

UNIVERSITY OF LONDON

INTERNATIONAL PROGRAMMES

BSc Computer Science and Related Subjects



CM3070 PROJECT

FINAL PROJECT REPORT

Cooking-Chat (Project Idea Title 2:A
Collaboration web application)

Author: Daniel Miller
Student Number : 190429579
Date of Submission: 18/09/2023
Supervisor : Don Wee

Contents

CHAPTER 1:	INTRODUCTION	3
CHAPTER 2:	LITERATURE REVIEW	3
CHAPTER 3:	PROJECT DESIGN	4
CHAPTER 4:	IMPLEMENTATION	5
CHAPTER 5:	EVALUATION	5
CHAPTER 6:	CONCLUSION	5
CHAPTER 7:	APPENDICES	5
CHAPTER 8:	REFERENCES	6

CHAPTER 1: INTRODUCTION

The idea for the project came when I noticed that there were no easy all in one solutions to share recipes and ask for instruction from my father while chatting with him about cooking.

The motivation behind the project is to fill a gap where there's a lack of solutions available on app stores and computer programs or webapps. I feel that the attempt at implementing this app would be widely appreciated by like-minded individuals that wish to make cooking a collaborative creative experience.

My aims with this project is to learn how to produce a fullstack application that can be published and used by these like minded user. The question I have is what can be considered a collaborative tool? My objectives for this app is to include user registration and authentication, live messaging feature which includes the use of a database and websockets and message pinning which involves updating the message data.

CHAPTER 2:LITERATURE REVIEW

// insert relevant images

What is collaboration?

Collaboration is the action of working with someone to produce something. So collaboration takes 2 or more individuals to come together and produce something. This article talks about the use of online forums and messaging apps for collaboration for learning. They had a group test out the 2 methods and gave feedback based on their findings and experience using the 2 different methods. My focus on the article is that, "Specifically, using the online discussion forum resulted in more communication aimed at knowledge construction, while using the mobile instant-messaging app resulted in more social interactions. " Sun, Z. et al. (2017) A tale of two communication tools: Discussion-forum and mobile instant-messaging apps in collaborative learning, British Educational Research Association. Available at: <https://bera-journals.onlinelibrary.wiley.com/doi/full/10.1111/bjet.12571> (Accessed: 13 June 2023). This justifies my vision of using a messaging app as a base for collaboration as it should give the users a more social experience while sharing their recipes and cooking techniques due to the instant communicative nature of messaging apps and the fact that the messaging app domain itself promotes direct collaboration between the 2 individuals.

Some individuals use PKM(Personal Knowledge Management) systems, "A personal knowledge management (PKM) system is a tool that helps you collect, organize, and access your personal knowledge. This can be anything from articles you've read, websites you visit often, or ideas you want to remember for later. " Rupp, E.E. (2023) "How to create your personal knowledge management system, " ABLE Blog: Thoughts, Learnings and Experiences [Preprint]. Available at: <https://able.ac/blog/personal-knowledge-management-system/#:~:text=A%20personal%20knowledge%20management%20>. In the context of my research, saving instructions and

ingredients list of recipes which is why I would like to explore the use of PKM apps as a pseudo recipe book.

Critical analysis of similar products

Obsidian Note taking app

Features

- Note taking app that uses markdown.
- It is a simple and distraction free language.
- Customizable experience.
- Extensible with community plugins
- Uses WikiLinks to link relating notes and topics together which can help visualize or find links that would go unnoticed.

Sharpen your thinking (no date) Obsidian. Available at: <https://obsidian.md/> (Accessed: 14 June 2023).

Obsidian is a great note taking app. It has allowed myself to let go of remembering some thoughts and jot them down, which helps me focus on other daily tasks. It is also quite extensible with lots of functions and community plugins that work together flawlessly to create a tailored experience. I myself use it for school, self reflection, personal knowledge management which also includes making notes on recipes I wish to recreate. This allows me to include steps in the form of plain text as well as formatted checklist that can be displayed on a separate note by including a customizable hashtag keyword such as "#ingredients" . I then have another command that I can use to reset the checklist on the recipe instead of manually unchecking each one.

The main downside of this is, it emphasizes personal knowledge management, rather than collaborative. While collaboration is possible with work around methods to sync notes (Syncthing: a continuous file synchronization program or through their own subscription (2019) Syncthing. Available at: <https://syncthing.net/> (Accessed: 14 June 2023)), it does not distinguish between users. While this meets the needs of a great PKM system, not being able to distinguish who said what. Discussion on the article being written would

also have to be done externally which may hinder creative train of thoughts. This would justify my reason to explore using a messaging app for collaboration as that would allow easy tracking of who said what in their own discussions.

CookBook - The Recipe Manager & Planner

Now I would like to focus on some of the apps and services that are available to store and manage their recipes. Firstly we have CookBook.

Features

- Import from web
- Scan physical recipes
- create from scratch
- Sort with tagging, rating, category
- Shopping checklist

The recipe manager & planner app (no date) CookBook. Available at:

<https://thecookbookapp.com/> (Accessed: 13 June 2023).²

The app allows for 40 recipes to be stored and 5 OCR scans for the free version. The recipes are stored on the account meaning that it is synced across all devices. Issue with that is that internet is required to be synced and updated.

Ingredient checklist feature is reset upon leaving and revisiting the recipe. This solves the problem of manually resetting the ingredients checklist but might not be ideal if someone wishes to visit other recipes or closes out the app in between actions. The same goes for the steps feature. It is the same exact crossed out checklist.

The app overall meets the needs for an individual looking to collect and expand their recipe list, but lacks a collaborative feature which hinders discussion and creativity.

Paprika Recipe Manager for iOS, Mac, Android, and Windows

Features

- Cloud Sync
- Web importing
- Smart Groceries list: ingredients are sorted by shopping list, ingredients will be aggregated eg. 1 + 2 eggs = 3 eggs

Paprika (no date) Paprika Recipe Manager for iOS, Mac, Android, and Windows.

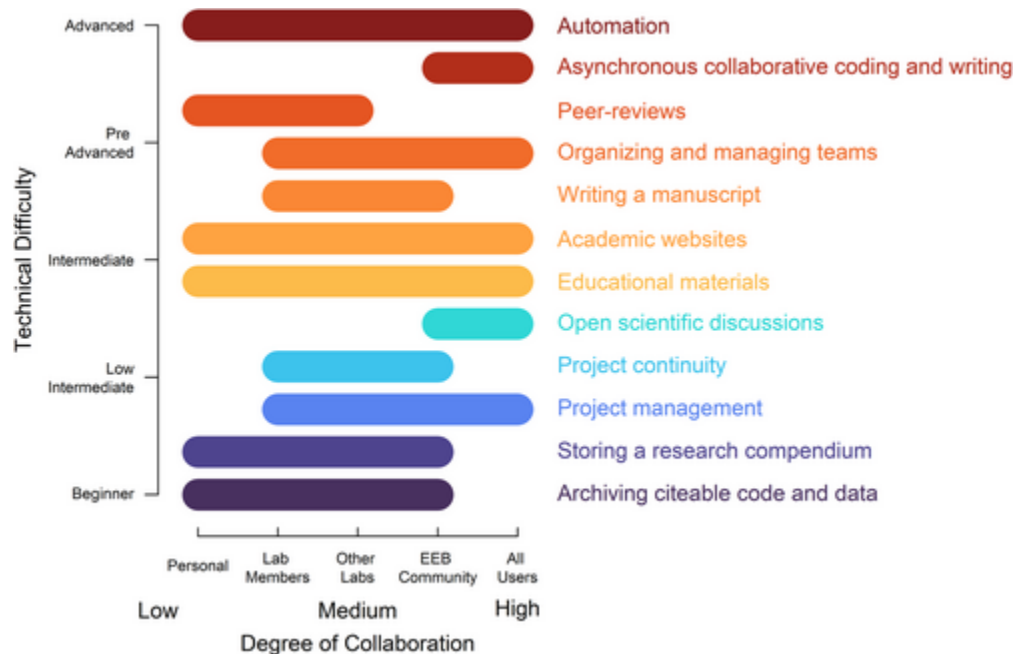
Available at: <https://www.paprikaapp.com/> (Accessed: 13 June 2023).

I won't go over Paprika in detail, as it seems to have a very similar set of features as CookBook app, which further proves my point that there is a lack of collaborative features amongst the most downloaded recipe apps.

Github: The Emergence of GitHub as a Collaborative Platform for Education | Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing

GitHub is a file hosting service with version control. It is typically used for software development but can also be used to host plain text files and other non-coding related domains.

While GitHub is great for file versioning and can be used for collaboration as users can download the remote repository, edit and create or remove files from the repository before pushing those changes back to the cloud for other users to sync up, the process of doing so requires quite a bit of knowledge on the subject to utilize the tool. My users are not expected to have that level of knowledge.



This image shows what I said to be true. It shows that Asynchronous collaborative coding and writing features in GitHub require one to be knowledgeable and familiar with the product or to at least be familiar with programming. The image was taken from this article that talks about the use of GitHub with Ecology and Evolutionary Biology in particular, and mentions that the industry has an increasing dependency on computational code to conduct research. I think this is relevant and similar to the environment and use case my app will be applying as the app is aiming to have written collaboration between users and they should not be required to understand how to use Git or similar collaborative tools with a high technical ceiling. Braga, P.H.P. et al. (2023) Not just for programmers: How GitHub can accelerate collaborative and reproducible research in ecology and evolution. Available at: <https://doi.org/10.1111/2041-210x.14108>.

Telegram

The messaging app I wish to focus on is called Telegram. The app has many features and ticks most boxes when it comes to instant messaging.

Features

- Available on all major platforms
- Simple and easy to use
- Messages are encrypted and can be set to self-destruct
- Synced across all devices
- Fast: delivers messages faster than any other application
- No limit on size of media or chats
- Open source and open API groups can hold up to 200,000 members
- very customizable

Telegram - A new era of messaging (no date) Telegram. Available at: <https://telegram.org/> (Accessed: 15 June 2023).

In addition to these features, Telegram also allows users to pin messages for quicker access to certain messages, for example maybe a house address, or in this context, a recipe/ingredients list. This is where I hope to theme and format the feature so make the

pinned messages look like recipe cards, that then expand to show the steps and methods of cooking and possibly also add a checklist feature within the expanded card

CHAPTER 3:PROJECT DESIGN

Overview

The project is a messaging app with home cooks, chefs and creatives looking to collaborate on cooking topics and recipes. The app will allow users to share and discuss recipes and cooking techniques amongst each other with instant messaging and a way to store and present recipes formatted in a way that feels familiar such as recipe cards.

Template being used: CM3035 Advanced Web Development

The template I am using is CM3035 Advanced Web Development: Project Idea 2: A Collaboration web application.

Domain and users

My users will range from regular everyday people to professional cooks or those that cook as a profession (including content creators and other creatives), and a decent understanding of technology and smart devices. They are people that use phones and computers daily to communicate, consume content and retain data and information.

The domain, or environment for the product, physically would be the kitchen, but not necessarily true. The actual domain for the product would actually be right next to the user, a tool the user can rely on instantly for their day to day life. With that in mind, the product is likely looking at covering Web domain as well as mobile domain so as to meet those requirements as a daily tool.

Design choices

Based on some of the research I have done in the Literature review as well as consideration for the user and domain the product will be used, some of the features I wish to include are:

- asynchronous messaging between at least 2 users
- recipe storing system
 - includes pinning feature
 - checklist feature
 - easily import recipes from web and scans
 - unit conversion and serving scaling
 - personal recipe vault and sharing vault
- simple to grasp and intuitive for less technically proficient user base
- available as mobile app and web app

Overall structure of the project

The overall structure of the project would be a mobile app on android as well as a web app on pc. These are the two platforms available to me, and the two I am most familiar with. The interface of the app will be similar to most popular chat apps.

Identification of most important technologies and methods

The application itself will likely be a fullstack app, and a typically techstack will likely be used, such as MERN or MEAN stacks.

It will be developed using one of the popular javascript frameworks. There is more research to be done on these frameworks to see which one is most suited for my application, but the list includes ReactJS, NextJS, Sveltekit and Angular as well as maybe some other more obscure frameworks. There are some pros and cons to using each of these and should be considered based on the applications needs.

For the backend, I would like to try and avoid using any paid services, but I will likely have to use the free tiers of these services to achieve my goals.

This includes:

- Firebase
- Supabase
- MongoDB
- Redis

Name:

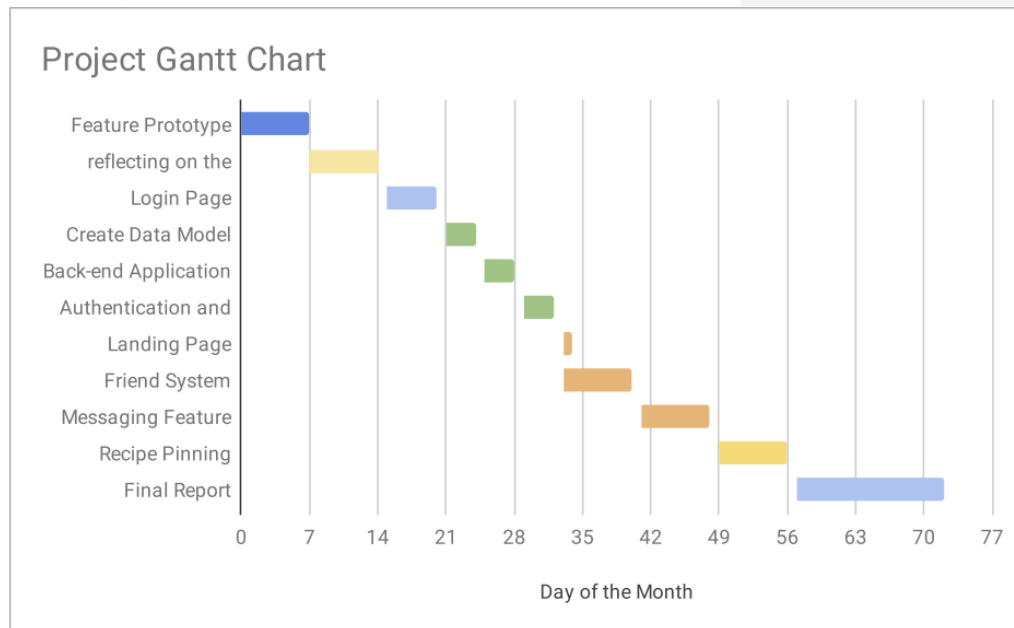
CM3070 Project

Student Number:

Preliminary Project Report

Work plan

TASK NAME	START DATE	END DATE	START ON DAY*	DURATION* (WORK DAYS)
Feature Prototype	7/7	7/14	0	7
reflecting on the evaluation of feature prototype	7/14	7/21	7	7
Login Page	7/22	7/27	15	5
Create Data Model and Database Schema	7/28	7/31	21	3
Back-end Application	8/1	8/4	25	3
Authentication and User creation	8/5	8/8	29	3
Landing Page	8/9	8/10	33	1
Friend System	8/9	8/16	33	7
Messaging Feature	8/17	8/24	41	7
Recipe Pinning Feature	8/25	9/1	49	7
Final Report	9/2	9/17	57	15



A plan for how you will test and evaluate your project.

Some of the libraries available for me to test my application.

- Unit tests: Jest or Mocha
- Integration tests: Cypress or TestCafe
- End-to-end tests: Puppeteer, Selenium or Playwright
- Snapshot testing: Jest snapshot tests
- Performance testing: Lighthouse or Google Analytics

The tests methods I believe are crucial for optimizing my project are unit tests, integration tests, end-to-end tests and snapshot tests.

CHAPTER 4:IMPLEMENTATION

What is my feature prototype?

For my feature prototype, I will be implementing a basic messaging app as I think that is the overarching idea for the project with additional features to fill in the market gap. At the end of the day, I am creating a collaborative tool through messaging, and if I am able to communicate with another webpage on the same network, it will be deemed a success at showing the most important feature for app.

Implementation

The prototype itself can be achieved through various ways, but I have chosen to go with the NextJS framework for these reasons.

Pros

- Server side rendering
- API routes out of the box making backend and frontend development simple
- automatic code-splitting, allowing for faster loading

Cons

- Steeper learning curve compared to vanilla ReactJS

Reasons I wanted to go with socket.io for feature prototype

For the feature prototype, I wanted to keep the application as simple as possible. The main goal for the feature prototype was to prove that a real-time chat app was within my scope. And I felt that to keep it simple and avoid any unnecessary complexity, I wanted to

avoid any user creation functions within the prototype and focus purely on the asynchronous messaging between 2 or more clients.

Socket.io is also a popular module for small projects to implement chat systems or webgames between 2 or more people through the use of websockets, and that is also why I will be using it.

Websockets cannot be used with the new default App routes API in NextJS

NextJS new default App routes API doesn't allow for stateful connections. This is because for serverless architectures, a function is only run for a short period of time and not continuously, so a constant connection through websockets is not possible.

"Unfortunately, Next.js doesn't support websocket servers

As Next.js backend server is running on Serverless environment by default. The core of Serverless is to host your code on 3rd party providers, same for the websocket servers: Use 3rd party real-time messaging services such as Ably or Pusher Channels, their services are more scalable, and suitable for Serverless environments

Vercel has a [guide](#) about the this, you can read this for further information"

- a discussion on reddit

https://www.reddit.com/r/nextjs/comments/13360t3/comment/jiaicln/?utm_source=share&utm_medium=web2x&context=3

Possible solutions

- In-memory management: Redis
- expressJS on another server for page routing

"It's important to note that serverless functions on Vercel do not support WebSockets.

Although Vercel is not the only option for deploying a Next.js application, this limitation might be a deal-breaker for some."

> Bilgili, D. (2022) _Implementing WebSocket communication in Next.js - LogRocket Blog_. Available at:

<https://blog.logrocket.com/implementing-websocket-communication-next-js/>.

Mistakes I have made

With little prior knowledge about NextJS, I thought the framework would be a good fit for the application I was looking to create. I slowly found out as I was doing my research and attempting to implement a prototype that NextJS is a very good framework with focus on the frontend and serving up assets, due to its optimized stateless architecture, but is a terrible choice for anything that requires states to be tracked such as a real-time chat app due to that stateless architecture. My aim with this framework was to implement a beautiful frontend design that would entice people to use the app. While this can still be achieved for the final product, it is not within the scope for the prototype and I will therefore be looking for other solutions.

With this in mind, my options are either to find a solution through the use of external API tools that can work around the issue of NextJS stateless architecture, such as Pusher, Firebase, Supabase as a backend and database, or to run a separate backend such as NodeJS and Express alongside my NextJS app.

I can also explore using other frameworks within the same language so that my experiences and knowledge can be easily transferred over. This include Sveltekit, Angular as well as vanilla ReactJS.

Evaluation of prototype

I have managed to implement a simple prototype that demonstrates live messaging through websockets. The application uses React as the frontend, which communicates to the backend which is built on NodeJS and Express, to send messages between multiple users through the use of websockets.

Things to consider

I would like to implement a similar prototype within NextJS or Sveltekit as those two frameworks are most similar to ReactJS and perform better than ReactJS.

I would also like to explore other solutions as well such as using Firebase or something similar to handle the backend and database or following up the feature prototype with a database to store and emit the messages.

As for the technology stack I intend to use, I would like to explore the use of some other frameworks. Other frameworks tend to provide developers a better experience with a much faster setup time for developer servers. This includes retrying NextJS with a new solution (Node and Express backend or backend services), Sveltekit as well as all 3 of these frameworks with the use of Vite as a module rollup.

NEW IMPLEMENTATION

My first prototype was using React with a node.js express server receiving calls from the front end using websockets. This allowed two or more clients to talk to each other. This prototype was just a proof of concept to see if I could achieve real-time chat functionalities for the final submission. There were many features missing that I would like to include in the final submission.

I chose for the project to be done with a JavaScript framework as that is what has been popular and current with the webdev job market. I chose NextJS as the framework to work with as the whole Server Side Rendering and Client Side Components interested me, and allowing devs to selectively choose how their application runs per component.

This did come at a cost, as I later found out that I would have to sacrifice other technologies I was hoping to use, that being websockets. Websockets is not currently support by NextJS, especially in their recent big release of version 13 with the swap to a new API server routes. I did try to run websockets on a Node with express server on a separate layer, which I did for my prototype. The prototype was originally using NextJS as the frontend, but due to my lack of experience and research, I couldn't figure out why they weren't communicating with each other, so I gave up on that idea and used plain React instead.

Some of the key features I decided to include in the final version was a proper login/registration and a user system to differentiate who is messaging who. This was achieved using NextAuth, a popular authentication and authorization library used in JavaScript frameworks that allows the user to sign in using either credentials or providers that I choose to include. In my case, I chose to include GitHub and Google as those are most common accounts for people to have. This also helped to protect different routes from unauthorized users or protect conversations from users that aren't involved in them.

I wanted to store the messages in a database so that users can have conversations and not worry about losing the chats that may contain important topics as this is basically being used as a knowledge management system. I originally wanted to explore using Redis as a database as I heard that it was quite fast and ideal for this real-time chat use case but was unable to get the Redis adapter for javascript working so instead I decided to use MongoDB hosted on mongodb.com and the database model was laid out using Prisma schemas. See [Appendix A](#) for reference to the ER diagram for my Prisma schema and [Appendix B](#) for the Prisma schema. I will also leave a link to dbdiagram.io to easily highlight and see the relationships as it can be a bit hard to discern which relationships are connected to where without highlighting.

From this diagram, it describes the relationships between the tables and information concerning themselves. An Account has a one-to-one relationship with User referenced with userId. A User can have many Conversations and a Conversation can have many Users. This is done by storing conversationIds fk referencing Conversation pk as an array value storing conversationId. This is because many Users can share a conversation, but they can also have other conversations as well. A similar relationship is used for seen Messages as a message can be seen by many users, and a user can see multiple messages at a moment. A user can also send multiple messages but a message can only be sent by one sender represented with user ID pk to Message senderId fk. A user can be involved in multiple conversations but a conversation can only have one instance of that user, but hold multiple different users which is represented with userIds [] fk referencing User ID pk.

While the ER diagram itself shows that this does not meet 1NF because of the use of array entities, it is common practice to denormalize data when creating the physical model as the use of arrays can be more efficient and lighter with the only downside being migrating to a different database system as some of them do not support arrays. There are more relationships, but I don't wish to elaborate more as it will be similar to what I have already described.

Another key aspect for the app was for the messages to be in real-time. In a collaborative discussion, people normally hit off each other's thoughts and ideas and without real-time messaging features, this could hinder the users as they would likely get distracted or do other tasks while waiting for a message to come in. This was done with the use of Pusher which is a hosted websocket solution. I used pusher as an alternative to websockets as I have seen from multiple community sources that websockets is not supported on the new App API routes. And while a solution could maybe be done using a backend Node express server, type safety and typescript would not be supported with this method and I wanted to explore using as it is an industry standard.

Pusher works similarly to websockets where by subscribing to a channel. These channels are only subscribed to in the message body and conversation list where we are showing real-time messages and notifications. We then bind the client to events to listen and receive any changes from the server. The server will "trigger" these events. For example, in the MessageBody component where users receive messages, when a message is sent, regardless of which user, a post request is made to the server, this then creates a new message in the conversation, and after that a pusher event is triggered on the server. The client picks up on this trigger because it is bound to it, and then this fires off another function which updates the client side interface with the new message. Similar methods are used for different functions, but basically a user fires off a request, this request then triggers an event which updates the client side interface.

And all this happens upon connecting to the conversation page, where a `useEffect` is updated whenever the `conversationId` changes. Meaning that, we essentially mount the users in the conversation into a pusher subscription. And this when a new message comes

in, if the other user is in the conversation currently, another call to the server is made to update the message object that the user has seen the message.

Going back to the conversations page, a list of conversations is displayed. Each one of these has a conversation box representing each ongoing conversation with another user. They each have an id assigned to them, and when clicked on, we use useRouter from next/navigation to push the url to the next page, so that we can use them as parameters to pull in data regarding that conversation, such as the message content.

```
const onClick = useCallback(() => {  
  router.push(`/conversations/${data.id}`);  
}, [data.id, router]);
```

This gets passed to the conversationId page and can then be used to get the messages within the conversation using the custom helper function getMessages which just takes in the conversationId string and uses prisma to find all messages linked to that conversationId. As for the message types, users can send either plain text messages or images which was implemented using Cloudinary, an image and video storage platform. So all images are stored there and a url string is saved into the database for the app to pull the image from.

The pinning feature was done using a boolean called pinned which will be empty upon message creation. The user can then click on the messages and pin the message which sends a call to the server, which then performs an update on the message. The pinned value gets updated with true. And the pinned message page will only display messages that have the pinned value as true.

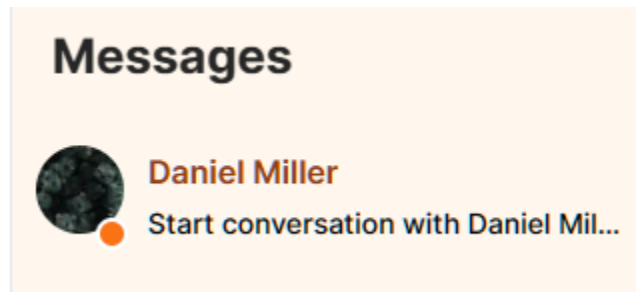
CHAPTER 5: EVALUATION

The initial evaluation for the project was done through manual testing. I verified as I developed the app to see if everything worked as intended as each element was crucial to the apps success.

The manual testing I did while developing the app was just me verifying that the components worked as they should. We start with the login/registration which can be seen at [Appendix C](#). The login/registration page is styled in colours I believe to be warm and cosy to match the nature of cooking as a hobby. Its supposed to invoke a calm response from the user. The function of the login page works as expect, logging in users that are registered on the app. The registration could use with some form validation. Uncommon symbols can be used for the user name and the password has no minimum length.

The manual testing for the dashboard page just required the layout to show no chat was open, show users on the platform that they can start a conversation with and allow them to navigate to the conversations page or logout. This is the expected outcome for the page.

Navigating to the conversation page shows a list of conversations. The conversations will show the latest messages or tell the user that an image was the most recent message, or if no messages have been sent yet, “Start conversation with {username}” will be displayed.



This is also expected behaviour. The conversation list is updated in real-time to show the latest response from the other user.

Clicking onto the conversation brings us into the message conversation page with the other user. A complete messaging ui is displayed, with a message body, a header bar with the other user displayed, and a message input. Messages are sent in real-time which can be seen from my app demo. It also displays relevant info such as the time the message was sent, whether the other user has seen the latest message. The messages are in sequence of latest sent. The header bar needs some more work as currently the use active status is hardcoded and non-functional.

The chat window also lacks a conversation info, which would be used to show a delete button for the conversation as well as in future development, show the list of users and the group name of the group as an additional feature. I wanted to add this feature as I think group conversations adds for more productive discussions among the cooks using the app and discussing about recipe ideas.

With further develop, I also wish to add rich text to the messages so that users can show bold headers, italic emphasized words and other kinds of styling to the messages. This would also help to show the pinned messages in a better format, closer to the intended recipe cards.

The pinned messages page shows the pinned messages correctly. The messages are in sequence of latest sent. With further develop, I wish to implement something I think necessary which is unpinning of messages. Some messages might no longer be relevant or might have been pinned accidentally.

Lastly, once the user signs out, the page is redirected back to the login page. This is as expected as I implemented a middleware to the app that allows pages to be protected based on whether or not there's a user session.

I also performed some unit tests to verify that my custom button components, as well as the NoOpenChat component displays whatever is passed into the component correctly. AuthForm was in the process of being tested but I did not have enough time to research on testing methods for that component. Based on the complexity of the component, I would have to do either end-to-end testing, integration testing or snapshot testing for such a component. The only thing I managed to do was mock up the user session for the

component, but I immediately met with another problem which was how to mock Next's useRouter hook.

If I managed my time better, I would have liked to learn how to do proper testing for my app using the correct methods.

CHAPTER 6: CONCLUSION

As a summary, my real-time cooking chat app has the basic functions for a chat app, using NextJS for the frontend as well as the new API routes for the server, authenticating and authorizing with NextAuth, storing data in the cloud using MongoDB via prisma, and achieving real-time chat like features with pusher.

Given more time, I would have liked to add a few more features to complete the basic chat app features, such as group chats, deleting messages, and a better formatting for the pinned messages as they were initially meant for recipe lists.

CHAPTER 7: APPENDICES

Appendix A: ER Diagram



<https://dbdiagram.io/d/650c474202bd1c4a5e04d7f6>

Appendix B: Prisma Schema

User Table

```
model User {
  id          String @id @default(auto()) @map("_id") @db.ObjectId
  name        String?
  email       String? @unique
  emailVerified DateTime?
  image       String?
  hashedPassword String?
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  conversationIds String[] @db.ObjectId
  conversations Conversation[] @relation(fields: [conversationIds], references: [id])
  seenMessageIds String[] @db.ObjectId
  seenMessages Message[] @relation("Seen", fields: [seenMessageIds], references: [id])

  accounts Account[]
  messages Message[]
}
```

Account Table

```
model Account {  
  id String @id @default(auto()) @map("_id") @db.ObjectId  
  userId String @db.ObjectId  
  type String  
  provider String  
  providerAccountId String  
  refresh_token String? @db.String  
  access_token String? @db.String  
  expires_at Int?  
  token_type String?  
  scope String?  
  id_token String? @db.String  
  session_state String?  
  
  user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
  
  @@unique([provider, providerAccountId])  
  ..  
}
```

Conversation Table

```
model Conversation {  
  id String @id @default(auto()) @map("_id") @db.ObjectId  
  createdAt ..... DateTime @default(now())  
  lastMessageAt DateTime @default(now())  
  name String?  
  isGroup Boolean?  
  ..  
  messagesIds String[] @db.ObjectId  
  messages Message[]  
  
  userIds String[] @db.ObjectId  
  users User[] @relation(fields: [userIds], references: [id])  
}
```

Message Table

```
model Message {
  id: String! @id @default(auto()) @map("_id") @db.ObjectId
  body: String?
  image: String?
  createdAt: DateTime! @default(now())

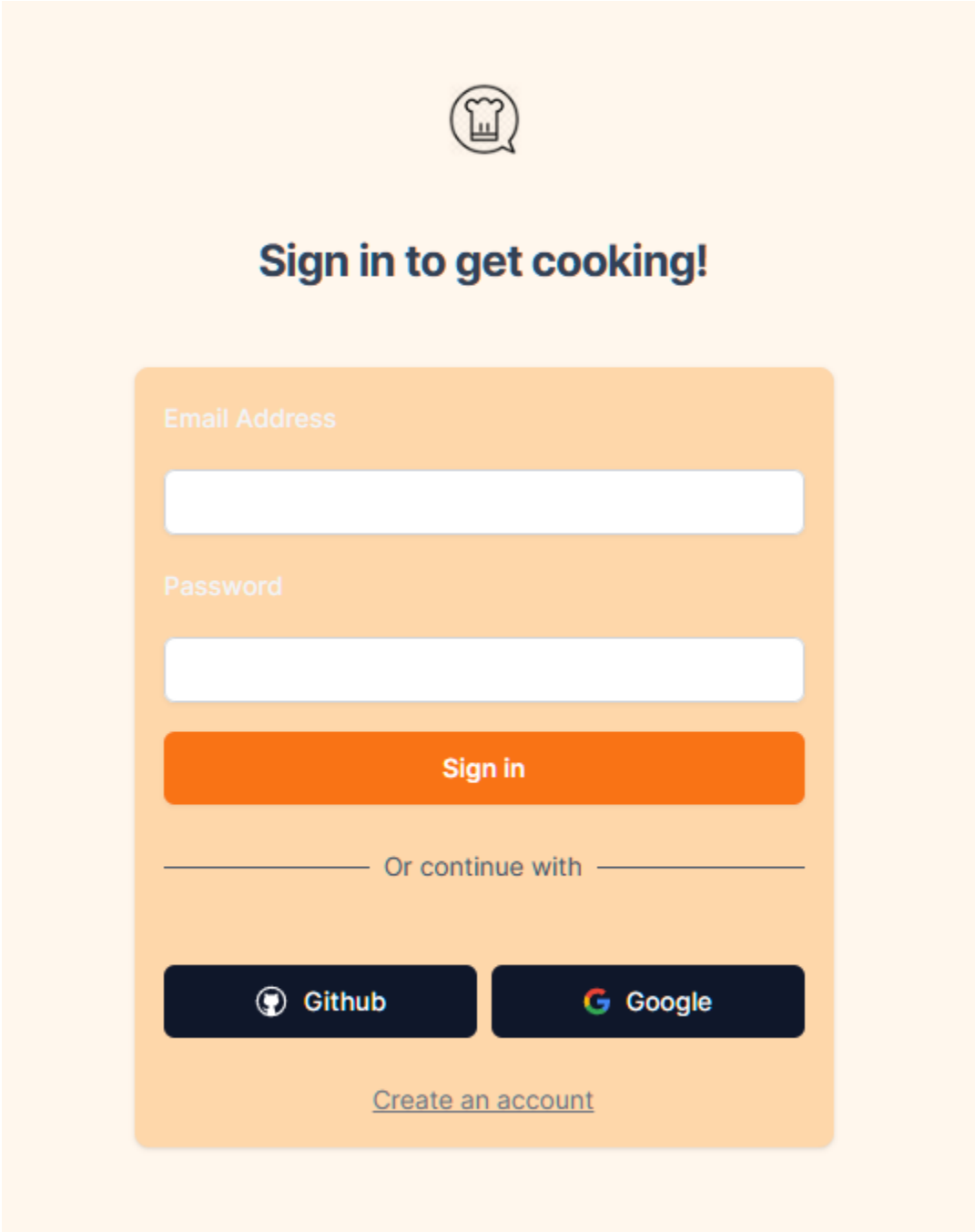
  pinned: Boolean?

  seenIds: String[]! @db.ObjectId
  seen: User[]! @relation("Seen", fields: [seenIds], references: [id])


  conversationId: String! @db.ObjectId
  conversation: Conversation! @relation(fields: [conversationId], references: [id], onDelete: Cascade)

  senderId: String! @db.ObjectId
  sender: User! @relation(fields: [senderId], references: [id], onDelete: Cascade)
}
```


Appendix C: Login/Registration form



The image shows a login and registration form for a cooking application. At the top, there is a circular icon containing a chef's hat. Below the icon, the text "Sign in to get cooking!" is displayed in a bold, dark blue font. The form itself is a light orange rounded rectangle. It contains two input fields: "Email Address" and "Password", both with white text on a light orange background. Below the "Password" field is an orange "Sign in" button. Underneath the button is a horizontal line with the text "Or continue with" in the center. Below this line are two dark blue buttons: "Github" with a white GitHub logo and "Google" with a white Google logo. At the bottom of the form is a link that says "Create an account" in a dark blue, underlined font.





Sign in to get cooking!

Email Address

Password

Sign in

Or continue with

 Github  Google

[Create an account](#)



Sign in to get cooking!

Name

Email Address

Password

Register

Or continue with



Github



Google

[Login](#)

CHAPTER 8:REFERENCES

Link to code hosted on Github: <https://github.com/doubleOZ/Final-Project.git>