Mellow

A Project Report
Presented to
The Faculty of the Computer Engineering Department
San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Bachelor of Science in [Computer/Software] Engineering

By

Albert Lu
Kha Nguyen
David Sambilay

03/2023

APPROVED FOR THE COLLEGE OF ENGINEERING

Andrew Bond, Project Advisor

Dr. Wencen Wu, Instructor

Dr. Rod Fatoohi, Computer Engineering Department Chair

ABSTRACT

A Mellow Approach to Organize Your Day

By Albert Lu, Kha Nguyen, David Sambilay

It is important to maintain organization in our lives, especially when responsibilities and commitments overwhelm us. One of many methods to achieve this is to list out the tasks to accomplish. This method makes our tasks manageable, and it helps us stay focused by having a clear outline of our responsibilities. Furthermore, creating a list of daily tasks helps to increase our productivity, decrease stress, and gives us a sense of satisfaction and accomplishment as we check off parts of our lists.

Even with methods like to-do lists, most people still struggle to maintain order in their lives. As our society becomes more digital, people's lives are getting busier and distractions are common. An efficient time scheduling method is in demand. With so many tasks to consider, it is hard for one to mentally consider them all in a simple way. Furthermore, the manual note taking method is not efficient enough to timely manage a big task list.

Mellow provides the solution for everyone to organize their tasks and create effective schedules. Our goal was to create an easy-to-use experience and aimed at letting users "do more with less", rather than spend time figuring out our application. Mellow uses a minimalistic but high-quality approach for users to navigate through the application so that users can effectively organize their plans. This was completed through the creation of a simple user interface design that is highly ergonomic and pleasing to view through the use of common and widespread design patterns. The design is also intended to help increase user retention and thus eliminate the need for the user to seek out a different application. As Mellow is a digital application, it is certainly far easier to record and keep track of a big task list compared to hand-taken notes.

Acknowledgments

We would like to thank San Jose State University for helping us gain technical and in-depth knowledge to develop this project. Furthermore, the school has created a platform and network for us to learn, pursue our passion and importantly, meet each other in order to complete this work.

We would also like to acknowledge the work of Dr. Wencen Wu for assisting us throughout the process of completing the project.

Special thanks to our advising professor Andrew Bond who guided us with information and experience to build the vision of our product. It is a great joy to have his friendship, and sense of humor to accompany us along the journey.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction (David Sambilay)

1.1 Project Goals and Objectives

The goal of Mellow is to let everybody have their own personal time management application that allows them to organize their lives. We want Mellow users to be at their most productive and to have an overall positive well-being. The broader context in which Mellow was placed is to be a relevant software in the productivity app market that can be used in any industry from personal to professional/organizational use.

1.2 Problem and Motivation

The problem Mellow tries to solve is to increase a person's effectiveness when encountering hectic schedules or tasks. As engineering students at San Jose State University, we also have our fair share of hectic courseworks, study material, and assignments that make our schedule hectic. Our experiences eventually lead us to coming with Mellow. Mellow is important because it is designed to let users create organized tasks and planned-out schedules. It addresses the problem of having hectic schedules and tasks which aligns with the motivation for creating Mellow.

1.3 Project Application and Impact

Mellow's capability to organize people's tasks and create effective schedules is suitable for people who have or will have busy schedules in areas such as academics, business, career, family, and more. For instance, college students can use Mellow to create a schedule and organize their tasks for a better productivity space. Mellow can help these students stay focused on their academic goals and make each goal manageable. Overall, it brings an impact to its user's productivity which leads to the betterment of every person's well-being that contributes to society.

1.4 Project Results and Deliverables

The project results we have are a fully functioning front-end designed based on the component and architectural design created. A serverless framework to connect the frontend and backend components are also created. The project deliverables include the code used for the Mellow architecture for both frontend and backend development.

1.5 Project Report Structure

The following report structure follows with these chapters. Chapter 2 starts with Mellow's brief background and continues to talk about related works. Chapter 3 goes over the overall project requirements for Mellow. It goes into detail with the business requirements, functional and non-functional requirements, interface, and technology requirements. This chapter elaborates everything needed for Mellow’s development. Chapter 4 shows Mellow’s design patterns. This chapter also includes the constraints, problems, and trade-offs encountered from the selected design and requirements. Lastly, Chapter 5 goes over the implementation of the design and requirements to create Mellow.

Chapter 2 Background and Related Work (Kha Nguyen)

2.1 Background and Used Technologies

Mellow will use a client-server architecture consisting of a backend and a frontend. The backend can be divided further into server and database components.

The database is expected to be PostgreSQL sourced from Amazon RDS. Amazon RDS offers cloud database hosting, which is a convenient alternative to self-hosting. This will offer the most reliability. The database itself will serve as a way to store data of all users of Mellow. It will allow data to be persisted even when the user is offline. This database will connect into a server running with another AWS Cloud service, Lambda.

The server will be the primary connection between the frontend and database. All connections will be routed through it. This prevents individual clients from directly accessing the database and improves the security of Mellow. Django will serve as the program running on the server. It is a free and open-source web framework that uses models and templates to abstract the process of handling data to and from the frontend.

The frontend is on the client side of the user. It will be based on a Model-View-Controller (MVC) architecture and will use HTML (HyperText Markup Language), CSS (Cascading Style Sheets), and Javascript. HTML (HyperText Markup Language) is used to write out the components to be used. CSS (Cascading Style Sheets) is intended to style the HTML components. To further aid in the development of the user interface, CSS libraries may be researched and used. Javascript is used to handle user interactions and programmability. But more specifically, the React.js framework will be used, which is based on Javascript. React.js uses Javascript and provides a simple way to handle Javascript interactions with HTML objects.

The model is the data stored temporarily in the browser and will be the one that chooses what the view will render. It will also provide the data for the controller to use. The view is what the web browser will render on the screen to the users. The controller is the logic within the React.js framework. It will handle any user interactions from the user view and will modify the model state accordingly. It will also handle any connections to the backend server, such as retrieving data from the database.

The connections between the backend and frontend are also important. Since Mellow is a web application where the client uses a web browser to connect, the link between the frontend and backend will be HTTP. This is the primary protocol that is used for web servers to communicate with clients.

To handle and control requests between the frontend and the Cloud server, Amazon API Gateway is utilized. This tool helps to create RESTful APIs in the server to manage traffic and receive, answer calls. Furthermore, API Gateway would direct and trigger Lambda computer server. The backend codes are stored in Lambda functions to communicate with the Postgres database.

To verify and access data based on different users, AWS Cognito is the tool for authentication and authorization. This feature not only helps to store and save identity, but also protect user information. Cognito as a backend solution connects with the client server through HTTP policy and with Lambda functions in the backend for each user data.

2.2 Literature Search

The goal of Mellow is to let everybody have their own personal time management application that allows them to organize their lives. We want Mellow users to be at their most productive and to have an overall positive well-being. The effectiveness and effects of various time management tools and methods is a well-researched area. A research paper from the Journal of Education for Business explores the impact of reminder apps on student learning and anxiety. Their findings suggest that reminder apps can improve students' feelings of organization and time management (Simmons et al., 2018). It also has a direct effect on their educational outcomes for they help ease anxiety and stress.

In many ways similar to a regular time management application, research by Stawarz and Blandford also explores possible designs for an effective medication reminder application (2014). Although most patients have access to smartphone applications designed to help them remember their medication, the effectiveness and quality of the app has not been evaluated yet. This can also be a similar issue when designing a time management app since users will tend to forget about their tasks or the tasks they added into Mellow. According to their paper, instead of passively reminding users through simple timer-based designs, reminder apps should be seen as an assistant that helps them achieve their goals and that helps them develop a new behavior (Stawarz & Blandford, 2014). Mellow needs an assistive feature similar to the medication reminder app, so its users can truly experience an application that will boost their productivity and organize their schedules.

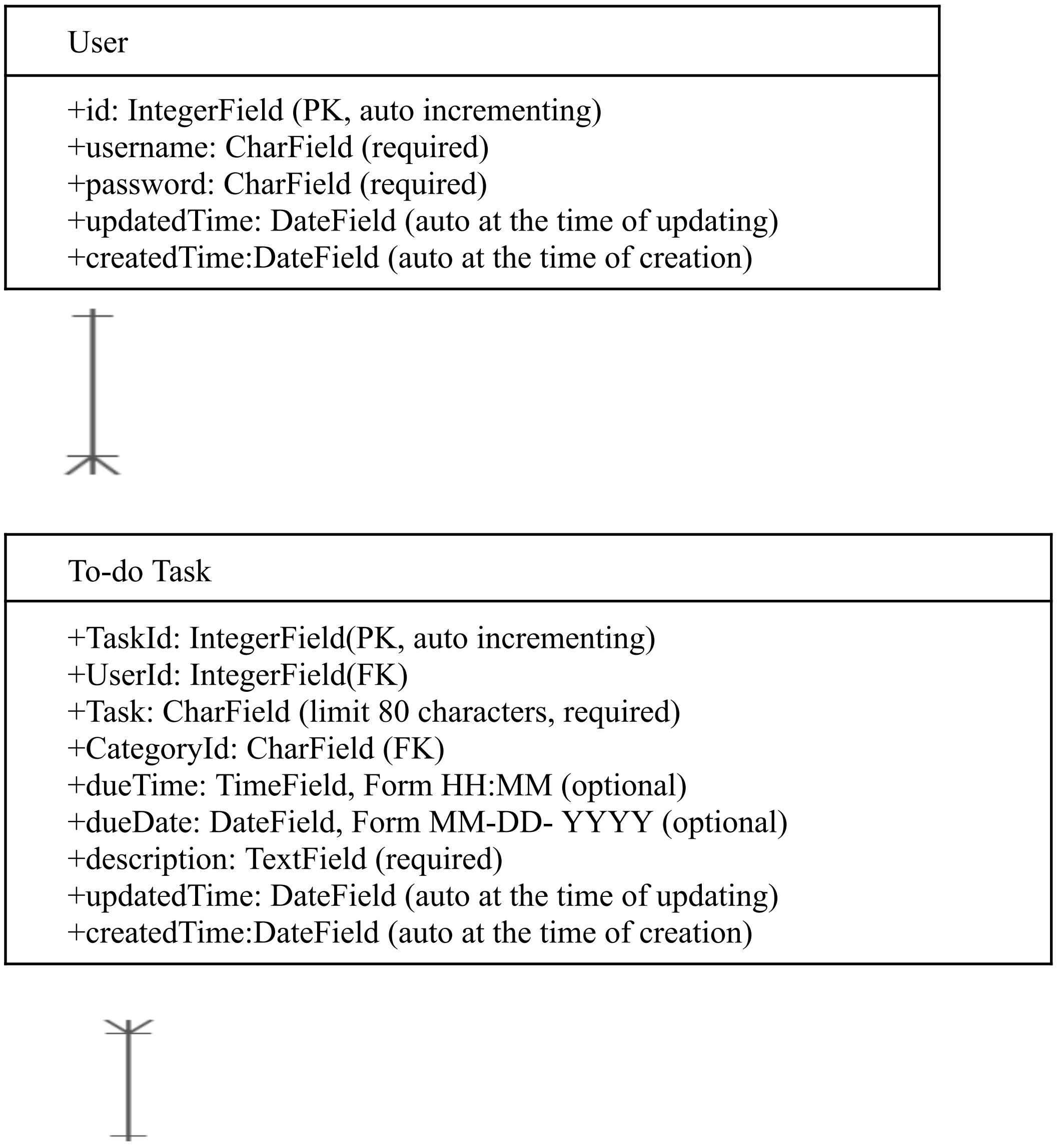
2.3 State-of-the-art Summary

There are several methods currently used in order to maintain order in our lives. One of these methods is the to-do list. According to Schrager and Sadowski, a todo list can take in many forms. A traditional way of doing it is pen and paper. There are also to-do list applications available in our computers, smartphones, and tablets. In order to make to-do lists more efficient, they mention the book “The 7 Habits of Highly Effective People” which suggests separating tasks in categories of importance and urgency (Schrager & Sadowski, 2016). Breaking down tasks into smaller chunks of tasks also makes to-do lists more accomplishable.

In some cases, we need to manage something in our lives that are more complex than simple tasks we can write down in a to-do list. This could be a personal project, an idea, or a group of people. A much more complex tool or system is required to help us manage these complex tasks and responsibilities. The article “Managing Ideas, People, and Projects: Organizational Tools and Strategies for Researchers” discusses certain tools and softwares that, together or individually, can be used to organize huge workloads like projects (Levin & Levin, M., 2019). Tools and software mentioned in this article includes Adobe Acrobat, Dropbox, MindNode, Evernote, Microsoft Word, and more. In addition, the article provides basic requirements a software/tool must have to enable their user to efficiently manage complex tasks. For instance, information should be organized hierarchically and searchable by keywords. Information should be easy to find and access to enable the user to do as little work as possible and focus more on the tasks at hand. These are some of the tools and principles that are currently being used today when one is managing complex tasks that a simple to-do list would simply not be enough.

Chapter 3 Project Requirements (Kha Nguyen)

3.1 Domain and Business Requirements



User

+id: IntegerField (PK, auto incrementing)
+username: CharField (required)
+password: CharField (required)
+updatedTime: DateField (auto at the time of updating)
+createdTime:DateField (auto at the time of creation)

To-do Task

+TaskId: IntegerField(PK, auto incrementing)
+UserId: IntegerField(FK)
+Task: CharField (limit 80 characters, required)
+CategoryId: CharField (FK)
+dueTime: TimeField, Form HH:MM (optional)
+dueDate: DateField, Form MM-DD- YYYY (optional)
+description: TextField (required)
+updatedTime: DateField (auto at the time of updating)
+createdTime:DateField (auto at the time of creation)

Category
+CategoryId: IntegerField(PK, auto incrementing) +CategoryName: CharField (limit 30 characters, optional)

Table 1. Database design model

3.2 System (or Component) Functional Requirements

Mellow User Story

#	Role (As a …)	Action (I would like to…)	Reason (So that…)
1	New User	Create a new account	I can join the Mellow application as a user.
2	Existing/New User	Sign in to account	I can access my personalized account and data.
3	Existing/New User	Log out of account	I can leave the application.
4	Existing/New User	Add task in to-do list (title, description, category, time)	I can save my task
5	Existing/New User	Remove existing task	I do not need the task anymore
6	Existing/New User	Edit task description	I can change detail of the task
7	Existing/New User	Select task's status	I can remind if it is finished or not
8	Existing/New User	Add task's category	I can organize tasks in specific category

9	Existing/New User	Remove task’s category	I can remove the category if not needed
10	Existing/New User	Search task information	I can look for task that I needed faster

Table 2: Functional requirement

Desired feature:
- User feedback
- Responsive mobile version

Optional feature:
- Reward system
- Colorized modes

3.3 Non-functional Requirements

- Simple to navigate: Reduce more than two seconds of response time with intentionally placements of components.
- Easy to use without complex or unnecessary steps to finish actions: Reducing numbers of clicks or scrolls to create tasks.
- Work as intended with little errors possible after detailed testing and quality assurance: Perform ninety percent of the use cases
- Fast response time with optimized codes and algorithms, not including server speed and latency.
- Effective interface interaction with users for easier understanding of functionality of each component: Reducing response time with icons instead of long text on categories.

3.4 Context and Interface Requirements

Context environments:
- Code editor and environment: Visual studio code
- Programming language: Javascript, Python
- Markup language: HTML, CSS
- Front-end library: React
- Backend framework: AWS Cloud services (Cognito, API Gateway, Lambda)

- Code storage and workplace: Github
- Communication: Discord
- Management: Trello

Interface requirement:

- Desktop or laptop with screen and monitor connecting to internet
- Internet browser

3.5 Technology and Resource Requirements

Hardware Requirements	**Technology**
Operating System	MacOS/ Window/ Linux
Input Device	Keyboard, mouse
Output Device	Screen

Table 3: Hardware requirement

Software Technology	**Description**
MaterialUI	Frontend library
ReactQuery	
JWT (Json Web Token)	Security for data encryption/ decryption
React Router Dom	Configuring routes
Amazon Cognito Identity	Middleware communication with Cognito service for frontend
PostgreSQL	Database

Table 4: Software resources

Chapter 4 System Design (Albert Lu)

4.1 Architecture Design

As stated previously, Mellow's architecture will mainly consist of three parts: a frontend, backend, and database. **Figure 1** below shows the full architecture of Mellow. It is clear from this figure that the backend will consist also of the database.

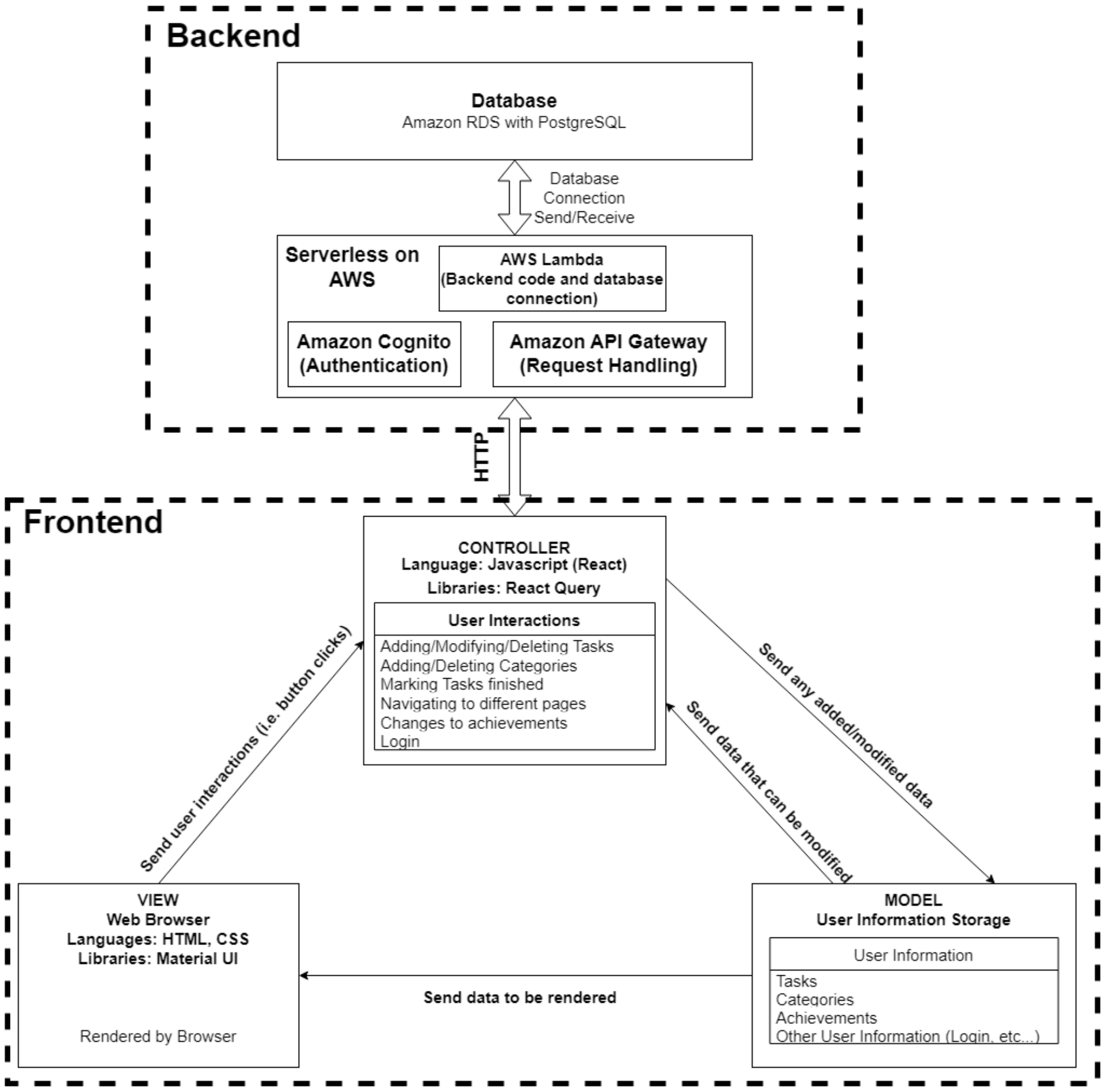

Backend
Database
Amazon RDS with PostgreSQL
Database Connection Send/Receive
Serverless on AWS
AWS Lambda (Backend code and database connection)
Amazon Cognito (Authentication)
Amazon API Gateway (Request Handling)
HTTP
Frontend
CONTROLLER
Language: Javascript (React)
Libraries: React Query
User Interactions
Adding/Modifying/Deleting Tasks
Adding/Deleting Categories
Marking Tasks finished
Navigating to different pages
Changes to achievements
Login
Send user interactions (i.e. button clicks)
Send any added/modified data
Send data that can be modified
VIEW
Web Browser
Languages: HTML, CSS
Libraries: Material UI
Rendered by Browser
Send data to be rendered
MODEL
User Information Storage
User Information
Tasks
Categories
Achievements
Other User Information (Login, etc...)

Figure 1: Full Architecture View of Mellow

Figure 1 shows how the flow of the data was expected to go from a high-level point of view. The frontend strictly follows a Model-View-Controller (MVC) architecture model that is implemented through common web technologies (HTML, CSS, Javascript). The interconnect between frontend and backend is essentially HTTP at the high level. The backend details the different technologies used and is essentially based on the "serverless" architecture from AWS. There are essentially three key parts as shown. It is a service-based architecture that allows for high flexibility and simplicity in design and implementation.

4.2 Interface and Component Design

The components of the frontend and backend are further detailed below. Figure 2 and Figure 3 give an overview of the login system and the actual pages to be implemented. They also show the different pages and how they are designed, such as what state to store and what user interactions are necessary. This drives the decisions into what UI components to display, such as a "login button".

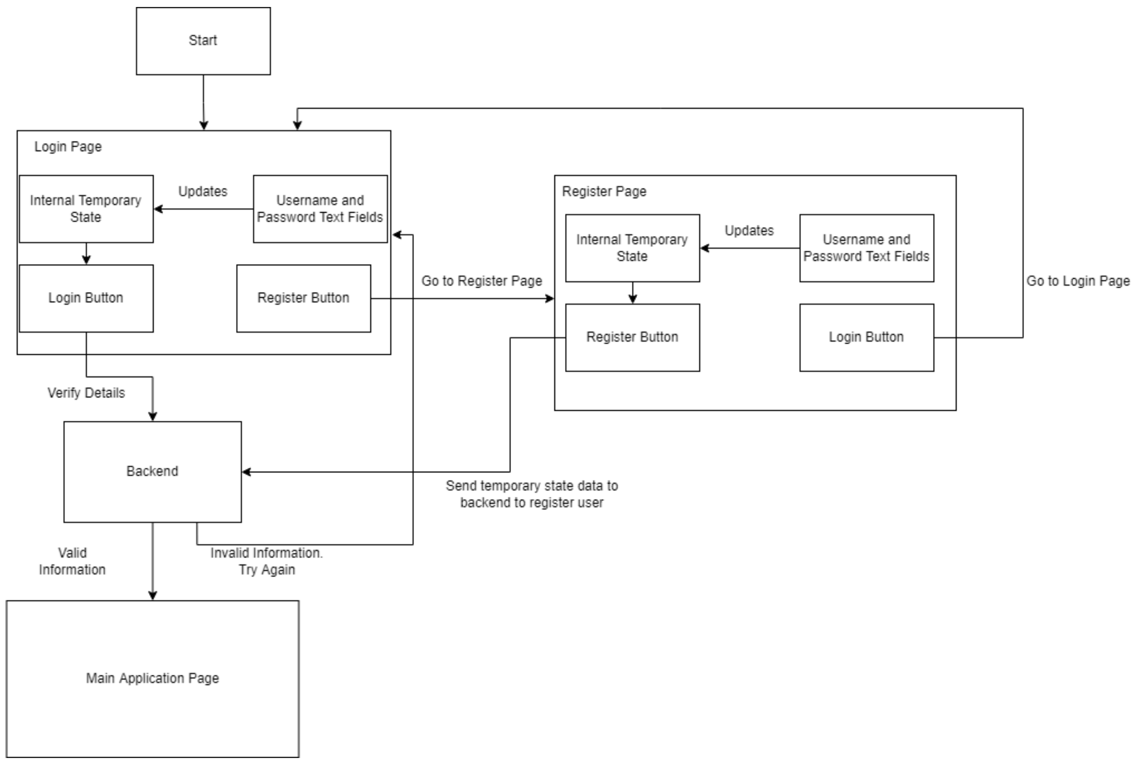


Figure 2: Main Page Flow Diagram

The page flow diagram shows another view of the architecture in how specific elements on the page will lead to what outcomes. It can be seen that three main pages will be used: a login page, registration page, and a main application page. The login and registration pages will rely on the backend significantly, so a separation of responsibilities is necessary. The main application page is where the list of tasks would be displayed. It is the main page that the user is expected to interact with.

Figure 3 shows the flow of the application after login in the main application page. It can be seen that a choice of two pages is available: a welcome page or a tasks list page. It is expected that the default page after login is the welcome page. The user can then choose the tasks list page. The data that is sent to the pages is also shown. It can be seen that the welcome page only needs the daily tasks list, which is expected to be just a filtered out version of the all tasks list. Quotes and achievements that help to make the application more "fun" are also fetched.

In the task lists page, two choices of views can be shown: a calendar view or list view. This is given as a preference to the user. Once the view is selected, all tasks will be displayed, but can be filtered out depending on which list of tasks are desired, such as only today's tasks or only the finished tasks.

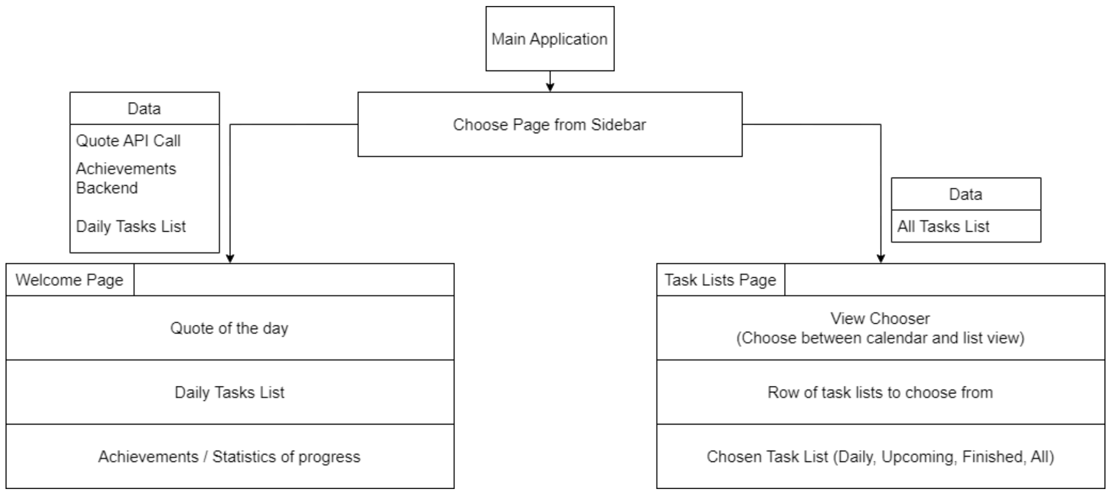


Figure 3: Pages and Data Flow Diagrams after login and registration

In the task views, it should be further noted that the tasks can be filtered by category name and also by different time settings, such as “today”, “upcoming”, and “past”.

It is important to also consider the interconnection to the backend. The table below shows the API calls that are used to interface between the frontend and backend. Some API calls have optional inputs and some may need authentication to be accessed. Authentication is necessary to distinguish between users for security purposes. It should also be noted that some inputs have optional filtering to allow the frontend to only display the tasks that are chosen by the user.

API Call	**Type**	**Inputs (not including authentication)**	**Outputs (Returns HTTP 200 for success)**	**Authentication Required?**
Get Tasks	GET	**Optional Filters:** Category Date Time Finished Status	Tasks lists (can be filtered) *Each Task:* Task ID Task Name Category Date Time	Yes
Add Task	POST	Task Name Category Date Time	N/A	Yes
Change Task	POST	Task ID Task Name Category Name Date Time	N/A	Yes
Delete Task	DELETE	Task ID	N/A	Yes

Get Categories	GET	None	Categories List	Yes
Add Category	POST	Category Name	N/A	Yes
Delete Category	DELETE	Category Name	N/A	Yes
Login	POST	Username Password	Authentication JSON Web Token (JWT)	No
Register	POST	Name Username Password Email Address	Authentication JSON Web Token (JWT)	No
Verify User	POST	Special code from email	N/A	No

Table 5: API Calls to interface the frontend with the backend

4.3 Structure and Logic Design

This section will further detail the structure and logic design of the components and how they interact at a lower level. It also explains the various flows of the application.

The frontend will first be discussed in terms of its structure and logical flow. As previously explained, it is based on a Model-View-Controller (MVC) architecture.

The model is essentially a temporary data storage that is to be placed on the user end side. It would not make sense to place this part on the server and to manage it for every user that accesses the application. A library called “React Query” was used to manage the “model” part of the application. It helps make it easier to ensure all components are synchronized with the same data. All flow of data coming from the server would need to be handled by the model, but the model would also have to interact with the controller to send the proper HTTP requests too.

The view is the web browser and what the user sees. It is the design of the pages and was done in HTML and CSS.

The controller is the part that controls the interconnections between frontend and backend and also the communications between the model and view. It also processes the user interactions and determines how to update the next state. It is the one that sends the HTTP requests to the backend and waits for the response. This response is then processed and sent to the model for storage. If any user interactions, such as pressing a button, are initiated, then the controller determines what to do next. It may change the view or update the model.

The backend is mainly the serverless architecture from AWS that provides all that is needed as a service. **Appendix 2** shows a diagram of the architecture more specifically. It can be seen how the different AWS services interact with each other. React interacts with the API gateway. AWS Cognito is also used to authenticate the user to provide access to the requested lambda functions. The lambda functions serve as the new logic of the backend. The logic is separated into callable functions that request data from the Amazon RDS PostgreSQL database and also performs some processing if necessary. This processing can include filtering of the data before sending to the React frontend.

4.4 Design Constraints, Problems, Trade-offs, and Solutions

4.4.1 Design Constraints and Challenges

From a resource and reliability perspective, we decided that libraries were a more practical choice to use rather than building a custom platform for Mellow. It would save on resources and be much more reliable than what can reasonably be done within our required development timeframe. ReactJS has a large community base and this means it would continually receive updates and bug fixes. This helps to ensure it is reliable. In contrast, if we were to develop our own solutions, we would be required to handle these fixes ourselves, which would add to the maintenance and development costs.

From a software perspective, some trade-offs and design choices were made. These choices were made to be able to fit within our constraints of a web application and also the problem we were trying to solve. Issues, such as browser compatibility and memory constraints were primarily considered. These are further explained in Section 4.4.2.

From a societal perspective, we had also previously considered the usage of advertising to generate revenue to support development. This was ultimately found to be a bad approach due to privacy and security concerns. Privacy and security are major societal issues in today's world. Advertisements are known to use tracking, which would hurt the privacy of our users. Incorporating privacy and security helps users trust Mellow. Furthermore, advertisements would also hurt the usability of the application for our users. This could be a major weak point that causes users to leave our platform over time. However, revenue is still important for maintaining the application. We discuss solutions in Section 4.4.2.

4.4.2 Design Solutions and Trade-offs

One alternative design choice considered was the database type. There were two options we found viable: relational databases and non-relational databases. Relational databases were what were first considered, such as SQLite, MySQL, and PostgreSQL. Non-relational databases would essentially be something like MongoDB, where data would be stored in a JSON file format. Non-relational databases are often easier to understand, but in the end, we found that our data fit a relational database much more elegantly. Our data would essentially be storing tasks related to a user, which is a key relation already. Another reason for the choice of a relational database is that it had more resources available and more services seemed to provide hosting for it. It is a "tried-and-tested" approach. As for which relational database, we decided to use PostgreSQL because it is a popular, modern choice for a database. It also supports being hosted on Amazon RDS, which will significantly help our development time for Mellow.

Another design choice we had to consider was whether to use ReactJS or just plain Javascript. React allows for faster development time and access to a large community of developers. This would allow us to solve our problems faster. It is also modular and allows for easier design. Plain Javascript would mean no framework is used, so it may take longer to develop due to needing to build basic building blocks from scratch. The advantage is the better portability if we ever decide to switch frameworks. ReactJS is harder to port due to its framework specific components. In the end, we decided to use ReactJS due to its modern approach to web development with components that increase modularity and development time.

Major design changes were also made regarding the choice of a "traditional" architecture for the server, such as Django, or a more flexible one like "Serverless on AWS". "Serverless on AWS" is a flexible service-based architecture that aims to remove the burden of server maintenance from developers. It includes AWS Lambda, Amazon API Gateway. AWS Cognito, and many more services. This is unlike Django in the sense that there is no need to manually host a server either. Instead, "lambda functions" are used to define the backend logic. These are essentially dedicated handlers to API requests, which allow for high scalability later on. Furthermore, it was found that there was great flexibility in the choice of the programming language with AWS compared to Django which could only offer the Python programming language. However, it does make it harder to make changes that are outside of the capabilities of AWS. "Serverless on AWS" was found to be the better choice for higher productivity and a more flexible architecture in general. It also had a much more supported service and was especially needed for the given time constraints we had. It also allowed more tasks also needed to be placed into the backend to allow for less resources on the client side.

As discussed in Section 4.4.1, revenue was also necessary to generate in order to maintain the application. Advertisements were considered but were found to negatively affect privacy, security, and the usability of the platform. Thus, subscriptions were found to be a better alternative. It would generate revenue but the user could now make a decision. It would not violate privacy and security would be easier to control without having to trust third-party trackers. The usability of the platform would remain the same such as how it was without advertisements.

Chapter 5 System Implementation (David Sambilay)

5.1 Implementation Overview

The backend follows a serverless framework and it is implemented in Amazon Web Services (AWS). There are a number of services used in order to fulfill Mellow's project design and requirements. AWS Lambda is the service used for processing all the events running in the backend. The runtime environment used for AWS Lambda is Node.js. One of the main functions of Lambda is that it contains the proper SQL queries and connections responsible for managing the database. The database service used is AWS RDS, and it uses PostgreSQL. It also handles the REST API requests and responses from AWS API Gateway. Finally, it is responsible for handling JWT tokens from Amazon Cognito. Cognito handles Mellow user authentication. AWS Lambda is dependent on the library "node-postgres" which is responsible for connecting to the database and writing SQL queries.

The frontend is implemented in React. There are libraries utilized in the frontend that are dependent on Mellow's implementation. The libraries used are React Router, React Query, and Material UI. React Router handles page routing and client and server-side routing. React Query handles state management, fetching data, and error handling. Material UI is a front-end component library used for creating simple components in Mellow, such as forms, buttons, lists, etc. These libraries were used to make development more efficient and to ensure a reliable product.

5.2 Implementation of Developed Solutions

The techniques used on the API implementation is by creating endpoints and methods that all require user authentication. Each method in the API has an AWS Cognito authorizer that needs to be present in order to send API requests. Users that are authenticated and logged in have access to this. Those without the proper credentials cannot use the API.

Since AWS Lambda is handling several different requests from the API, Lambda needs to have a way to identify what event is passed and it should be handled correctly. This is achieved through the use of a switch case statement that checks each case by the event's endpoint/path name and its method name (GET, POST, etc.)

5.3 Implementation Problems, Challenges, and Lesson Learned

Some of the major challenges faced in implementing Mellow is the AWS Lambda design. The challenge was how to verify the authenticity of a user's JWT token within the Lambda code. After verification, we need to retrieve the user's data as it is needed for our implementation. Attempts on writing lines of code for JWT verification were done but it was not successful. It was later discovered that we could have used the authorizer provided within the AWS API Gateway service that automatically handles it for us. Not only that it verifies the user, but it also provides the user's data. A lesson learned from this challenge is the importance of checking services that could possibly handle a task for you instead of manually doing it in code. We did not realize API Gateway offers the functionality we needed. A lesson learned is that it is important to learn more about cloud services and what it can offer before trying to manually do it on code.

Chapter 6 Tools and Standards (Albert Lu, David Sambilay)

6.1. Tools Used

The software tools specifically used as part of the final application are listed below. The reasons and choice are further explained too.

Software Technology	**Description / How it is used**	**Why it is selected**
Material UI	Frontend library	Easy-to-use prebuilt components to easily create UI for the web app. It is used in React.
React Query	Data fetching library	All-in-one library that handles fetching and updating state of the web app on the front-end side. It is used in React.
JWT (Json Web Token)	Security for data encryption/ decryption	Simple way to provide user authorization and data security. Technology used along with AWS cloud.
React Router Dom	Configuring routes	Common React library for router configuration. Easy to use and all team members are familiar with it.
Amazon Cognito Identity	Middleware Communication with Cognito service for frontend	Library suggested by AWS when handling Cognito functions in the frontend. Used in the login, register, and sign out functionality in React frontend
PostgreSQL	Database	Relational database that is supported on AWS services and is used to store user data persistently.

The next section lists more specific elements of the Amazon Web Services architecture on the backend.

Software Technology	**Description / How it is used**	**Why it is selected**
Amazon Web Services (AWS)	Backend Architecture	AWS is used as the backend server where requests are processed between the frontend and database. It consists of Cognito, Amazon RDS (Database), and more.
AWS Lambda	Related to AWS, but is specifically the request handling part	Service is required to handle requests from services such as API Gateway, Cognito, and RDS.
AWS API Gateway	REST API deployed in AWS	Service is used to create API calls for requests that will be called whenever the user interacts with the frontend.
AWS Cognito	Authentication Service	Service is required so that users registered are verified and their user credentials are saved.
Amazon RDS	Database Connection	It is necessary because the data is stored in the cloud rather than locally and administrators can have easy access to the database in real time.

In addition to the key software technologies listed above, some tools more specific to development are listed below.

Software Technology	**Description / How it is used**	**Why it is selected**
Operating System	MacOS / Windows / Linux	Windows was used due to familiarity and because it is a highly supported environment for our development tools.
Postman	API Request Development	It is used to send and manage API requests in an easy-to-use interface. This allows us to

	Environment	perform tests on our API to ensure it is working fully.
PgAdmin	Database Management Environment	PgAdmin is used to easily manage PostgreSQL databases. It helps to add, delete, and edit tables and rows to quickly test new features in our application. It also helps us to test that everything is working.
Visual Studio Code (VS Code)	Code Editor	Visual Studio Code was our code editor of choice because of how easy it is to use. It helped to speed up our development of the frontend.
GitHub	Version Control System (VCS)	GitHub was chosen due to its popularity and our need of a central repository to store all our code. This allows us to be confident our sources are saved and makes it easy for us to work together by pulling or pushing any changes. We can see what each other changed.

6.2. Standards

One of the main standards used in our project was the **Representational State Transfer (REST)** architecture style for our API. This was critical to making the connection between the frontend and backend parts of our application. It allowed us to define our API in an easy-to-use and consistent way. It was used specifically with our choice of using "GET", "POST", "DELETE", and "PATCH" requests based on what functionality was being used. For example, a "GET" request would be a request from the client for simply fetching data from the backend. A "POST" request would also be a request from the client to tell the backend to change some data, such as when adding or removing tasks. "DELETE" was for when deleting a task or category. "PATCH" was for when updating a task. This type of consistency allowed our API to be easy to understand.

Another standard used was **JavaScript Object Notation Web Token (JWT)** for authentication of users. More specifically, it is an open standard called RFC 7519 (https://datatracker.ietf.org/doc/html/rfc7519) that is meant to be secure, efficient, and convenient for storing user authentication data. It can be easily passed between server and client. There is no need to contact the server to verify the token. This token is passed on every request so that the server knows which user to edit data on behalf of on the backend side. This is made possible and not resource intensive because JWT is easy to store and small enough. It is also easy to parse and decode because it uses JSON (Javascript Object Notation) that is widely used and thus many libraries support it efficiently.

Hypertext Transfer Protocol (HTTP) is another standard that was used extensively. It is how the server communicates with the client. It defines the status codes used, such as 200 and 504. These were used to communicate whether something was successful or not. It was used within the browsers we used to test with.

Chapter 7 Testing and Experiment (Kha Nguyen)

7.1 Testing and Experiment Scope

Mellow's testing and experiment scope involves thorough testing to ensure that the project attains the highest possible level of quality. Our testing objective is to focus on the aspects of Mellow that creates a friendly UI that is easy to navigate. We want to make sure that the entire system is fully tested and free of defects. We will use manual testing techniques to achieve our testing objectives. We will focus on testing the key features of Mellow such as UI responsiveness, authentication, data format, etc. These areas of testing are explained in detail on the table below.

Testing Items	**Explanation**
UI Responsiveness	Web adjustment to different screen sizes and viewports
UI Layout	Correct data appear in the right fields
Frontend Navigation	Routing correctly
Web Performance	Response time from each request
Authentication	Authenticate/ deny users with username and password
Authorization	User access and modify the right resources
Database	Data is collected and removed
Data format	Format enter from the frontend fit with the

	format in the backend
Database security	Avoid incorrect data into the database. Also important to prevent malicious actors.

The manual testing will be performed by the developers of Mellow. When testing is complete, the recorded experiment results are displayed and the analysis.

7.2 Testing and Experiment Approach

The following sections below highlight the specific cases that are tested in our approach. It covers the categories specified in the table in **Section 7.1**.

UI Responsiveness

Case 1: Login page when full screen size and after shrinking

Case 2: User page when full screen size and after shrinking

UI Layout

Case 1: After adding category, the added category is shown under the category form

Case 2: After adding task, the added task is shown under the task view

Frontend Navigation

Case 1:Routing from login page and sign up page

Case 2: Routing from login page and user page after authentication

Case 3: From user page to Log in page after log out

Web Performance

Case 1 : Performance of authentication user then redirecting to user page

Case 2 : Performance of registering a user

Case 3: Performance of adding task and showing on the frontend

Case 4: Performance of adding category

Authentication

Case 1: Deny if enter wrong credentials

Case 2: Authenticate if user enter their right credentials (username/password) and move on to their own data

Case 3: If logging in (already authenticated), able to log out

Authorization

Case 1: Users are able to access their own resource

Case 2: Users are able to update their resources (task, category…)

Case 3: Tasks cannot be added without the token credential

Database

Information enter by users on the frontend is stored in the Postgresql database

Data format

When creating the task in the frontend, the values from the task should follow the format that is expected from the backend. This can be tested in Postman and verified in the Postgres database.

Case 1: User successfully adds task with the correct format

Database security

Attributes of an entity in the database have a specific format (domain). For instance, the attribute “name” that has a domain of varchar can only have varchar. It is important to avoid adding incorrect data into the database. This can be tested in Postman and verified in the Postgres database.

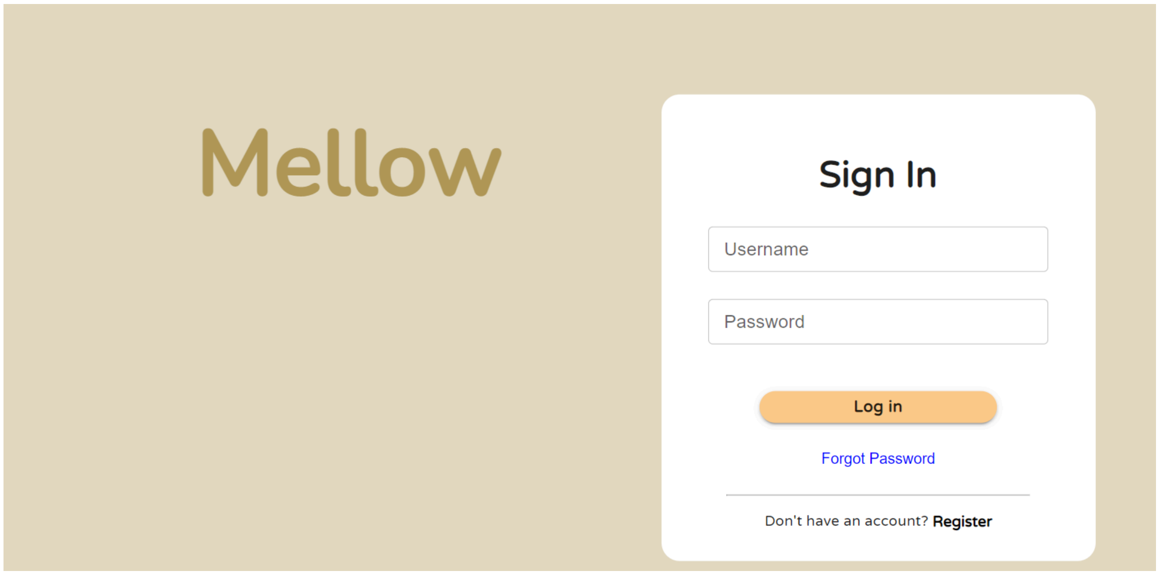
Case 1: User does not add any incorrect data in the database.

7.3 Testing and Experiment Results and Analysis

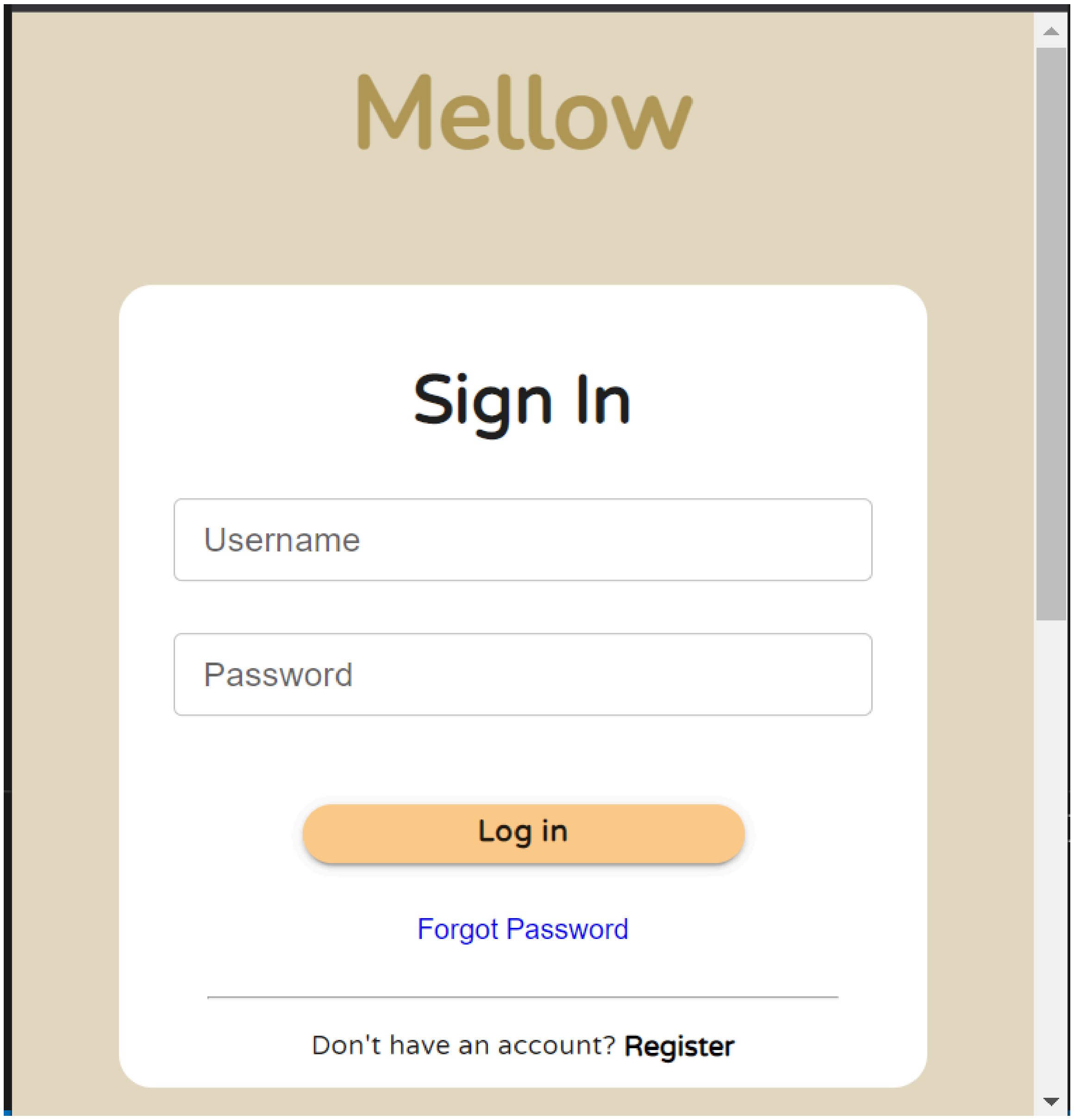
This section describes the results and evaluation of the testing approach detailed in the earlier sections. Images are used as a guide to explaining the results.

UI Responsiveness results

Case 1:

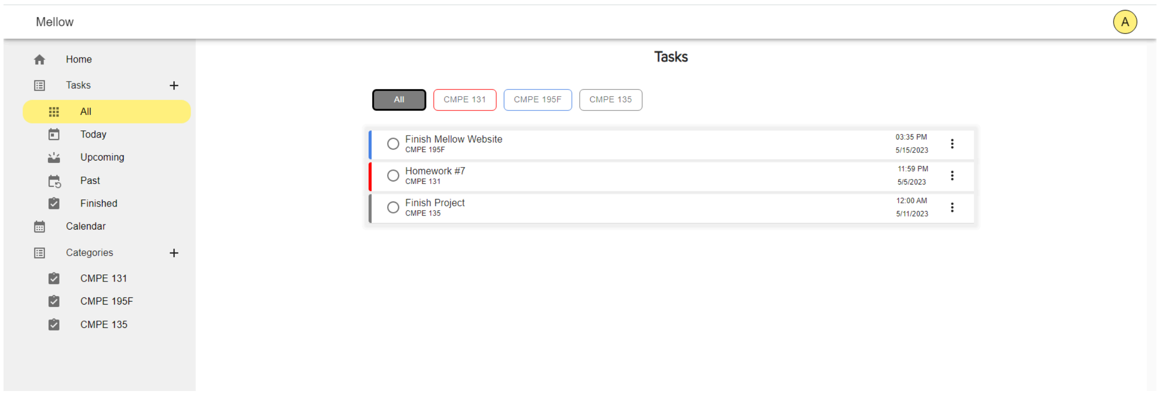


Full screen Login page

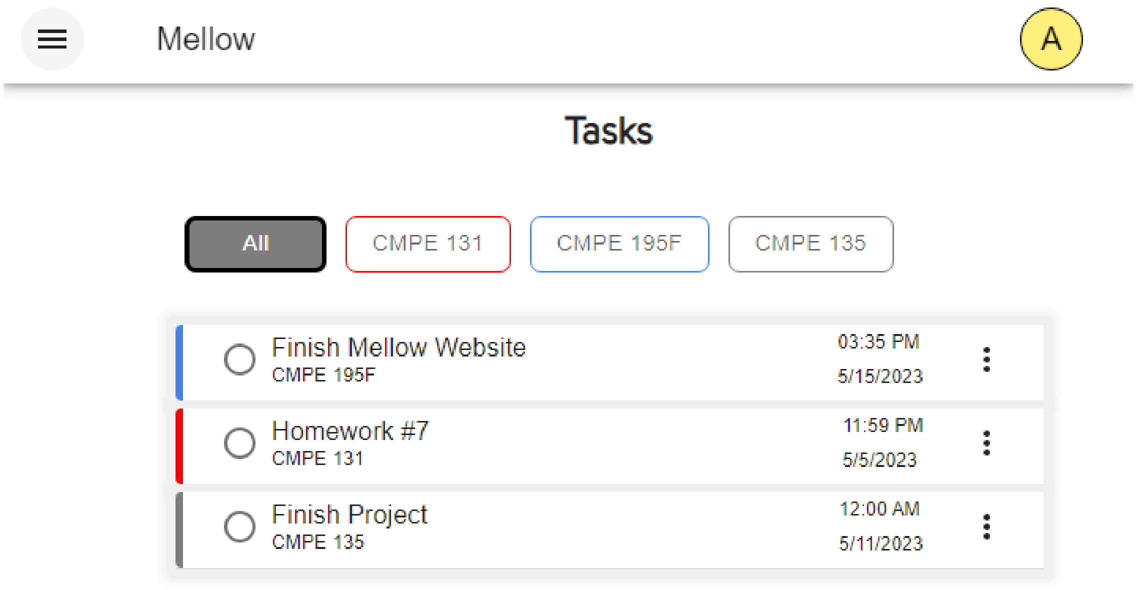


Log in page if width page is below 1200px

Case 2:



User page full screen

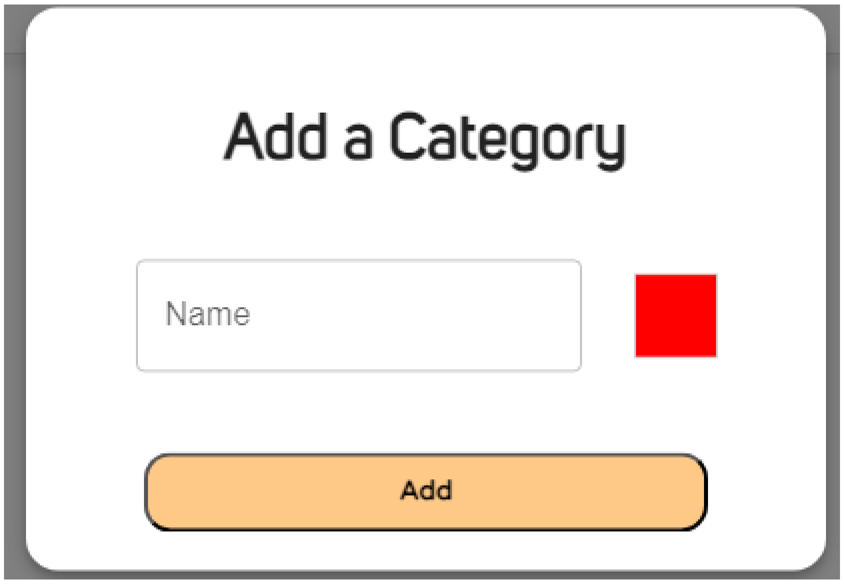


User page after shrinking

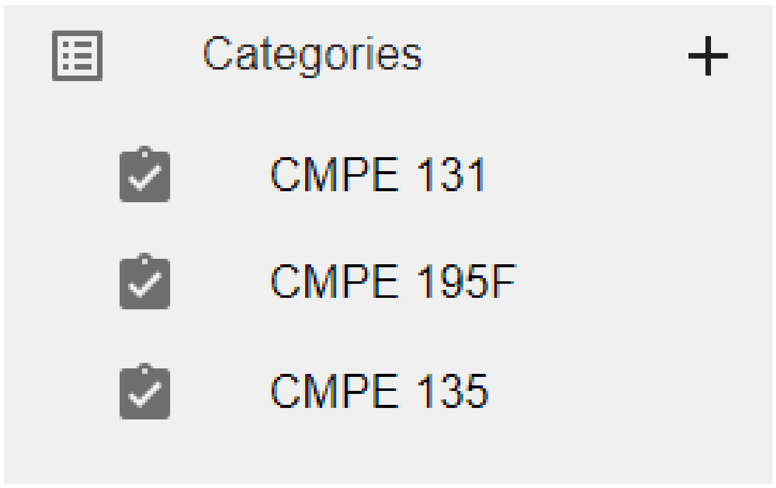
UI Layout

Case 1:

Adding category form:

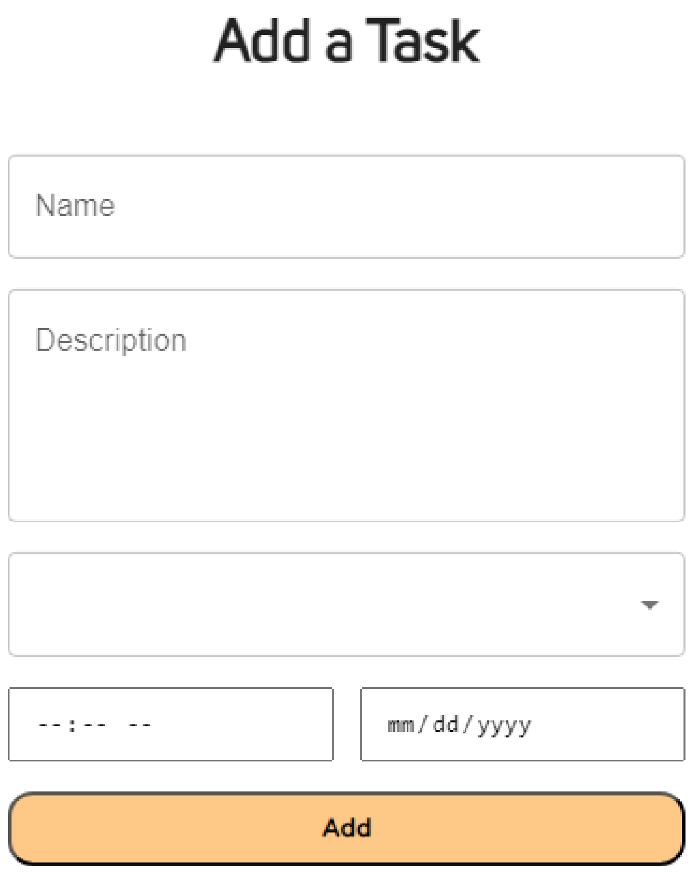


Successfully shows category after adding

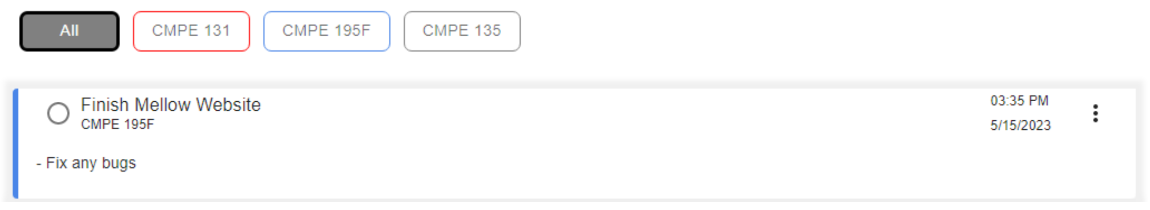


Case 2:

Adding task form



Successfully showing tasks after adding



Finish Mellow Website
CMPE 195F
- Fix any bugs
03:35 PM
5/15/2023

Frontend Navigation

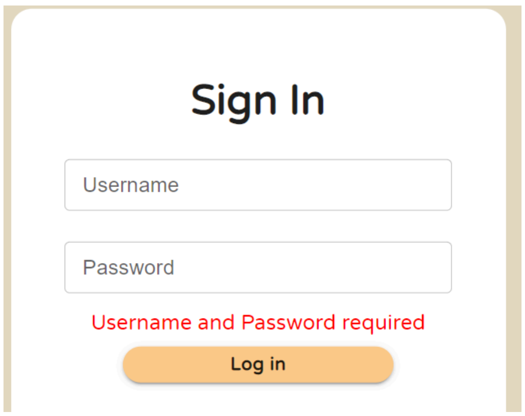
Cases	Navigation	Result
1	From Login to Sign Up page after user click on register button	Success
2	From Sign In to User page after user authenticate correctly	Success
3	From user page to Log in page	Success

Web Performance

Cases	Functionality	Average performance (seconds)
1	Authenticate user and redirect to user page	1.7
2	Register a user	1.2
3	Add task and shown on the frontend	1
4	Add category	0.8

Authentication

Case 1: Deny with incorrect username/password

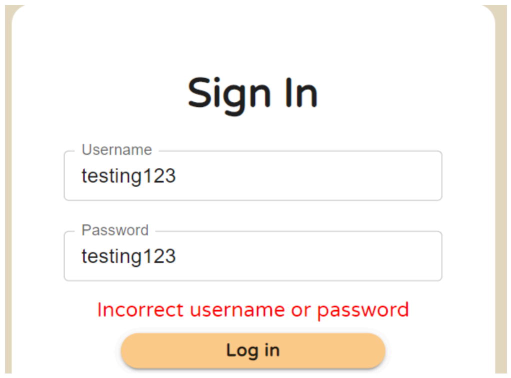

Sign In

Username

Password

Username and Password required

Log in


Sign In

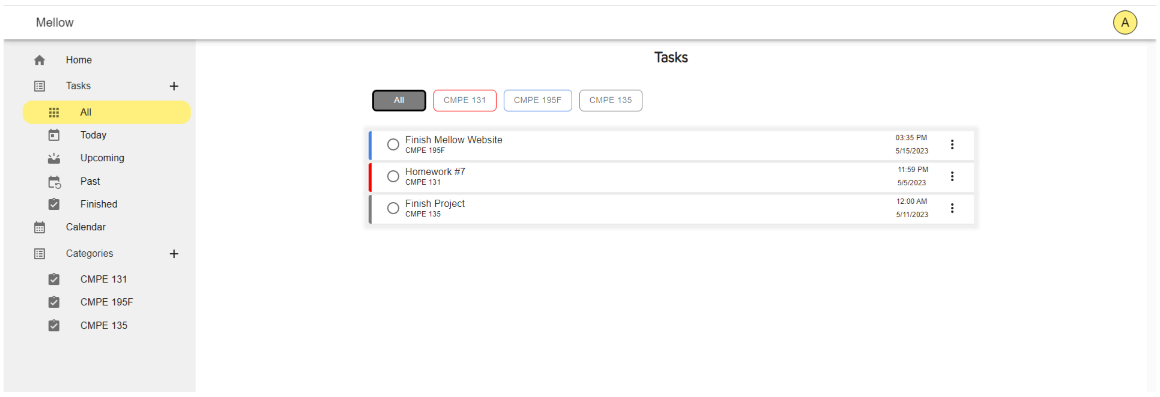
Username
testing123

Password
testing123

Incorrect username or password

Log in

Case 2: Successfully log in user if enter credentials correctly


Mellow

A

Home
Tasks +
All
Today
Upcoming
Past
Finished
Calendar
Categories +
CMPE 131
CMPE 195F
CMPE 135

Tasks

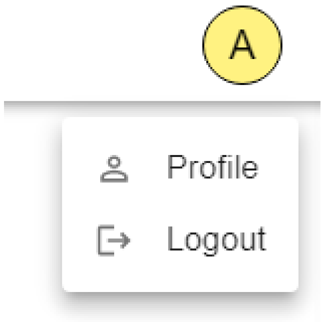
All | CMPE 131 | CMPE 195F | CMPE 135

Finish Mellow Website
CMPE 195F
03:35 PM
5/15/2023

Homework #7
CMPE 131
11:59 PM
5/5/2023

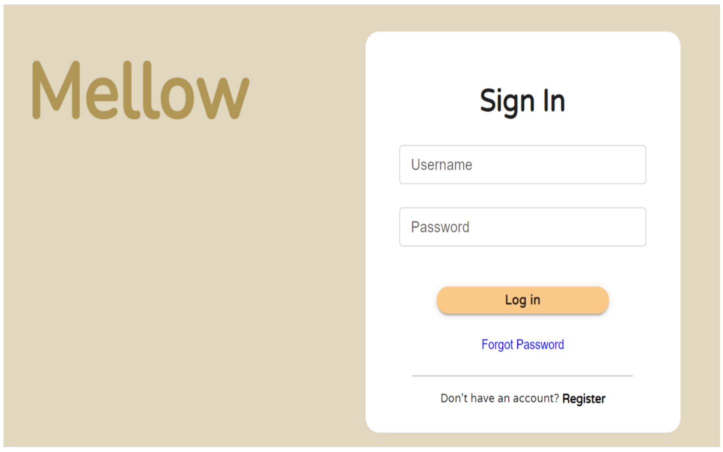
Finish Project
CMPE 135
12:00 AM
5/11/2023

Case 3: Successfully log out


A

Profile

Logout


Mellow

Sign In

Username

Password

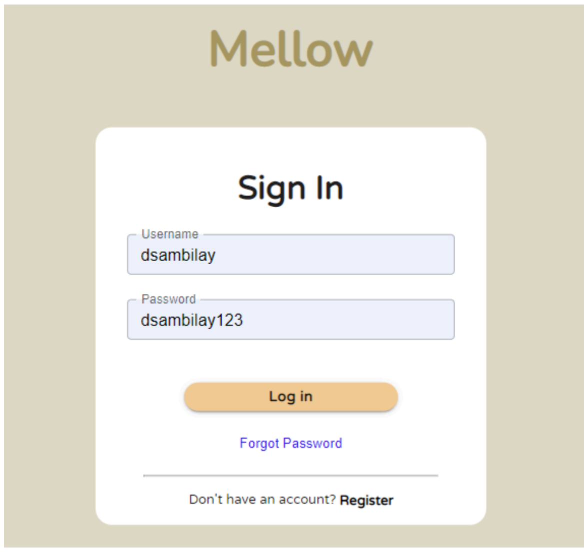
Log in

Forgot Password

Don't have an account? **Register**

Authorization

Case 1: Logging in

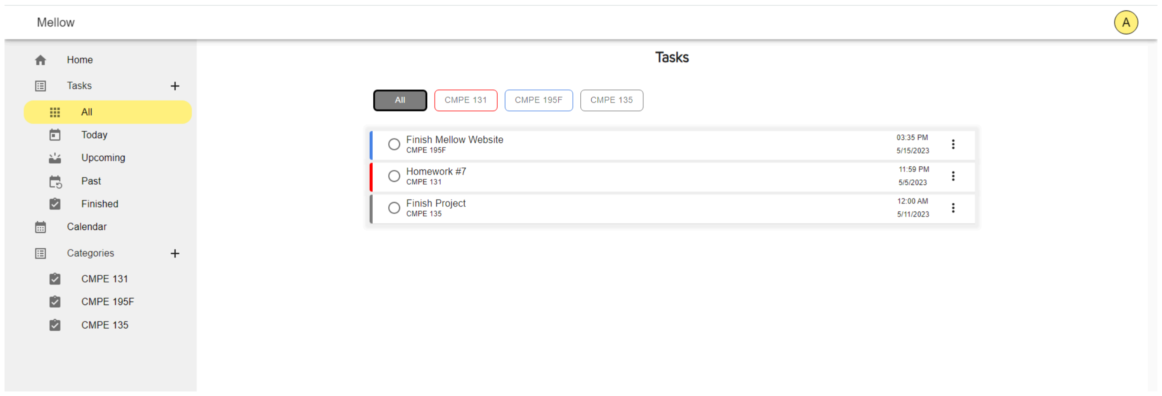


Mellow

Sign In

Username: dsambilay

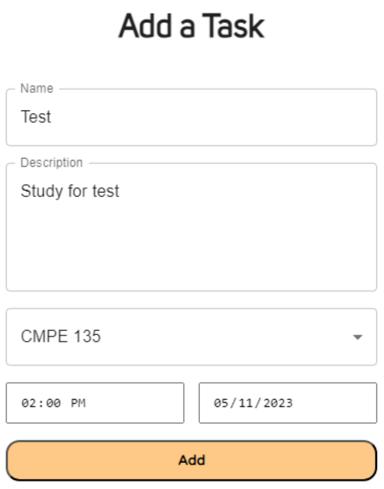
Password: dsambilay123



Finish Mellow Website	CMPE 195F	03:35 PM 5/15/2023
Homework #7	CMPE 131	11:59 PM 5/5/2023
Finish Project	CMPE 135	12:00 AM 5/11/2023

Data and resources of the user is retrieved and shown after logging in

Case 2: Update



Add a Task

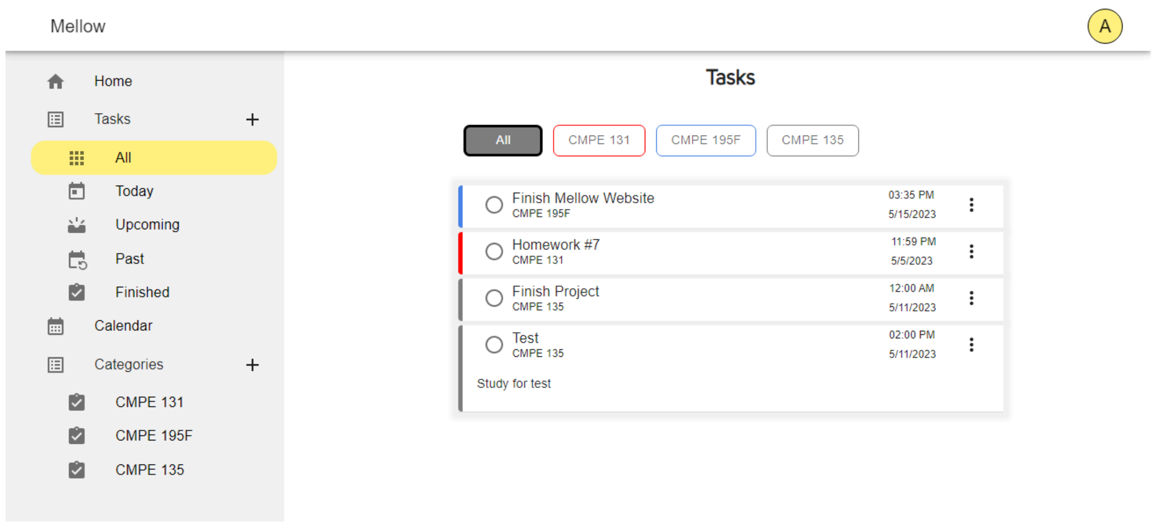
Name: Test

Description: Study for test

CMPE 135

02:00 PM | 05/11/2023

Add



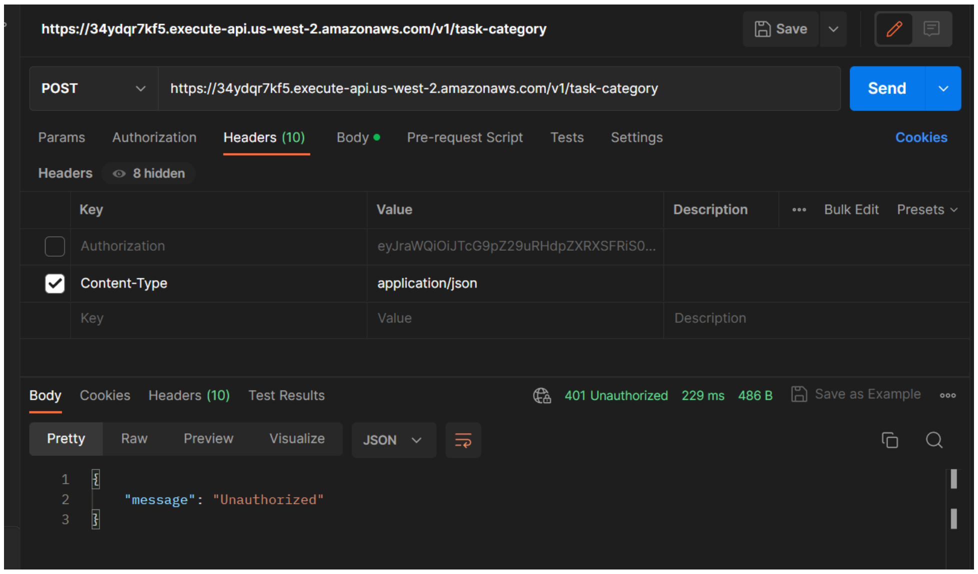
Mellow

Tasks

Finish Mellow Website CMPE 195F	03:35 PM 5/15/2023	
Homework #7 CMPE 131	11:59 PM 5/5/2023	
Finish Project CMPE 135	12:00 AM 5/11/2023	
Test CMPE 135 Study for test	02:00 PM 5/11/2023	

Data and resources of each user is retrieved and shown after user post and update the information

Case 3: Deny



https://34ydqr7kf5.execute-api.us-west-2.amazonaws.com/v1/task-category

POST https://34ydqr7kf5.execute-api.us-west-2.amazonaws.com/v1/task-category

Headers (10)

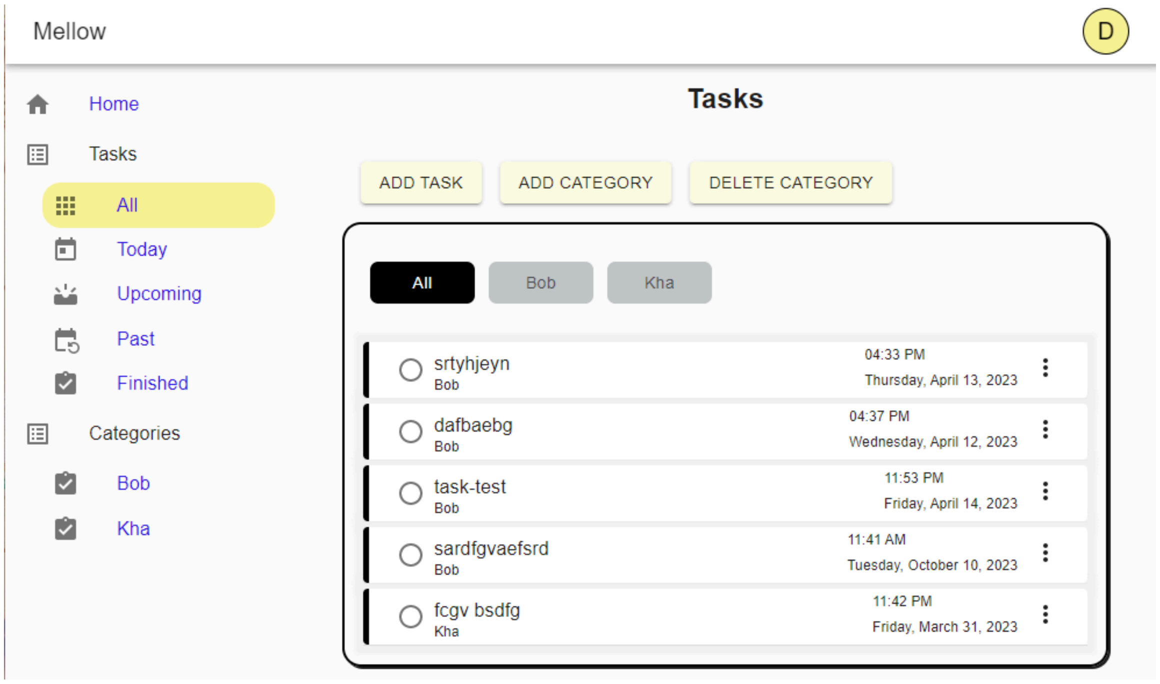
	Key	Value	Description
☐	Authorization	eyJraWQiOiJTcG9pZ29uRHdpZXRXSFRiS0...	
✓	Content-Type	application/json	

401 Unauthorized 229 ms 486 B

```
{
    "message": "Unauthorized"
}
```

Tasks cannot be added without the token credential. The API does not allow it and throws an error.

Database

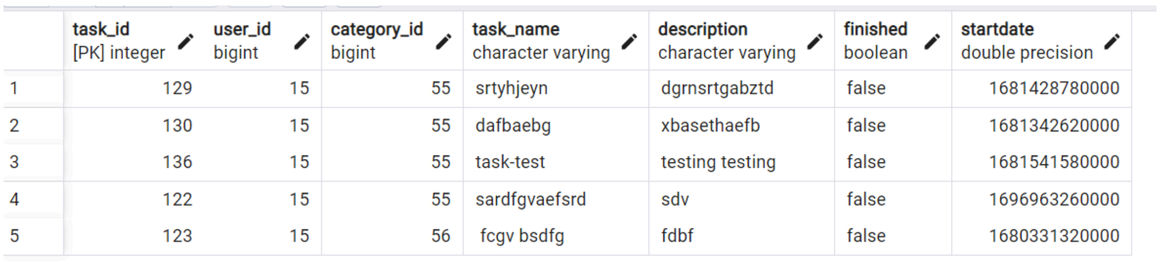


Mellow

D

Tasks

Task	Category	Time	Date
srtyhjeyn	Bob	04:33 PM	Thursday, April 13, 2023
dafbaebg	Bob	04:37 PM	Wednesday, April 12, 2023
task-test	Bob	11:53 PM	Friday, April 14, 2023
sardfgvaefsrd	Bob	11:41 AM	Tuesday, October 10, 2023
fcgv bsdfg	Kha	11:42 PM	Friday, March 31, 2023

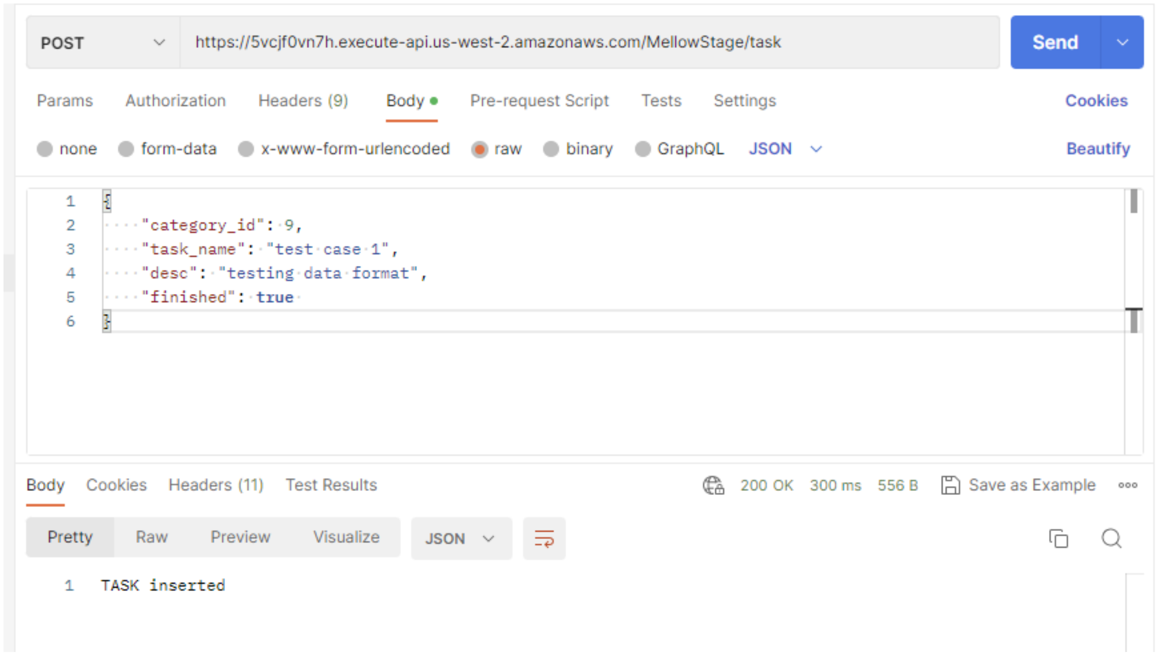


	task_id [PK] integer	user_id bigint	category_id bigint	task_name character varying	description character varying	finished boolean	startdate double precision
1	129	15	55	srtyhjeyn	dgrnsrtgabztd	false	1681428780000
2	130	15	55	dafbaebg	xbasethaefb	false	1681342620000
3	136	15	55	task-test	testing testing	false	1681541580000
4	122	15	55	sardfgvaefsrd	sdv	false	1696963260000
5	123	15	56	fcgv bsdfg	fdbf	false	1680331320000

Task information is shown in Postgresql database

Data format

Case 1:

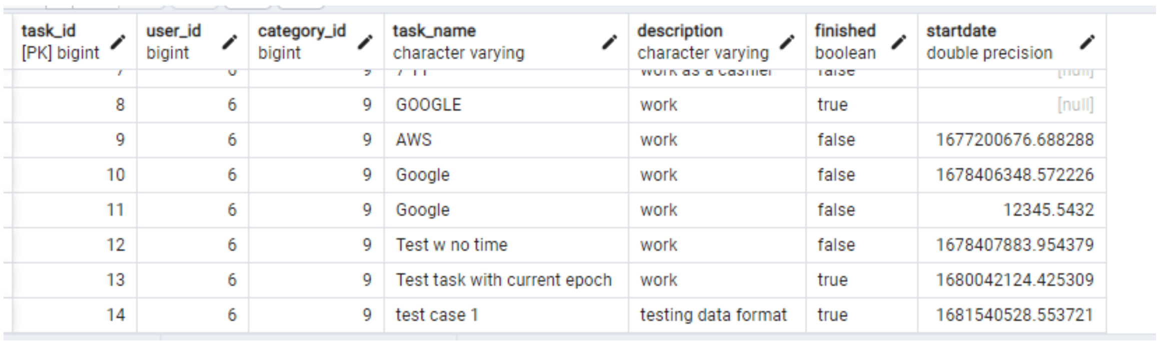


https://5vcjf0vn7h.execute-api.us-west-2.amazonaws.com/MellowStage/task

```
{
    "category_id": 9,
    "task_name": "test case 1",
    "desc": "testing data format",
    "finished": true
}
```

```
TASK inserted
```

The request sent was able to create a 200 OK response. We can check the database if it was inserted correctly as expected.

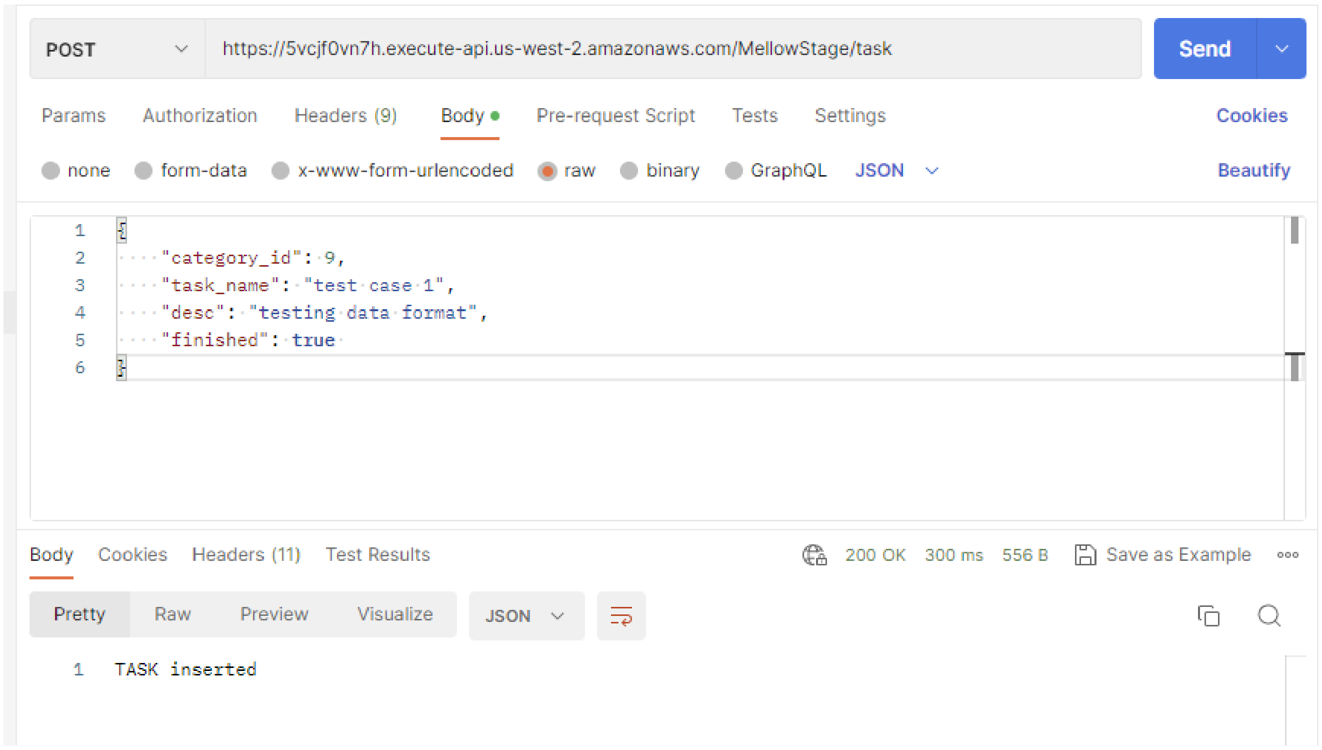


task_id [PK] bigint	user_id bigint	category_id bigint	task_name character varying	description character varying	finished boolean	startdate double precision
8	6	9	GOOGLE	work	true	[null]
9	6	9	AWS	work	false	1677200676.688288
10	6	9	Google	work	false	1678406348.572226
11	6	9	Google	work	false	12345.5432
12	6	9	Test w no time	work	false	1678407883.954379
13	6	9	Test task with current epoch	work	true	1680042124.425309
14	6	9	test case 1	testing data format	true	1681540528.553721

The raw JSON in Postman (frontend) matches the expected values we have in the backend/database.

Database security

Case 1:

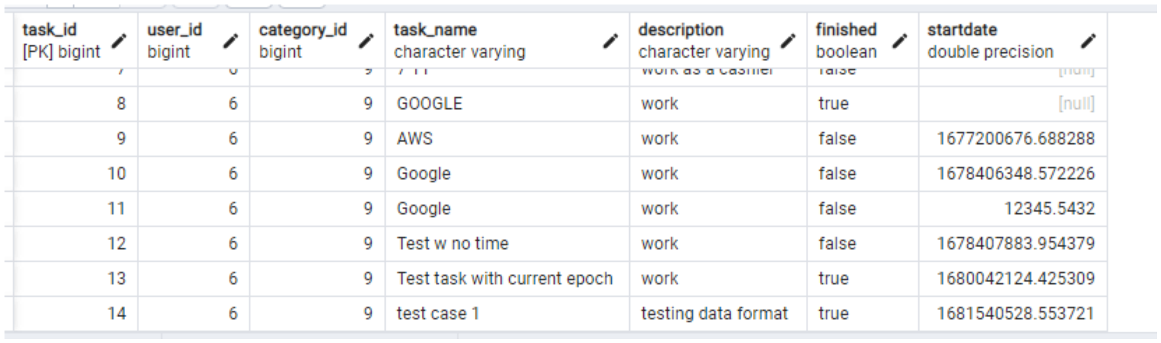


https://5vcjf0vn7h.execute-api.us-west-2.amazonaws.com/MellowStage/task

```
{
    "category_id": 9,
    "task_name": "test case 1",
    "desc": "testing data format",
    "finished": true
}
```

```
TASK inserted
```

The request sent was able to create a 200 OK response. We see that the “category_id” value is an integer and the “task_name” is a varchar. We can confirm in the database if those values were accepted and it did not cause any errors.



task_id [PK] bigint	user_id bigint	category_id bigint	task_name character varying	description character varying	finished boolean	startdate double precision
8	6	9	GOOGLE	work	true	[null]
9	6	9	AWS	work	false	1677200676.688288
10	6	9	Google	work	false	1678406348.572226
11	6	9	Google	work	false	12345.5432
12	6	9	Test w no time	work	false	1678407883.954379
13	6	9	Test task with current epoch	work	true	1680042124.425309
14	6	9	test case 1	testing data format	true	1681540528.553721

In the last row, the “category_id” value is indeed an integer and the “task_name” is a varchar.

Chapter 8 Conclusion & Future Work (Kha Nguyen & David Sambilay)

Overall, our application Mellow provides a solution for everyone to organize their tasks and create effective schedules. Our goal was to create an easy-to-use experience and aimed at letting users "do more with less", rather than spend time figuring out our application. The main stack technologies we use to develop the software are React Framework for frontend and AWS Cloud Services for backend.

Throughout the development of our project, we placed a great deal of emphasis on following best practices and principles of building high-quality software, specifically a web application. We utilized agile methodologies, and broke down our development process into small sprints. We prioritized communication and collaboration. Throughout the initial and development phase of our project, we schedule meetings once or twice a week to discuss our progress and also to create a space for brainstorming and planning.

We also followed established coding standards to ensure maintainability of our codebase. These standards include writing consistent style of code shown in official documentations, using readable and proper variable names, and adding meaningful comments in our codebase for readability and for whenever we need a quick recap of the written code. We also followed basic testing techniques such as unit testing. The testing techniques were all done manually. Through our commitment in following these practices and principles, we were able to build a web application that is maintainable and easy to use.

As for the future of Mellow, we planned to extend Mellow's features onto the mobile platform as an application on IOS and Android since it currently only serves on the web. As more users rely on their mobile devices in their daily lives, it is convenient that we make Mellow more accessible. Furthermore, we plan to add interesting features to provide more utility for our users. This plan includes several new features such as sharing tasks between users for collaboration, a reward system to boost motivation and an alert system for better remiding. Overall, we are excited about Mellow's potential to continue growing and serve more users. We look forward to working together and making the best version of Mellow for our users.

References

Amazon Web Services. (1983). AWS Lambda Developer Guide. AWS Docs. Retrieved March 3, 2023, from https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

Amazon Web Services. (n.d.). Serverless microservices. AWS Docs. Retrieved March 3, 2023, from https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/serverless-microservices.html

Amazon Web Services. (n.d.). Amazon Cognito Developer Guide. AWS Docs. Retrieved March 3, 2023, from https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-scenarios.html

Amazon Web Services. (n.d.). Amazon API Gateway Developer Guide. AWS Docs. Retrieved March 3, 2023, from https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

Levin, & Levin, M. (2019). Managing Ideas, People, and Projects: Organizational Tools and Strategies for Researchers. iScience, 20, 278–291. https://doi.org/10.1016/j.isci.2019.09.017

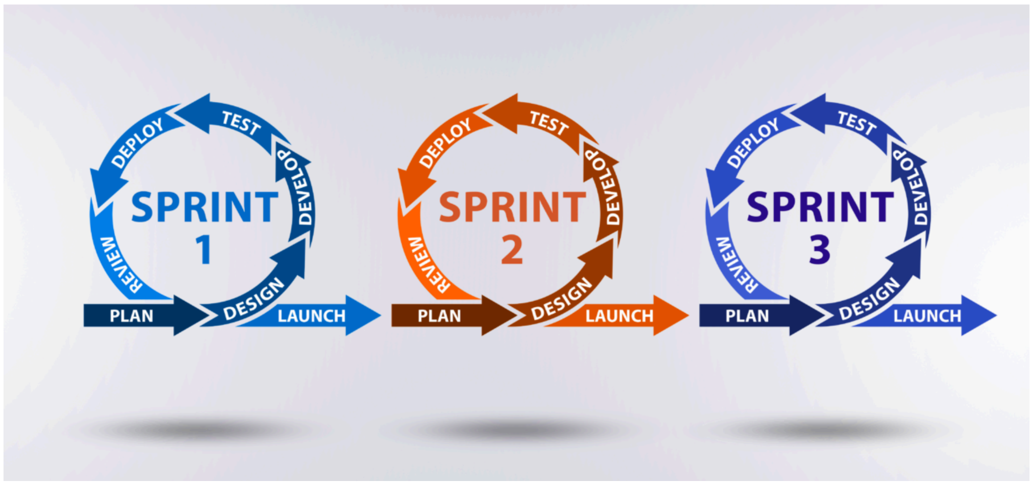
Meta Platforms, Inc. (n.d.). A JavaScript library for building user interfaces. Retrieved March 3, 2023, from https://reactjs.org/

Schrager, & Sadowski, E. (2016). Getting More Done: Strategies to Increase Scholarly Productivity. Journal of Graduate Medical Education, 8(1), 10–13. https://doi.org/10.4300/JGME-D-15-00165.1

Jones, M. B., Bradley, J., &amp; Sakimura, N. (2015, May 19). JSON Web Token (JWT). IETF Datatracker. Retrieved April 14, 2023, from https://datatracker.ietf.org/doc/html/rfc7519

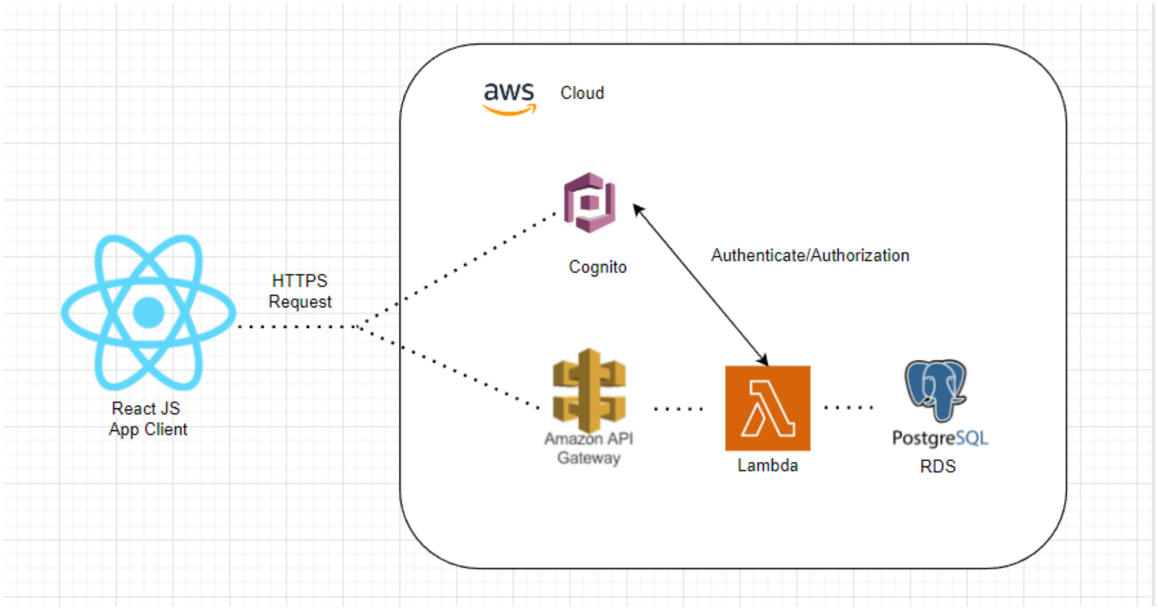
Appendix

Appendix 1 Agile software development methodology



Source: https://kruschecompany.com/agile-software-development/

Appendix 2 Architecture Tool



aws Cloud
Cognito
Authenticate/Authorization
HTTPS Request
React JS App Client
Amazon API Gateway
Lambda
PostgreSQL RDS