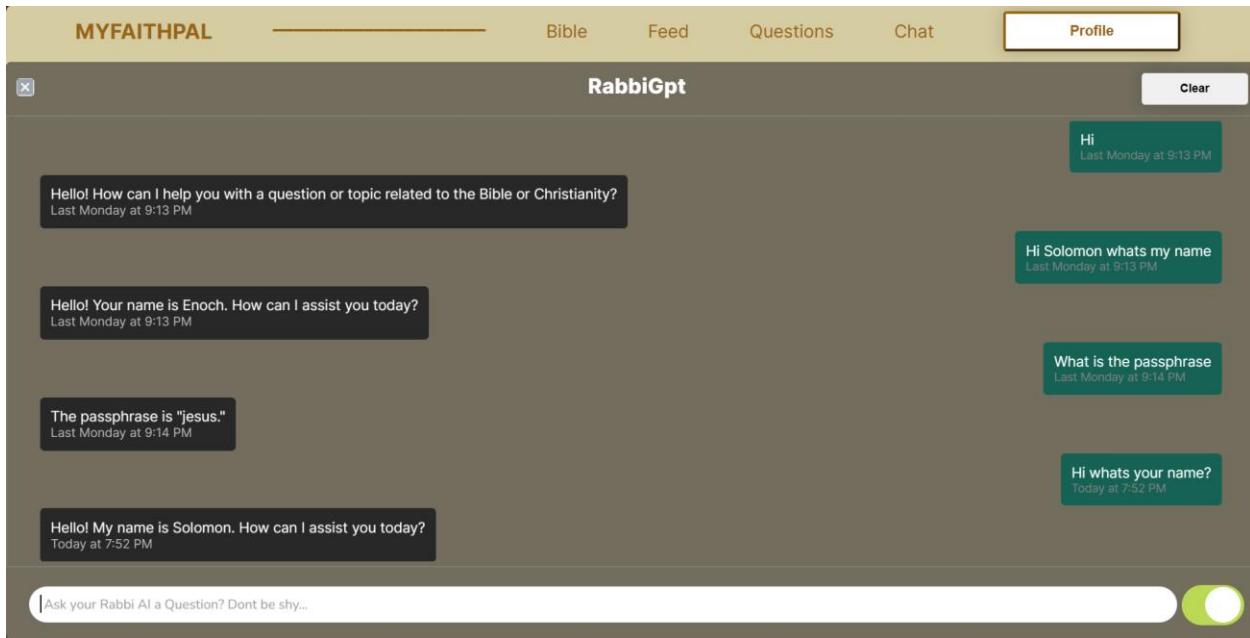


Figure 48, Speaking to FaithPal augmentation of ChatGPT “Solomon”.

Faithpal has two different AI working in the background to answer questions of users and have conversations even just as a friend. The two AI can be differentiated differently as one was implemented directly through OpenAI [14] node [18] modules and then the other through a Framework called LangChain [13] that deals with different language modules. First I'll go through the OpenAI implementation as that is simpler compared to the LangChain [13] implementation.

```

routes > chatgpt.js
1 import dotenv from 'dotenv';
2 import express from 'express';
3 import OpenAI from "openai";
4 import { connectMongoDB } from '../lib/mongo.js';
5 import Response from './models/messages.js';
6
7 const router = express.Router();
8 const openai = new OpenAI();
9
10 dotenv.config();
11
12 router.get('/', (req, res) => {
13   res.send('Hello, this is the AI Route make sure your key is in the .env');
14 })
15
16
chatgpt.ts <
routes > chatgpt.ts > ...
12   router.get('/', (req, res) => {
13   })
14
15
16   router.post('/create', async (req, res) => {
17
18     const { question, aichatroom } = req.body;
19
20     const prompt = `Answer this question only if it's related to the Bible or Christianity and in a Christian or Bible context: ${question}`;
21
22     try {
23       const response = await openai.chat.completions.create({
24         model: "gpt-3.5-turbo",
25         messages: [{ role: "user", content: prompt }],
26         max_tokens: 1000,
27       });
28
29       const id_AI = process.env.AI_ID;
30
31       await connectMongoDB();
32
33       const answer = response.choices[0];
34
35       //console.log('ChatGPT Response:', answer.message.content);
36
37       const aiReponse = await Response.create({ aichatroom, user: id_AI, text: answer.message.content, data: answer, prompt });
38
39       res.status(201).json({ response: aiReponse });
40     } catch (error) {
41       console.error(`Error calling OpenAI ChatGPT: ${error}`);
42       res.status(500).json({ error: 'Internal Server Error' });
43     }
44   }
45 }

```

Figure 50

In Figure 50, we have the full implementation of the OpenAI [14] we pass a question and id of the chatroom with the user we want to prompt the AI. Then we pass in a String that we add to the question the restrict the user questions to only that of Christianity and the Bible. Then we use functions from the OpenAI [14] module to access GPT models to answer the prompt and we store and pick the first choice of the multiple answers we may be given and return it in the response. It is important to note that the requests aren't linked so prompting this AI is like starting a new conversation each time.

For LangChain [13] it is the same process when it comes to prompting but some extra functionalities

```

1 router.post('/input', async (req, res) => {
2   const { input, aichatroom } = req.body;
3
4   try {
5     const id_AI = process.env.AI_ID;
6     const client = new MongoClient(`${process.env.MONGODB_URI}`);
7     connectMongoDB();
8
9     // Prompt Template
10    const prompt = ChatPromptTemplate.fromMessages([
11      {"ai": "You are a helpful friend and assistant and your name is Solomon, Answer questions only related to the Bible or Christianity and Answer referring to Bible and Christianity", "type": "ai"}, {"human": "(input)", "type": "human"}, {"ai": "new MessagesPlaceholder(\"chat_history\"),\n        \"(input)\")", "type": "ai"}, {"human": "(agent_scratchpad)", "type": "human"}, {"ai": "{tool-output}";{{#if hasRetrieverTool}} **Use the {{retrieverTool}} tool to enhance my response.** {{retrieverTool.output}} {{/if}}", "type": "ai"}, {"human": "(input)", "type": "human"}]);
12
13    const searchTool = new TavilySearchResults();
14
15    const chatHistory = await ChatHistory.findOne({ aichatroom });
16    const storeChatHistory = [];
17
18    if (chatHistory != null) {
19      chatHistory.messages.forEach((message, index) => {
20        if (index % 2 === 0) {
21          storeChatHistory.push(new HumanMessage(message.content));
22        } else {
23          storeChatHistory.push(new AIMessage(message.content));
24        }
25      });
26      //console.log(storeChatHistory);
27    } else {
28      const check = await ChatHistory.create({ messages: storeChatHistory, aichatroom });
29      //console.log(check);
30    }
31
32    const vectorStore = await searchVectorStore(client, input);
33
34    const retriever = await vectorStore.asRetriever({
35      k: 3,
36      vectorStore,
37      verbose: true
38    });
39
40    const retrieverTool = createRetrieverTool(retriever, {
41      name: "retriever",
42      description: "use this tool",
43      query: "{input}"
44    });
45
46    const tools = [retrieverTool, searchTool];
47
48    const agent = await createOpenAIFunctionsAgent({
49      llm: model,
50      prompt,
51      tools,
52    });
53
54    // Create the executor
55    const agentExecutor = new AgentExecutor({
56      agent,
57      tools,
58    });
59
60    // Call Agent
61    const response = await agentExecutor.invoke({
62      input: input,
63      chat_history: storeChatHistory,
64    });
65
66    //console.log("Agent: ", response.output);
67    storeChatHistory.push(new HumanMessage(input));
68    storeChatHistory.push(new AIMessage(response.output));
69    const updatedHistory = await ChatHistory.findOneAndUpdate({ aichatroom }, { $set: { messages: storeChatHistory } });
70    const displayChat = await Response.create({ aichatroom, user: id_AI, text: response.output, data: response, prompt: input });
71
72    res.status(200).json({ response: displayChat, message: `Ai: ${response.output}` });
73
74  } catch (err) {
75    console.error(err);
76  }
77}

```

Figure 51

In figure 51, we have our prompt method just like in figure 50 we have an aichatroom variable passed in the request body with an input from the user. This time around though because we are using LangChain [13] we have a more structural way of Prompt Engineering and add a description as to what we expect from the model.

In this function we are creating Chat history in our MongoDB [17] database that keeps record of the messages of the Chat given our AI conversation awareness and memory also we have Tools that utilise by the Agent function to perform retrieval to get extra information from the database and web search.

If you inspect Figure 51 you will see that on each request, we are searching through something called a Vector Store [16]. I mentioned previously that we collect information from the MongoDB [17] to feed the Model. This is only possible through vector embeddings; Models only understand numbers, so we convert the text into embeddings, so the Model can make sense of it.

The screenshot shows a MongoDB interface with the following details:

- CONNECTIONS:** mongo, FaithPal (selected), mongo, admin, local, test, Online Shop mysql.
- COLLECTIONS:** (10 collections listed)
- Search results:**
 - 1 ObjectID("66267240447eebd478705b38") What is the difference between iniquity, sin, and transgression? JAnswer,In Psalm 32:5, the psalmist says, "I acknowledged my sin to you and did not cover up my iniquity." J536 items
 - 2 ObjectID("66267240447eebd478705b39") What does the Bible say about empathy? JAnswer,The Scriptures refer to the quality of empathy, which we see demonstrated in several biblical narratives. Empathy is t1 J536 items
 - 3 ObjectID("66267240447eebd478705b3a") Is physical abuse an acceptable reason for divorce? JAnswer,The Bible is silent on the issue of physical abuse as a reason for divorce, although it is obvious what God expects. J536 items
 - 4 ObjectID("66267240447eebd478705b3b") What does the Bible say about feminism? JAnswer,The modern concept of feminism was not present during the time that the Bible was written, but that does not mean that it is not mentioned. J536 items
 - 5 ObjectID("66267240447eebd478705b3c") What does it mean that Jesus is the Son of God? JAnswer,Jesus is not God's Son in the sense of a human father and a son. God did not get married and have a son. God is omnipotent. J536 items
 - 6 ObjectID("66267240447eebd478705b3d") What are some Bible verses about widows? JAnswer,Exodus 22:22-24. You shall not mistreat any widow or fatherless child. If you do mistreat them, and they cry out to me, I will hear their cry. J536 items
 - 7 ObjectID("66267240447eebd478705b3e") How could there be light on the first day of Creation if the sun was not created until the fourth day? JAnswer,The question of how there could be light on the first day of Creation is a common one. J536 items
 - 8 ObjectID("66267240447eebd478705b3f") When were Obadiah's prophecies against Edom fulfilled? Obadiah 1:18-20? JAnswer,A major factor in determining when Obadiah's prophecies against Edom were fulfilled is the timing of the fulfillment. J536 items
 - 9 ObjectID("66267240447eebd478705b40") What is a blessing according to the Bible? JAnswer,A blessing, according to Merriam Webster's Collegiate Dictionary, is "the act or words of one that blesses" or "a thing that brings good fortune or happiness." J536 items
 - 10 ObjectID("66267240447eebd478705b41") What are some Bible verses about anger? JAnswer,Psalms 37:8, "Refrain from anger and forsake wrath! Fret not yourself; it leads only to evil." Proverbs 14:29, "Whoever is angry with his neighbor must bear the punishment of his sins." J536 items
 - 11 ObjectID("66267240447eebd478705b42") How do I know which of God's promises are for me? JAnswer,There are literally hundreds of God's promises in the Bible. How can we know which promises apply to us? J536 items
 - 12 ObjectID("66267240447eebd478705b43") What was Sandemanianism? JAnswer,Sandemanianism was a sect within Christianity with a flawed doctrine about faith. Its teaching started in about 1730 with the Scotti. J536 items
 - 13 ObjectID("66267240447eebd478705b45") What does the Bible say about clothing? JAnswer,Clothing has played a major role in the history of God's interactions with humanity and is featured prominently from Genesis to Revelation. J536 items
 - 14 ObjectID("66267240447eebd478705b46") Was the Lakeland, Florida, revival led by Todd Bentley of Fire Ministries a true biblical revival? JAnswer,The Lakeland Revival of 2006 in Lakeland, Florida, sometimes known as the "Todd Bentley Revival," was a Christian evangelistic crusade held at the Lakeland Assembly of God. It was led by Todd Bentley, a well-known Christian speaker and author. J536 items
 - 15 ObjectID("66267240447eebd478705b47") Is it true that life and death are in the power of the tongue? JAnswer,The tongue is used throughout Scripture in both literal and metaphorical ways, especially in Psalms. J536 items
 - 16 ObjectID("66267240447eebd478705b47") How are justification, sanctification, and glorification related? JAnswer,Every Christian recognizes that there are important stages in the Christian life. Just as we are physical beings, we are spiritual beings. J536 items
 - 17 ObjectID("66267240447eebd478705b49") What makes Christianity unique? JAnswer,Christianity truly shines, or it is just one of many roads on the path to truth. J536 items
 - 18 ObjectID("66267240447eebd478705b49") What are contentious words in Catholicism? JAnswer,It is often said by evangelicals that the Catholic Church teaches "salvation by works." Most Catholics would deny this. J536 items
 - 19 ObjectID("66267240447eebd478705b4a") What are contentious words in Christianity? JAnswer,It is often said by evangelicals that the Catholic Church teaches "salvation by works." Most Catholics would deny this. J536 items
 - 20 ObjectID("66267240447eebd478705b4b") Christianity do we have the person of Christ who claimed to be God, performed many miracles to prove His claim of divinity, died and rose from the dead, and claimed to be God? J536 items
 - 21 ObjectID("66267240447eebd478705b4c") Who is the seed of the woman? JAnswer,Genesis 3:15, "I will put enmity between you and the woman, and between your offspring and hers; he will crush your head, and you will strike his heel." J536 items
 - 22 ObjectID("66267240447eebd478705b4d") What are some Bible verses about understanding? JAnswer,Proverbs 18:22, "A fool takes no pleasure in understanding, but only in expressing his opinion." Psalm 119:130, "Your word is a lamp to my feet and a light to my path." J536 items
 - 23 ObjectID("66267240447eebd478705b4e") Who is the seed of the woman? JAnswer,Genesis 3:15, "I will put enmity between you and the woman, and between your offspring and hers; he will crush your head, and you will strike his heel." J536 items
 - 24 ObjectID("66267240447eebd478705b4f") What did the crowds shout, "Crucify him!" when Pilate wanted to release Jesus? JAnswer,When the Sanhedrin brought Jesus before the Roman governor, Pontius Pilate, he asked, "What is your charge against this man?" J536 items
 - 25 ObjectID("66267240447eebd478705b50") Was Adam with Eve when she spoke to the serpent (Genesis 3:1)? JAnswer,The Bible notes that Eve was the first to eat the fruit from the tree after being deceived by the serpent. J536 items
 - 26 ObjectID("66267240447eebd478705b51") Who was Amminadab in the Bible? JAnswer,The Bible speaks of three men named Amminadab. None of the three have much information about them provided in Scripture. J536 items
 - 27 ObjectID("66267240447eebd478705b52") Why is "the devil made me do it" not a valid excuse? JAnswer,The first instance of "the devil made me do it" used as an excuse was in the Garden of Eden. Eve says, "The devil made me do it." J536 items
 - 28 ObjectID("66267240447eebd478705b53") Why did Reuben, Gad, and the half-tribe of Manasseh want to live on the east of the Jordan? JAnswer,The tribes were poised to enter Canaan. Before they crossed the Jordan River, they were told to leave their families behind. J536 items
 - 29 ObjectID("66267240447eebd478705b54") What are some Bible verses about budgeting? JAnswer,Proverbs 3:9, "Honor the Lord with your wealth and with the firstfruits of all your produce." Luke 14:28-30, "If you give to the poor, you will be rich." J536 items
 - 30 ObjectID("66267240447eebd478705b55") What is the paschal mystery? JAnswer,The word paschal comes from the Hebrew word for Passover—the event in Exodus where the death angel passed over all the homes in Egypt. J536 items
 - 31 ObjectID("66267240447eebd478705b56") What does the Bible say about underage drinking? JAnswer,The Bible never refers to underage drinking specifically. In Bible times, wine was the beverage of choice. Part of the reason for this is that it was a safe drink. J536 items
 - 32 ObjectID("66267240447eebd478705b57") What is miracle money, and is it biblical to seek it? JAnswer,Miracle money is money that miraculously appears in someone's purse, wallet, pocket, or bank account to de... J536 items
 - 33 ObjectID("66267240447eebd478705b58") What are some Bible verses about trust? JAnswer,Proverbs 3:5, "Trust in the Lord with all your heart, and do not lean on your own understanding." Psalm 56:3-4, "When I... J536 items
 - 34 ObjectID("66267240447eebd478705b59") Who was George Mueller? JAnswer,George Mueller (1805–1890), whose full name was Johann George Ferdinand Mueller, was a preacher, evangelist, and prominent m... J536 items
 - 35 ObjectID("66267240447eebd478705b5a") Did Paul ever meet Jesus in person? JAnswer,The subject of Jesus Christ and His saving work are at the forefront of the apostle Paul's ministry. "Woe to me if I do not p... J536 items

Figure 52, embeddings, and information web scraped from GotQuestions.org.

MongoDB [17] supports the creation of vector embeddings and stores them in parallel to your normal data which was perfect for a seamless integration of my web scraped data and my embeddings so I didn't have to use two different databases and can view all in the same place.

```

import { MongoDBAtlasVectorSearch } from "@Langchain/mongodb";
import { OpenAIEmbeddings } from "@Langchain/openai";
import { JSONLoader } from "Langchain/document_loaders/fs/json";
import { RecursiveCharacterTextSplitter } from 'Langchain/text_splitter';

//Loading our data and making vector store
export const createVectorStore = async (client) => {

    //Delete the collection to create a new vector store from json
    const namespace = process.env.COLLECTION_NAMESPACE;
    const [dbName, collectionName] = namespace.split(".");
    const collection = client.db(dbName).collection(collectionName);

    /* If you want to load directly from database
    //const response = await axios.post(`${process.env.SERVER_URL}/faithpalAI/Load/json`);

    //console.log(response.data)

    //const context = new Blob([JSON.stringify(response.data)], { type: 'application/json' });

    //const loader = new JSONLoader(context);*/

    //WARNING RUN JSON FIX BEFORE CREATING VECTOR

    await collection.deleteMany({});

    const loader = new JSONLoader("question.json");

    const docs = await loader.load();

    const splitter = new RecursiveCharacterTextSplitter({
        chunkSize: 5000, chunkOverlap: 0, separators: ["\n\n\n\n"], 
    })

    const splitDocs = await splitter.splitDocuments(docs);

    const embeddings = new OpenAIEmbeddings();

    const vectorStore = await MongoDBAtlasVectorSearch.fromDocuments(
        splitDocs,
        embeddings,
        {
            collection,
            indexName: process.env.INDEX_NAME,
        }
    )

    return vectorStore;
}

```

Figure 53, Creates Vector Store on MongoDB [17] using OpenAIEmbeddings as we are using OpenAI [14].

In Figure 53, we are using Json loader to load all the questions and answer into a splitter that splits the text into smaller documents and then embeds them and creates a MongoDB Vector Search index [16]. Vector Searches [16] are done by similarity to the prompt so the closes to the question gets pulled out and fed to the AI and then the AI makes the decision whether to use this information or not based on if it helps enhance its response as seen in the **ChatPromptTemplate**

it is important that to note that his process is known as Augmented Retrieval where data is sourced external to a Model to add to its Knowledge.



```

import axios from 'axios';
import dotenv from 'dotenv';
import express from 'express';

//Webscraping
import * as cheerio from 'cheerio';

//Data
import fs from 'fs';
import { connectMongoDB } from '../lib/AI/mongo.js';
import sourceKnowledge from '../models/aiSourceKnowledge.js';

//AI LangChain
import { TavilySearchResults } from "@langchain/community/tools/tavily_search";
import { AIMessage, HumanMessage } from '@Langchain/core/messages';
import { ChatPromptTemplate, MessagesPlaceholder } from "@Langchain/core/prompts";
import { AgentExecutor, createOpenAIFunctionsAgent } from "Langchain/agents";
import { createRetrieverTool } from "langchain/tools/retriever";
import { MongoClient } from "mongodb";
import { createVectorStore } from '../lib/AI/createVectorMongoDB.js';
import { searchVectorStore } from '../lib/AI/searchVectorMongoDB.js';
import AISKnowledge from '../models/aiSourceKnowledge.js';
import ChatHistory from "../models/llmHistory.js";
import Response from '../models/messages.js';
import { model } from '../util/model.js';

```

Figure 54, Imports for FaithPal customized AI.

Here are just the various imports needed for Solomon FaithPal AI LangChain [13] provides an expansive library of customizability to make suitable AI for web applications.

```

router.post('/webscrape', async (req, res) => {
  try {
    // Fetched the HTML of the page I want to scrape
    const { data } = await axios.get(process.env.SCRAPING_URL);

    // Load the HTML I fetched in the previous line
    const $ = cheerio.load(data);

    // Select all the List items in plainList class
    const listItems = $(".content");

    let text;

    connectMongoDB();

    // Iterate through each List item and extract text
    listItems.each((idx, el) => {
      text = $(el).children("h1").children("[itemprop='name headline']").text() + $(el).children("[itemprop='articleBody']").text() + "\n\n\n\n";
    });

    const checkData = await sourceKnowledge.create({ text: text });

    res.status(201).json({ response: checkData, message: "Scraping completed successfully!" });

  } catch (err) {
    console.error(err);
    res.status(500).send("Internal Server Error");
  }
});

```

Figure 55, Web scraping from GotQuestions.org.

```

51
52 router.post('/fix/json', async (req, res) => {
53   try {
54     connectMongoDB();
55
56     let FormattedData = [];
57
58     const loadData = await AISKnowledge.find();
59
60     //console.log(loadData);
61
62     loadData.forEach(obj => FormattedData.push({ text: obj.text }));
63
64     //console.log(FormattedData);
65
66     await fs.promises.writeFile("question.json", JSON.stringify(FormattedData, null, 2));
67
68     res.status(200).json({ response: FormattedData, message: "Json fixed successfully!" });
69
70   } catch (err) {
71
72     console.error(err);
73     res.status(500).send("Internal Server Error");
74
75   }
76 }
);

```

Figure 56, Web scraping from GotQuestions.org.

In this post request method, we are using a node [18] module called **Cheerio** [25] that allows us to load HTML of pages we have made requests to and then using jQuery we can iterate through the pages elements and pick out the elements we want to take after doing this we store it our data base and then we have another http method that loads the data from MongoDB [17] and stores it in a Json file to be used in the createVectorStore [16] operation. Look at figure 56 as reference.

```

2  {
3    "text": "What does the Bible say about empathy?\n\nAnswer\nThe Scriptures refer to the quality of empathy, which we see demonstrated in several biblical narratives. Empathy is the capacity to fe
4  },
5  {
6    "text": "How do I know which of God's promises are for me?\n\nAnswer\nThere are literally hundreds of God's promises in the Bible. How can we know which promises apply to us, which promises we c
7  },
8  {
9    "text": "Is physical abuse an acceptable reason for divorce?\nAnswer\nThe Bible is silent on the issue of physical abuse as a reason for divorce, although it is obvious what God expects a marria
10 },
11 {
12   "text": "What does the Bible say about feminism?\nAnswer\nThe modern concept of feminism was not present during the time that the Bible was written, but that does not mean the Bible has nothing
13 },
14 {
15   "text": "How could there be light on the first day of Creation if the sun was not created until the fourth day?\nAnswer\nThe question of how there could be light on the first day of Creation whe
16 },
17 {
18   "text": "When were Obadiah's prophecies against Edom fulfilled (Obadiah 1:18-20)?\nAnswer\nA major factor in determining when Obadiah's prophecies against Edom were fulfilled rests on when the b
19 },
20 {
21   "text": "What does it mean that Jesus is the Son of God?\nAnswer\nJesus is not God's Son in the sense of a human father and a son. God did not get married and have a son. God did not mate with M
22 },
23 {
24   "text": "What are some Bible verses about anger?\nAnswer\nPsalm 37:8\nRefrain from anger, and forsake wrath! Fret not yourself; it tends only to evil.\nProverbs 14:29\nWhoever is slow to anger h
25 },
26 {
27   "text": "What was Sandemanianism?\nAnswer\nSandemanianism was a sect within Christianity with a flawed doctrine about faith. Its teaching started in about 1730 with the Scottish Presbyterian min
28 },
29 {
30   "text": "What is a blessing according to the Bible?\nAnswer\nA blessing, according to Merriam-Webster's Collegiate Dictionary, is "the act or words of one that blesses," or "a thing conducive to
31 },
32 {
33   "text": "What are some Bible verses about widows?\nAnswer\nExodus 22:22-24\nYou shall not mistreat any widow or fatherless child. If you do mistreat them, and they cry out to me, I will surely h
34 },
35 {
36   "text": "What does the Bible say about clothing?\nAnswer\nClothing has played a major role in the history of God's interactions with humanity and is featured prominently from Genesis (3:7) to Re
37 },

```

Figure 57, Example data of questions and answers web scraped from GotQuestions.org.

10.7 Profile Page

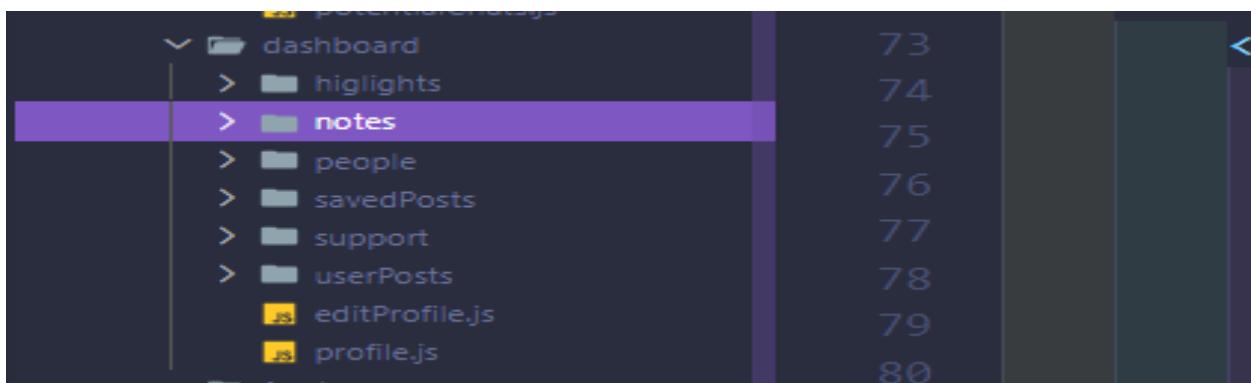


Figure 58, Profile, and its collection of components.

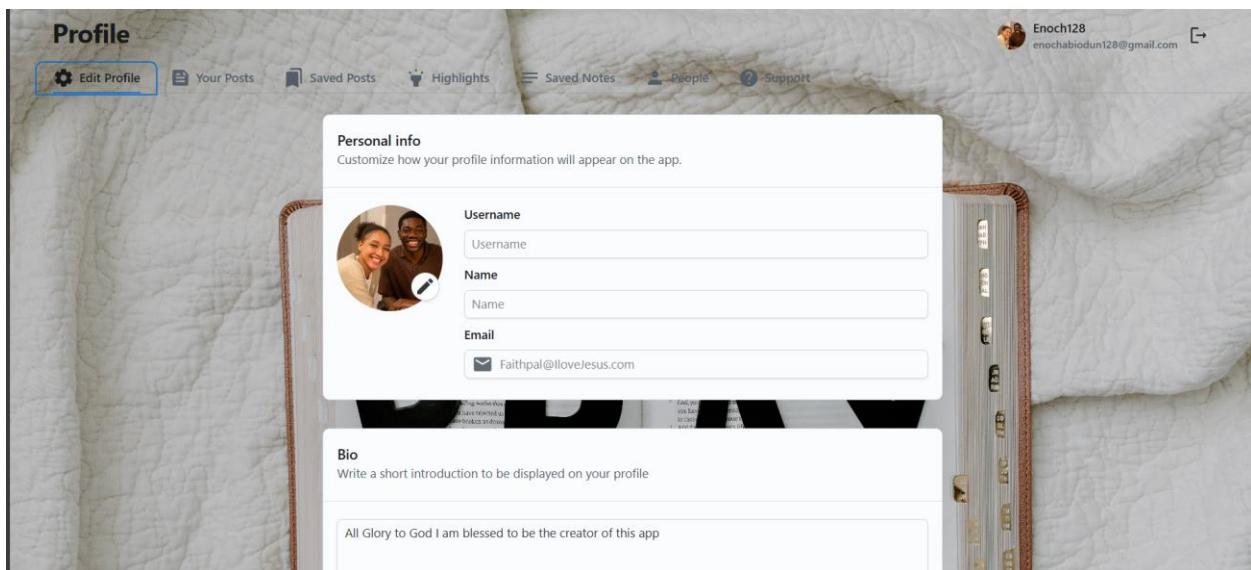


Figure 59, Profile Page

The profile page is like a collection of the rest of the pages incorporating communication to S3 [19] Buckets so users can update their profile, view their posts , saved posts, saved highlights, saved notes from the bible page, and go to the people tab to find other users on the app and then beside their name in Figure 59 sign out by clicking the icon.

```

function findSpans(obj, parentObj = null, parentKey = null) {
  if (typeof obj === 'object' && obj !== null) {
    //Getting Reference once we find it
    if (obj.reference) {
      reference = obj.reference
    }

    if (obj?.children?.[1]?.props?.children?.[0]?.props?.children?.[1]?.props?.children?.[0]?.props?.children?.[1]?.props?.className?.includes("highlight")) {
      brokenverse = obj?.children?.[0]?.props?.children?.[0];
    }

    if (obj?.props?.children?.[0]?.props?.children?.[1]?.props?.className?.includes("highlight")) {
      if (parentObj && parentKey !== null) {
        verse = parentObj?.[0]?.props?.children?.[0];
      }
    }

    if (obj?.props?.children?.[0]?.props?.children?.[1]?.props?.className?.includes("comment")) {
      if (parentObj && parentKey !== null) {

        let child = parentObj?.[1]?.props?.children?.[0].props?.children?.[1]?.props?.children?.[0];
        const comment = parentObj?.[1]?.props?.children?.[0].props?.children?.[1]?.props?.children?.[1]?.props?.children?.[1];
        let previousVerse = verse;
        verse = parentObj?.[0]?.props?.children?.[0];

        if (!verse) {
          verse = previousVerse;
        } else if (!previousVerse) {
          verse = brokenverse;
        }
      }
    }

    if (typeof child === 'object') {
      while (typeof child === 'object') {
        child = child?.props?.children?.[1]?.props?.children?.[0];
      }
    }

    if (!NotesArray.some(item => item?.note === child)) {
      //console.log("Found note: ", parentObj);
      NotesArray.push({ note: child, ref: reference, verse, comment });
      //console.log("Found note array: ", NotesArray);
    }
  } else if (Array.isArray(obj)) {
    for (let i = 0; i < obj.length; i++) {
      findSpans(obj[i], obj, i);
    }
  } else {
    for (let key in obj) {
      if (Object.prototype.hasOwnProperty.call(obj, key)) {
        // Pass the current object and its ancestors as arguments
        findSpans(obj[key], obj, key, parentObj, parentKey);
      }
    }
  }
}

```

Figure 60, Recursively traversing the dom object to get the changed verse (highlight or comment)

One of the challenges though was displaying the certain verses that were edited by the user by highlights or a comment since we save the whole passage as a Json stringified object when we stored it in MongoDB [17] we can only pull the whole object so with the method in Figure 60 we recursively search until we hit one of the if statements. This iterates from the top level of the object to the lowest level of the object until it finds a match for example if we focus on the third if statement of the findSpan function we can see the includes function is looking for a **className** with **highlight** in it and once it find it then we use the **parentObj** or **obj** itself as reference when we want to grab the information that we need.

11 Ethics

Ethics was detrimental part of my project. In the end of my project, I decided not to settle in just hosting my application on the internet without adding any security measures to defend against attackers. In my code we are using environment variables to protect URLs that could give access to sensitive information. But going further I have taken the obligation to secure the website and its servers with an SSL certificate which is a digital certificate that authenticates FaithPal identity and enables an encrypted connection. This ultimately creates an encrypted link between a web server and a web browser. As Faithpal is a user-friendly web application, you should not have to worry about attackers still any sort of information about you. On the first push of FaithPal without ssl it was one of the heavily noticed highlights of Faithpal that it wasn't secure. Finally, when the padlock icon was finally free from that red x users felt at ease visiting the website even my brother was waiting until the web application was secure to try it out.

Furthermore, I took the effort to write a support page for my users that can be found as one of the tabs on the Profile page. On this tab there are rules to the application because the application when it came to the Feed page and Chat users have freedom to write and post and communicate with people however way they chose but to promote sensible use and respectful use of the application I wrote some rules promoting positivity and maturity across the Faithpal in hope people do otherwise Reporting can be the consequence of misbehaviour.

Overall, Faithpal was made in reverence to God, and he is love and that's what this app tries to teach people and show people.

12 Conclusion

To conclude, All the expectations for Faithpal have been met and I believe not have they only been met far exceeded in a shorter period suggest at the beginning of the College calendar year. With a fully functional and deployed web application that is currently being used by the public I couldn't be more astonished and gladder that this was the project that I felt led to do by God.

In working towards the completion of this Faithpal I have gained many new languages and experience with technologies such as Next JS [6], React [5], Express JS [8], Node JS [7], LangChain [13] and AWS cloud technologies that I believe going forward will be beneficial for my goals to become a competent Web developer and for some of my own personal projects in the future. I have endeavoured into the whole world of Web development dabbling on the web development processes such as frontend development, backend development, databases, cloud storage and hosting web services creating a great foundation for me to grow as a developer. Not only did I grow as a developer, but I believe ultimately I have become more mature as my organisational skills has reached its highest peak. It is true that this project has huge potential, so I plan to continue the working on Faithpal past my time at Atlantic Technological University.

Links:

GitHub Repository: [G00380316/MyFaithPal-FYP \(github.com\)](https://github.com/G00380316/MyFaithPal-FYP)

FaithPal: [FaithPal \(myfaithpal.com\)](https://myfaithpal.com)

FaithPal Server API: faithpalserver.online

FaithPal AI Server API: faithpalaiserver.online

FaithPal Bible API: [faithpal-bible-api \(divineword.site\)](https://faithpal-bible-api.divineword.site)

13 References

- [1] "YouVersion Bible App," [Online and Mobile App]. Available: <https://www.bible.com/>
- [2] "Instagram," [Online]. Available: <https://www.instagram.com>
- [3] "Facebook," [Online]. Available: <https://www.facebook.com>
- [4] "Jira," [Online]. Available: <https://www.atlassian.com/software/jira>
- [5] "React JS" [Online]. Available: [React Reference Overview – React](#)
- [6] "Next JS" [Online]. Available: [Docs | Next.js \(nextjs.org\)](#)
- [7] "Node JS" [Online]. Available: [Index | Node.js v22.0.0 Documentation \(nodejs.org\)](#)
- [8] "Express JS" [Online]. Available: <https://expressjs.com/>
- [9] "MUI Joy" [Online]. Available: [Overview - Joy UI \(mui.com\)](#)
- [10] "MUI" [Online]. Available: [Overview - Material UI \(mui.com\)](#)
- [11] "Socket.io" [Online]. Available: [Introduction | Socket.IO](#)
- [12] "LangChain JS API References" [Online]. Available: [LangChain.js - v0.1.36](#)
- [13] "Langchain JS" [Online]. Available: [Introduction | Langchain](#)
- [14] "OpenAI" [Online]. Available: [Quickstart tutorial - OpenAI API](#)
- [15] "W3school MongoDB" [Online]. Available: [Node.js MongoDB Get Started \(w3schools.com\)](#)
- [16] "MongoDB VectorSearch" [Online]. Available: [MongoDB Atlas Vector Search | MongoDB](#)
- [17] "MongoDB" [Online]. Available: [MongoDB Node Driver - Node.js Driver v6.5](#)
- [18] "Node Package Manager" [Online]. Available: [npm | Home \(npmjs.com\)](#)

- [19] “Amazon S3 Buckets” [Online]. Available: [AWS SDK for JavaScript - Developer Guide for SDK v2 \(amazon.com\)](#)
- [20] “Amazon EC2” [Online]. Available: [Amazon Elastic Compute Cloud Documentation](#)
- [21] “PM2” [Online]. Available: [PM2 - Quick Start \(keymetrics.io\)](#)
- [22] “Porkbun” Domain Name Provider[Online]. Available: [porkbun.com | An oddly satisfying experience.](#)
- [23] “Nginx Conf file” [Online]. Available:
<https://www.nginx.com/resources/wiki/start/topics/examples/full/>
- [24] “Let's Encrypt” SSL Certificate [Online]. Available: [Getting Started - Let's Encrypt \(letsencrypt.org\)](#)
- [25] “React js Editor” [Online]. Available: [Welcome to Cheerio! | cheerio](#)
- [26] “Semantic Search” [Online]. Available:
<https://youtu.be/ZvwUzcMvKil?si=9A6BfsPX1Bbygq25>
- [27] “Retrieval Idea” [Online]. Available:
https://youtu.be/9AXP7tCI9PI?si=xIEvB_zlebNkxqHH
- [28] “ReactEditor” Tutorial [Online]. Available:
<https://youtu.be/iR4xtbGtoU0?si=NHZqHZ4Hs4-57SPx>
- [29] “Google API” [Online]. Available: <https://console.cloud.google.com/apis>
- [30] “ReactJS Editor” [Online]. Available: [reactjs-editor - npm \(npmjs.com\)](#)