Time Blocker App

Jaecob Dobler

Caleb Leullen

Chris Wilcox

Contents

Customer Problem Statement

Problem Statement:

In the modern world, it can be difficult for people to manage their time efficiently and achieve their goals. Typically, time blocking is done with pencil and paper, which results in a very manual process where the individual must decide what they must do during each period of time and decide what to do next. However, with a modern web application this process could become less time-consuming and thought intensive. By allowing the system to take the guesswork out of what an individual should do when, they should be able to achieve more of their goals.

Glossary of Terms:

- **Web Application**
  - Web applications are a modern type of website that can be written in many different languages.
- **React**
  - A JavaScript library developed by meta (formerly Facebook) with focus on reusable components.
- **Next.js**
  - A full stack framework built on top of react. This is what we will use for our application to allow for easy and fast communication with the database.
- **Task**
  - A task in our system is something that a user can create, and we store. They can name a task whatever they like and give it a description if they want to.
- **Schedule**
  - The schedule within our application will be a visual representation of a user's day.

System Requirements

Functional Requirements:

No.	Priority Weight	Description
REQ-1	High	Application should allow users to create an account
REQ-2	High	Application should allow user to create new tasks; tasks should contain a name, time they need to be completed by, and an expected duration. The task could optionally contain a description.
REQ-3	High	A user should be able to view and manage their tasks.
REQ-4	High	The system should automatically generate/modify a time block schedule based on the user's tasks.
REQ-5	Medium	Tasks should have categories that are stored in the database
REQ-6	Low	Users should be able to invite other users to a shared task.
REQ-7	Medium	Tasks should be able to have dependencies (other tasks that depend on it).
REQ-8	High	Tasks should have an option for fixed times. (Similar to a calendar event)
REQ-9	Low	Option to import selected google calendar events as fixed time items.

Nonfunctional Requirements:

Flexibility	Application will function across multiple platforms due to it being web based.
Security	The application will need to securely keep user data on the server.
Maintainability	The application and server will need to be maintained over the life of the product to ensure usability.

User Interface Requirements

The user interface will begin with a login screen. The user will be able to use different methods to login to the service.

After the user is logged in, they will come to the main interface page. This will show the current date and what is scheduled for that day. The user will also be able to show different dates as well.

The main interface will have a function that will allow the user to input new events into the calendar.

Plan of Work:

Weeks 1 – 2:

- Create project proposal.
- Set up development environment.
- Complete

Weeks 3-4:

- Create problem statements and system requirements.
- Complete

Weeks 5-8:

- Initialize Solito project.
- Create interface and connection to the database.
- Gather requirements for backend and implement.

Weeks 9-12:

- Create login flow.
- Create authentication processes.
- Create the landing page.

Weeks 13-15:

- Development of tasks and the daily schedule.
- Develop backend to generate dynamic schedules based on user input.
- Record the final demo to showcase the app’s capabilities.

Functional Requirements

Stakeholders

Project Designers

Jaecob Dobler

Caleb Leuellen

Chris Wilcox

Developers

Jaecob Dobler

Caleb Leuellen

Chris Wilcox

Managers

Jaecob Dobler

Caleb Leuellen

Chris Wilcox

Sponsors

Professor

Users

Actors and Goals

User – This actor can login to the service, create new calendar items, view calendar items, and view events for the day/week/month.

System – This actor will allow a user to create an account and interact with the UI.

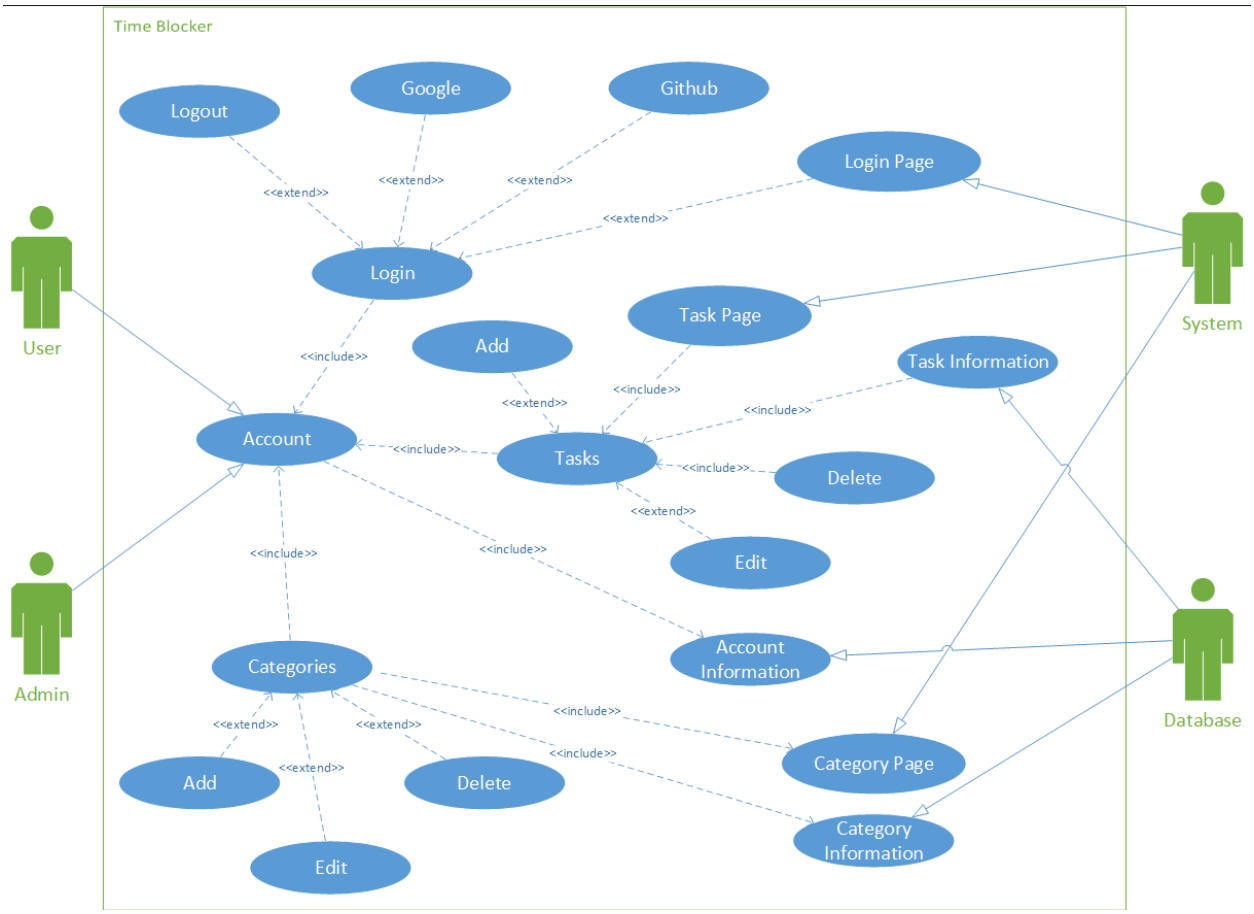
Database – This actor will be responsible for storing all the information input into the system.

Admin – This user will be able to assist a user with their account and with interacting with the UI.

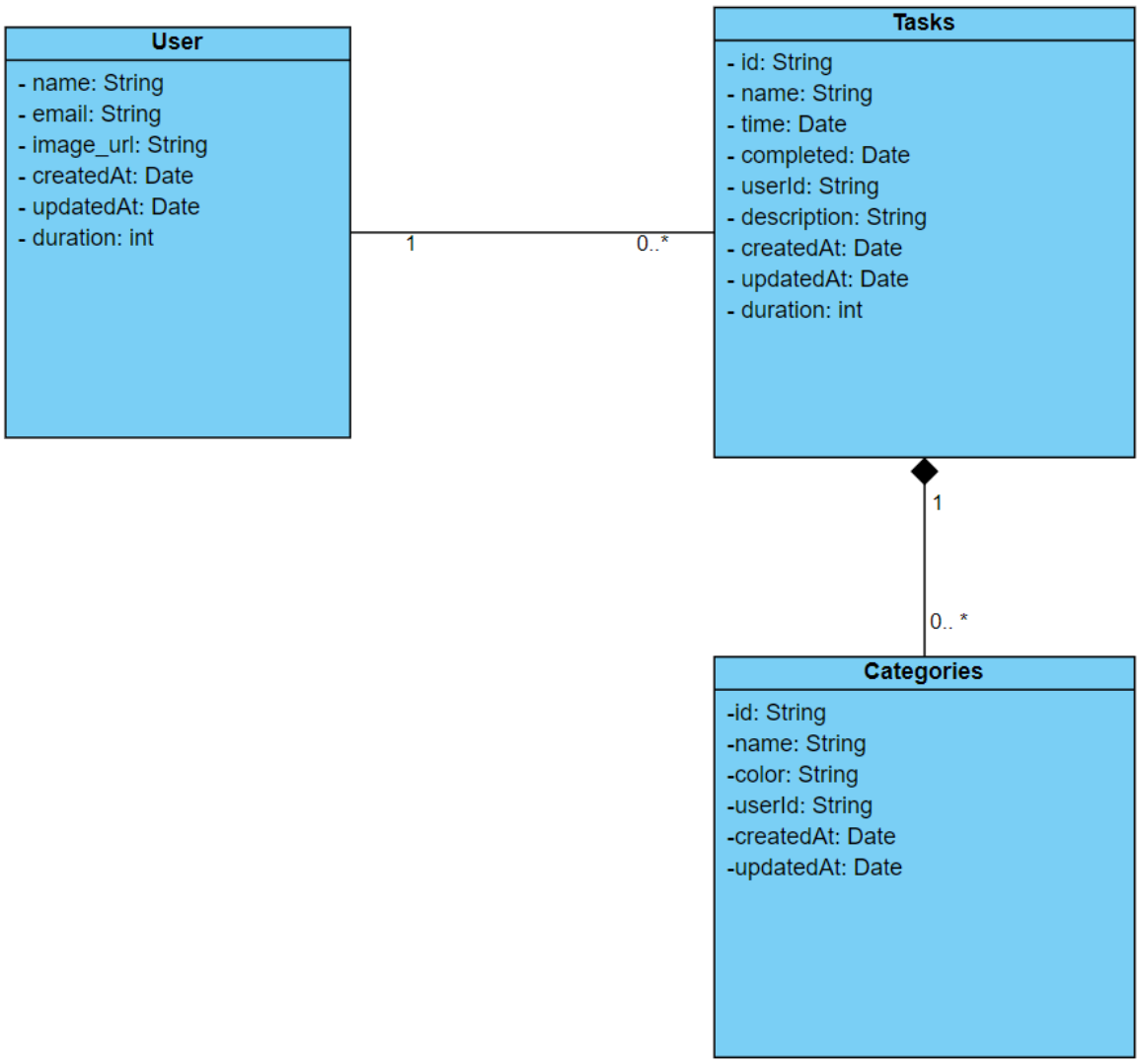
- **Use Cases**
- Create Tasks
- Update Tasks
- Delete Tasks

- Create Categories
- Update Categories
- Delete Categories
- Login with 0Auth2
    - Google
    - GitHub
- Edit profile
- View Pages
    - Dashboard
    - Report
    - View Report
    - Manage Tasks

Use Case Diagram



Class Diagram



Sequence and Activity Diagrams

User logs in with 0 auth

1. Check if a user is already logged in, if so reroute them to the dashboard.
2. When the user clicks login and grants access to their account, the database is checked to see if an account already exists.
3. If the account exists, it logs in the user to that account. If the account does not exist it will create a new account in the database.

User creates a task

1. User fills out task form, form validated on client.
2. Form data is sent to API and added into database. Includes(categories)

User creates a category

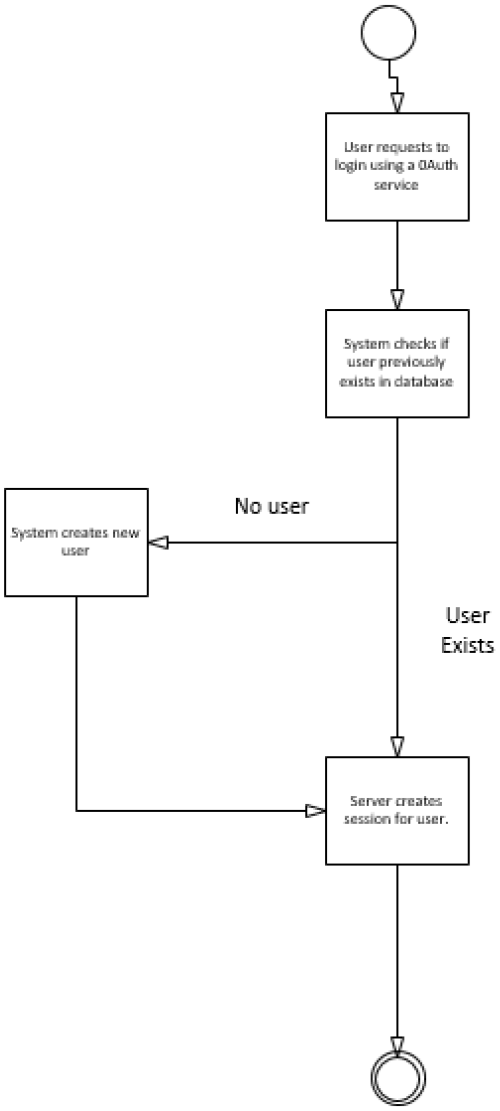
1. User fills out category form, form validated on client.
2. Form data is sent to API and added into database.

User deletes a task

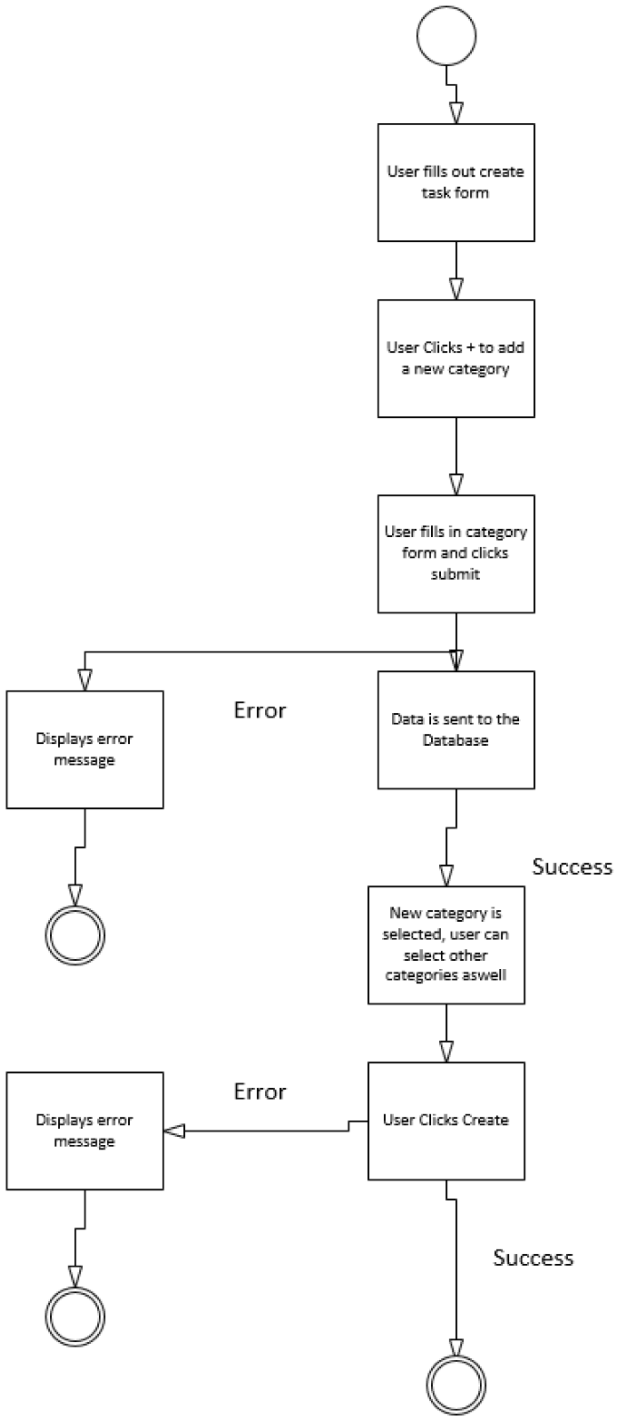
1. User pushes delete on task that they would like to remove.
2. The index of the mapped object is used to determine which task to remove.
3. The task ID is sent to the backend to remove from the database.

Activity Diagrams:

User Logs in with 0Auth

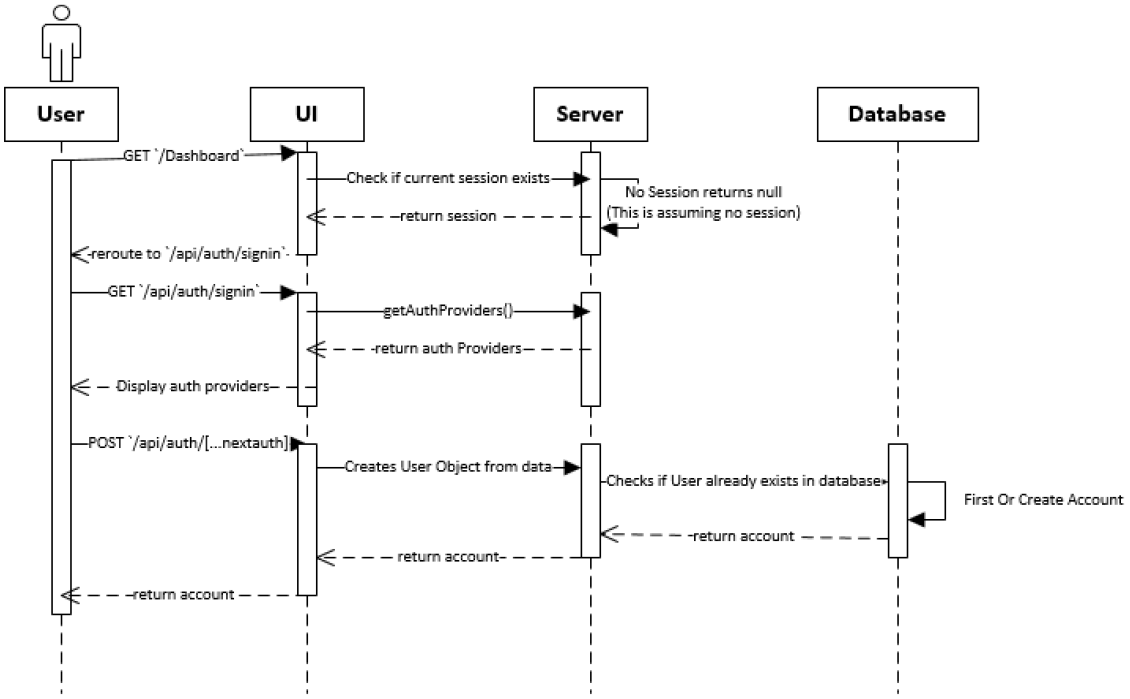


User creates a task with a new category

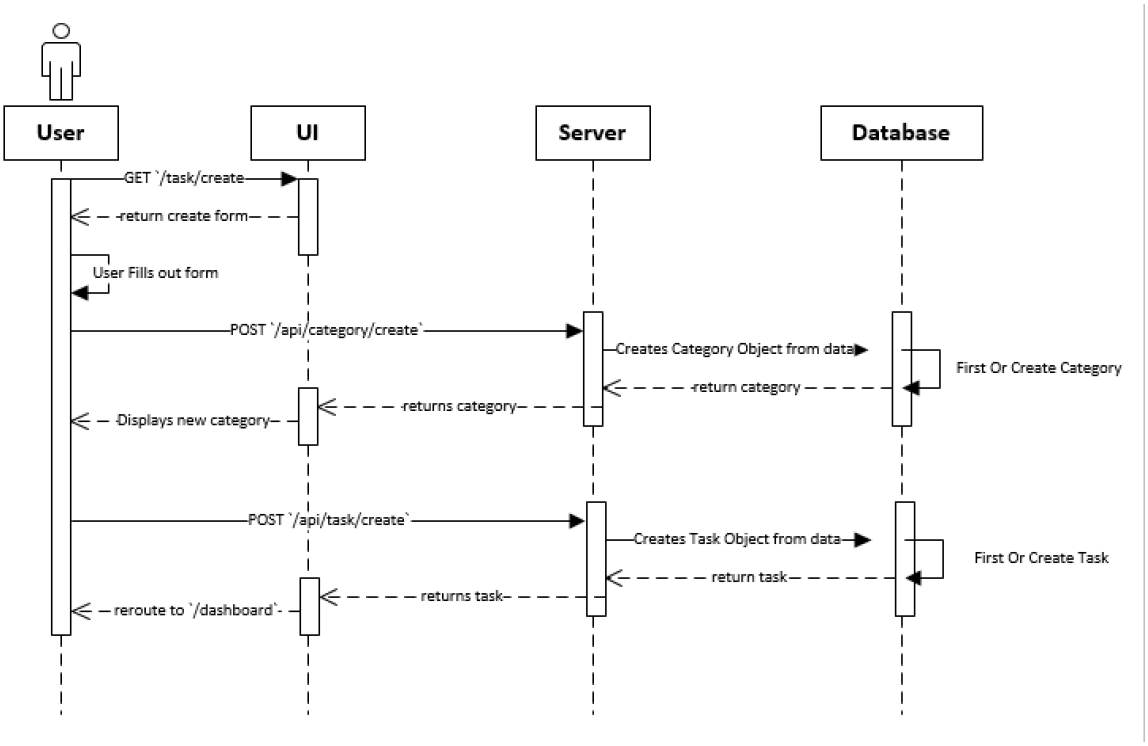


Sequence Diagrams:

User logs in with 0Auth

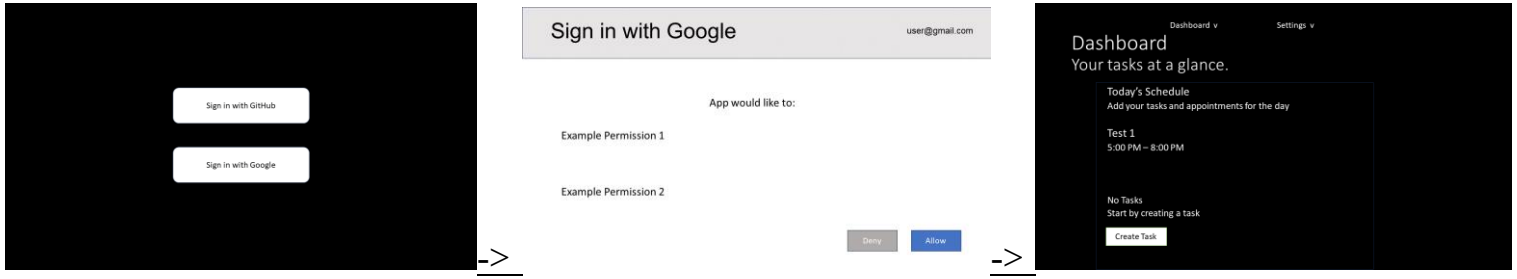


User Creates a new task with a category



User Interface Specification

Reference



Click “Sign in with Google” Click “Allow” when prompted If successful, user will end up on the Dashboard

Use Case: User creates a new Task

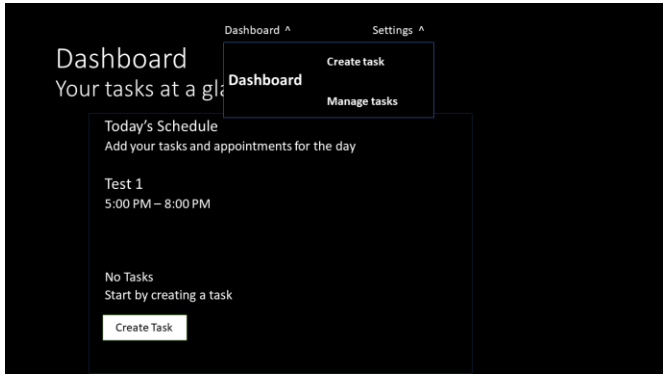
From the dashboard click the Create Task button.

Alternatively, Click Create Task from the top menu.

Fill in all the required details and then click Create.

User is redirected back to the dashboard but the new task is added.

Use Case: User deletes a Task

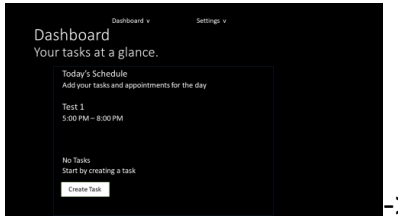


->

On the top menu, click “Manage Tasks”

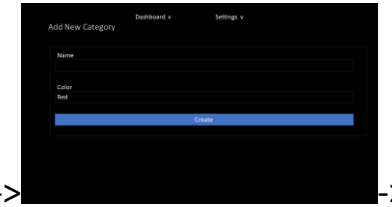
Click “Delete” next to the task that needs to be removed

Use Case: User adds a category



->

->



->

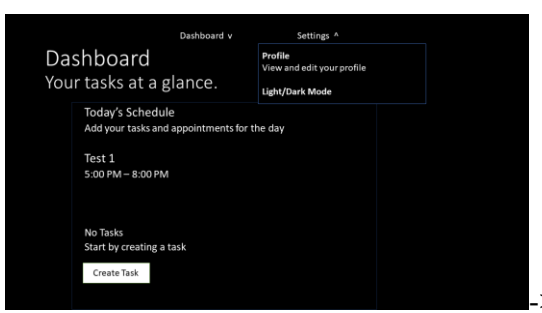
Proceed with task creation as normal and click “Confirm.”

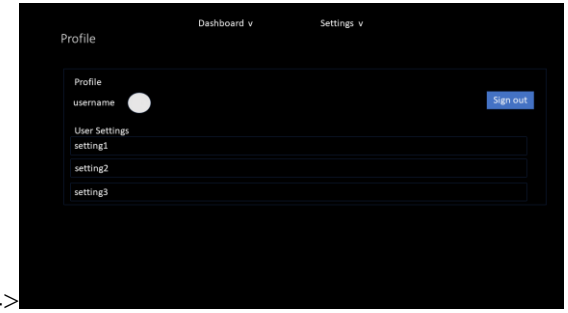
Click the orange + next to categories creating the task.

Fill in required information

Use your new category and finish

Use Case: User edits profile



->


From the top menu, click “Profile” Edit the user settings as needed

Usage Scenario	Navigation	Clicks	Keystrokes
User logs in with 0Auth2 (Google/GitHub)	Sign in screen, Authentication, Dashboard	<5	<20
User creates a new Task	Dashboard, Task creation screen, Dashboard (updated)	<10	<50
User deletes a Task	Dashboard, Task management screen	3	0
User updates a Task	Dashboard, Task management screen, Task edit screen	4	<50
User creates a new category	Dashboard, Task creation screen, Category creation screen	5	<10
User updates a category	Dashboard, Task creation screen, Category management screen	5	<10
User deletes a category	Dashboard, Task creation screen, Category management screen	5	0
User edits profile	Dashboard, Profile settings screen	<5	<20

Project Plan

Weeks 1 – 2:

- Create project proposal.
- Set up development environment.
- Complete

Weeks 3-4:

- Create problem statements and system requirements.
- Complete

Weeks 5-8:

- Initialize Solito project.
- Create interface and connection to the database.
- Gather requirements for backend and implement.

Weeks 9-12:

- Create login flow.
- Create authentication processes.
- Create the landing page.

Weeks 13-15:

- Development of tasks and the daily schedule.
- Develop backend to generate dynamic schedules based on user input.
- Record the final demo to showcase the app’s capabilities.

Traceability Matrix

System Requirements

No.	Priority Weight	Description
REQ1	5	User is able to login or create an account using 0Auth from another account (google or github).
REQ2	5	User is able to create tasks that are saved and displayed on their dashboard and “manage tasks” page.
REQ3	2	Tasks can have many categories, and categories can be created by the user.
REQ4	4	User is able to update tasks.
REQ5	2	User is able to update categories
REQ6	4	User is able to delete tasks
REQ7	1	User is able to delete categories

Use Cases

No.	Description
UC1	Create Tasks
UC2	Update Tasks
UC3	Delete Tasks
UC4	Login with 0auth2
UC5	Edit profile
UC6	View Pages

Traceability Matrix

Req't	PW	UC1	UC2	UC3	UC4	UC5	UC6
REQ1	5				x		
REQ2	5	x	x	x		x	x
REQ3	2	x	x	x			
REQ4	4		x				
REQ5	2		x				
REQ6	4			x			
REQ7	1			x			
Max PW		5	5	5	5	5	5
Total PW		7	13	12	5	5	5

System Architecture and Design

System Architecture

The Time Blocker App will have a simple system architecture. It uses both a client and server-based architecture, the server will allow multiple simultaneous users. The client devices will request information from the web server. The web server will build dynamic parts of the page using next.js, the static parts of the site are rendered on the client. The Vercel database will also be accessed by the application to provide the information to the user. The client machine will access the users' information, the user can also delete, add, and modify items.

Subsystems

There are two subsystems for this app. The first is the web server based on next.js and typescript. The database will be based on Vercel to allow for persistent data storage.
The app will run on any device with a web browser since the app is web based. The database server is hosted in the cloud to allow users to access the app from anywhere.
The app will populate the day's activities to begin with. The user will also be able to access other days. The user does need to input meetings/appointments/etc. before the day's activities can be populated.
The database subsystem will allow for persistent data storage. This subsystem allows the user to input the meetings/etc., into the app to be able to see the day's activities.

Persistent Data Storage

The app will use a database for persistent storage. This will be accomplished using Vercel and their database functions. The user can add items, delete items, and modify items using the web server interface.

Global Control Flow

Middleware will be implemented to control the flow of users within our application. The middleware will check if the user is logged in.
To provide users with relevant information the database queries will include relations to the user's ID. This will help ensure that users do not receive/ are able to view other users' data.

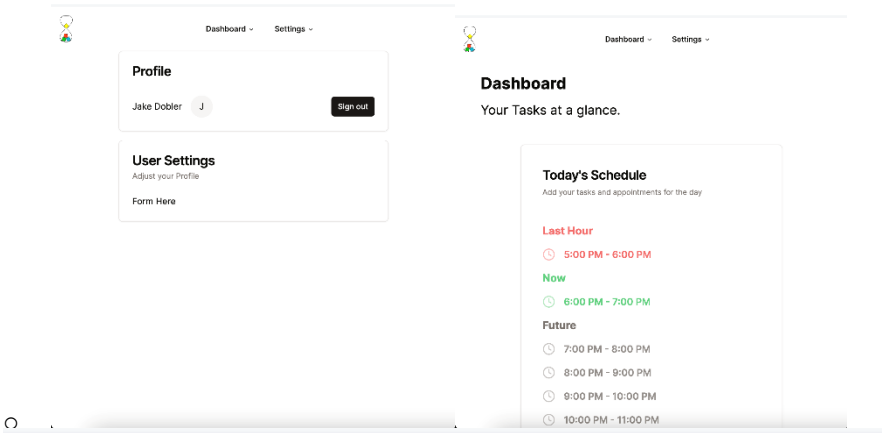
Hardware Requirements

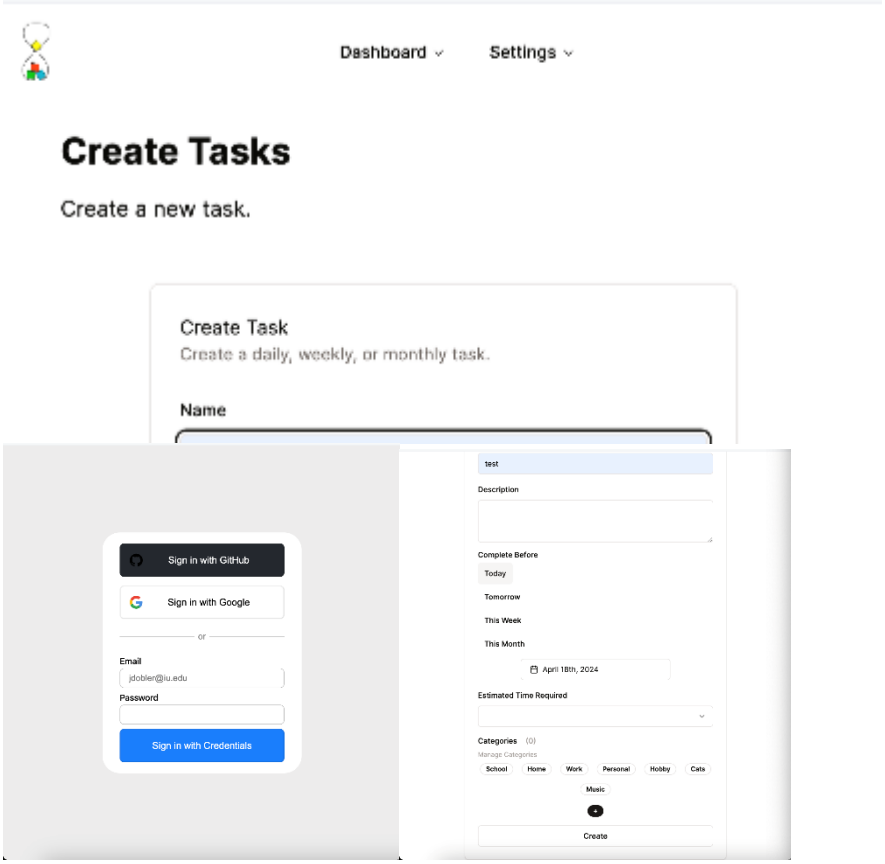
Vercel Hosting
Vercel PostgreSQL DB
(Both services are similar but simpler than AWS.)

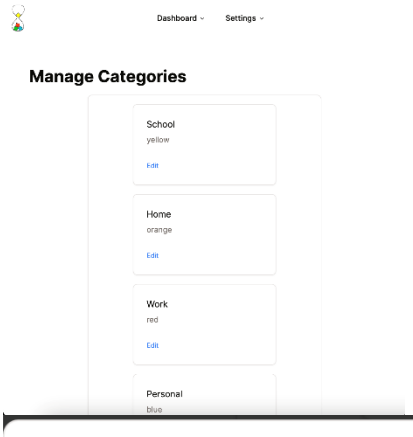
User Interface Design and Implementation, Design of Tests

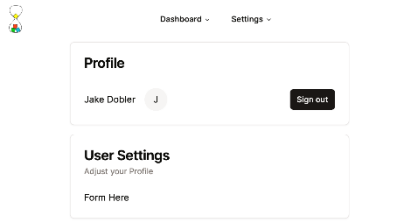
User Interface Design and Implementation

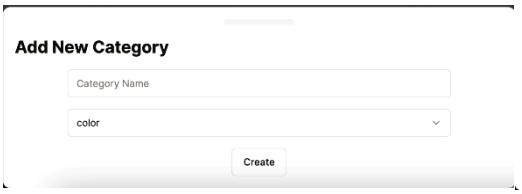
- Our initial mockups were not completely in line with our component library, but using the mockups as a wireframe was helpful. Some significant changes in our user interface are the timeline view on the first page. Initially we wanted to create a schedule/calendar sort of view but given that the user could have little to no tasks this was opted out.











Design of Tests

- Connect to our database
  - This will check that our database is reachable
- Login as a user
  - This will use credentials saved by running `npx playwright codegen 127.0.0.1:3000 --save-storage=auth.json`
- Create a task
  - Validating that a task can be created
- Update a task
  - Validated that the task can be updated

- Create a category
  - Validating that a category can be created.
- Update a category
  - Validating that a category can be updated.
- Remove a category
  - Validating that a category can be removed.
- Remove a task
  - Validating that a task can be removed.
- Our test coverage will be close to 100% of what we have implemented thus far.
- Our integration testing strategy will be run through github actions.
- Playwright will be used to provide end-to-end testing.

References