amora

Team 4 - Design Document

Alex Hardewig, Zach Johnson, Benjamin Kahlert, Eric Vondrak, David Wood, Ian Zanger

Purpose

Managing group work can be difficult due to conflicting schedules, unclear task assignments, differing rates of work, and disruptive meetings. Our application will address these issues by providing an easy way for groups to see what project tasks are incomplete/in progress/complete, filter tasks by the subgroups and individuals they are assigned to, and monitor what tasks are in progress across the team. Further, a “today” view individual to each team member will allow users to manage what tasks they hope to accomplish in their remaining hours of work that day. Additional features such as a points system and productivity rating will make the application more fun for casual users and more useful for managers who want to view which members contribute the most.

The functional requirements will be:

1. User
    a. A user can create an Amora account using their Google account. This will create an Amora profile associated with their Google profile.
2. Project
    a. A project is a collection of tasks.
    b. A project can have any number of users associated with it.
    c. The project’s creator becomes its manager and has increased customization options for that project.
3. Task
    a. Any user can create a task. Tasks can be created for any project.
    b. A manager can assign themself or another team member a task.
    c. A task is either unassigned, incomplete, in progress, or completed.
    d. All project members can view the tasks created for a team.
4. Manager
    a. A manager is the user who created the team.
    b. A manager can assign tasks to projects, people to tasks, priority levels to tasks, and deadlines to tasks.
5. Comments
    a. A user can comment on any task within their team.
    b. A user can edit a comment that they created.
6. History
    a. Each project will have an archive of completed tasks.
7. Calendar

    a. A user can add tasks to their “Today” view, which will show the day’s events and tasks assigned to them for the day.
    b. A user will be notified when it is time to start a task.
    c. A user will be able to see an estimated amount of time left in a day.

8. Dashboard
    a. A dashboard will show all of the available tasks to a user.
    b. Personal tasks will also be included in this dashboard.
    c. A manager will have a special dashboard that will show updated options on each task.

Design Outline

Client-Server Architecture

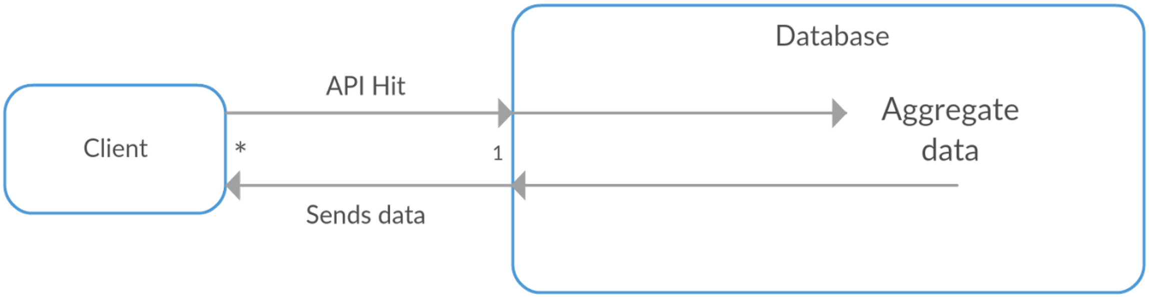
Our program will use a client-server architecture in terms of how our software interacts with a database. The server (Google's Firebase) will keep, secure, and provide all of the data for our users while the client will be responsible for retrieving and using that data in a meaningful way.

Client:

- Sends HTTP request to Firebase
- Receives data in response to request in JSON format
- Manipulates/Uses data in meaningful way for the user

Server:

- Receives and Validates request
- Database is either updated or information is sent back depending on request



Model-View-Controller Architecture

Our program will also employ a model-view-controller architecture to use data to display information to the user. Generally, a view is what the user sees, a controller is how a user interacts with a program, and a model is where all the data is stored in the client. Our application is not a deviant from the formula. The user will see the program through the view. When the user clicks on anything in the view, the actions they perform will be made meaningful by a controller class which will manipulate a model.

View:

- Selects required data that can be found in the model

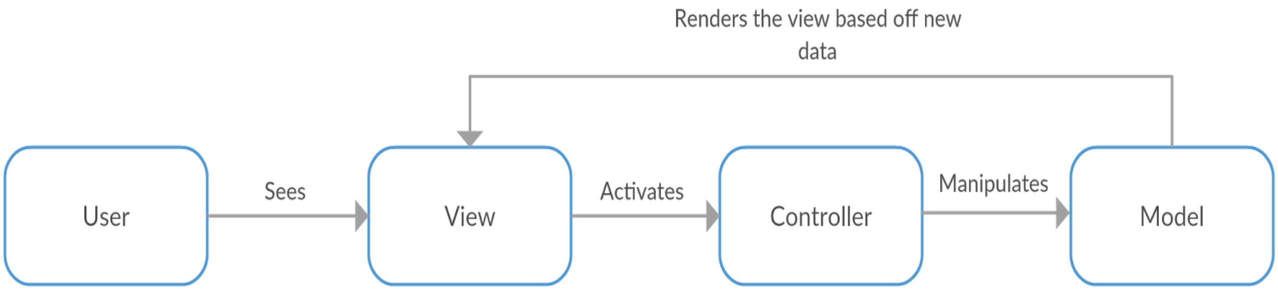
- Displays information in a pleasing and efficient way to the user

Controller:

- Sets up and listens for actions from the user
- When an action listener is triggered, a certain request is performed (usually involves manipulating the model)

Model:

- Receives data from Firebase and user input
- Stores data in an efficient manner so view and controller can access it quickly



Design Issues

What front-end framework/library should we use?

- **React**
- Angular
- Bootstrap/JQuery
- Vanilla JS

We have chosen React as our primary library due to its benefits in creating a modern, fluid user interface. It focuses on the use of components to compartmentalize data and enable a high degree of reusability. This concept is perfect for Amora due to its nature as a task-oriented application - there will be many design components such as tasks and reminders that are repeated across the sections of our user interface. Additionally, React enables live updates of data displayed as it is updated and altered, without the need for refreshes. This is important for our application so that users can see what their teammates are editing as it happens.

What type of backend should be implemented in our application?

- Develop a custom backend and server solution
- **Use a prebuilt backend service**

We have decided to utilize an existing backend service for our application. Using a Backend-as-a-Service will keep us from having to learn new languages to build our own database and server, which would be a significant time investment as our team has limited experience in that area. This will allow us to spend more time on the front-end usability of the application, which is ultimately what the user experiences and forms their opinions based on. Though a prebuilt service will lack the flexibility and customization of a tailor-made solution, we don’t anticipate this to be an issue. Almost all of our data will be simple text, so we likely would not benefit much from a more complex solution.

Which Backend-as-a-Service should we use?

- **Firebase**
- Parse Server
- Deployd

- MongoDB Stitch

We chose Firebase because it is simply the best supported and most well-rounded Backend-as-a-Service. It is well-documented, with extensive APIs that will give us many options when considering processes such as authentication. It features a real-time database that will pair nicely with our choice of React for our front end library. It is also cross-platform, making it much easier for us to expand to iOS and Android in the future if we choose to do so. It allows for plenty of data accesses and storage space during our development, testing, and initial launch, and we can easily upgrade to a higher plan as needed.

How should we handle the user login system?

- Build our own verification system
- **Use an API**

We have chosen to use an API for handling login information. Though we could manage our own database of email addresses and passwords, this poses many security risks and would inevitably be less secure than using another company's established API. Even if our application is unlikely to store highly sensitive data, security is of utmost importance. Many users tend to reuse passwords, so there is no room for error in storing them. Though Firebase does support custom authentication and storage in the database, and this information would be stored by Google on servers as secure as those of a normal Google login, this is not transparent to users. Users are not aware that we are using Firebase, and would see a custom login screen, which is less trustworthy than the authentication processes they already know from trusted sources. Additionally, we believe users would prefer not to have to sign up for yet another account for a new web application. Ultimately, using an API is the preferred choice due to ease of integration and preferences of users.

Which API should we use for our login system?

- Github
- **Google**
- Facebook
- LinkedIn

After determining that we would use an API for our login system, we decided Google was the best choice for handling authentications. We've chosen Google for a few reasons. The simple fact that it's well-known and trusted among millions of people makes it an attractive choice. Additionally, it is integrated with Firebase, which simplifies the authentication process and ensures compatibility. We considered using Github for authentication because a focus of our application is professional programming teams, but we also want it to be accessible to non-programmers and amateur users. Everyday people don't normally have Github accounts, so using it for authentication would require more setup for them. Google strikes a balance by being commonly used by both amateur users and professionals.

How should we structure the code of our application?

- **Model/View/Controller**
- Model/View/Presenter
- No established model, just code organically and see how it develops

It was a pretty clear choice for us to have some sort of structure in our code. Having no order or architecture for our internal code would result in chaos. No standard would also make issues like merge conflicts even more unruly. We even plan to name all of our variables, classes, and components similarly and so we can read each other's code with ease. We've chosen an MVC architecture because it is a structure our team is most familiar with, and it is also widely used in industry.

How much customization should we allow?

- No customization
- **Some customization**
- Full customization

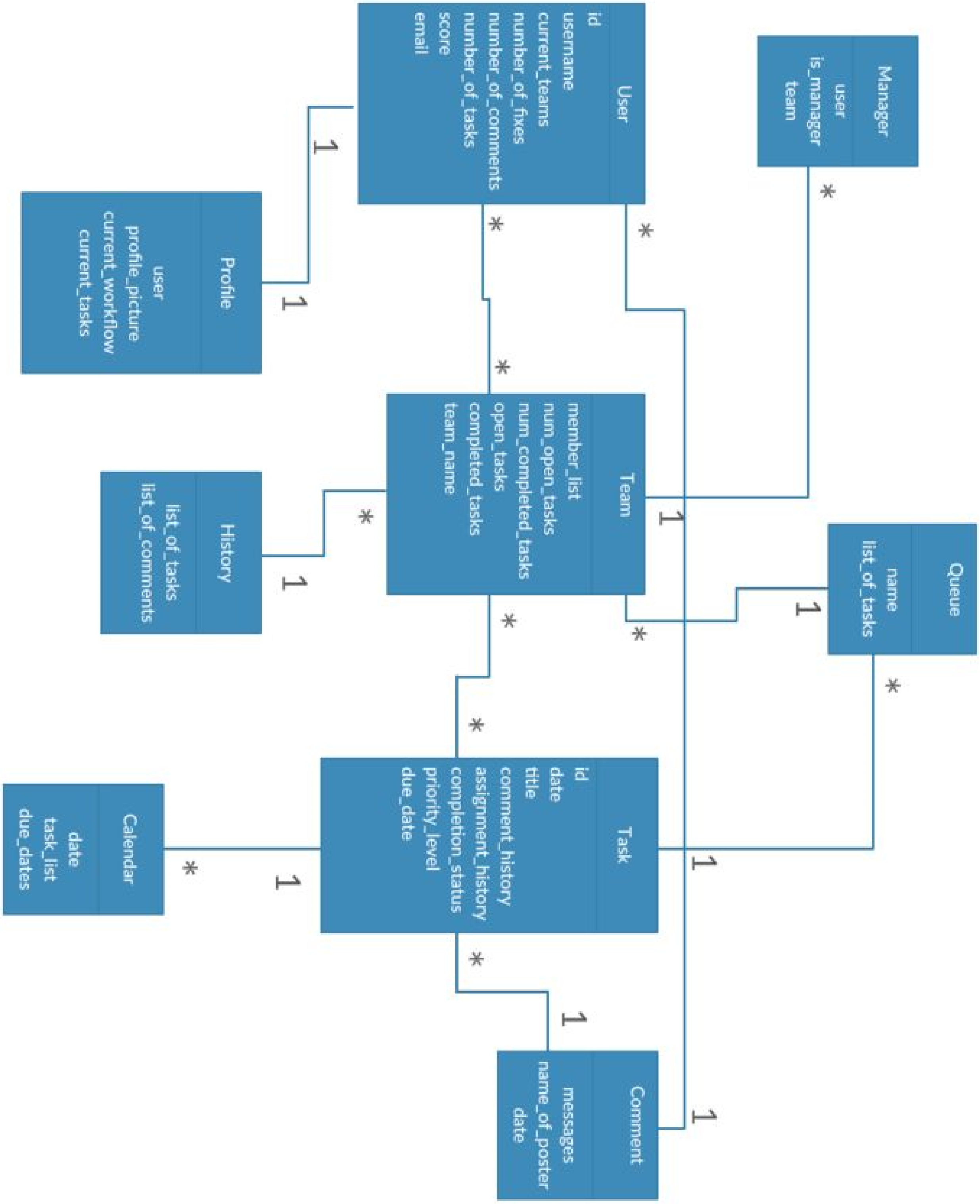
We have decided to allow for a moderate amount of customization by users, but in a controlled way. Having no customization is more limited than we believe users would want, as many people want to color code tasks for easier viewing. In this way, customization is actually a productivity aid, and not just a "nice to have." On the opposite end of the spectrum, too much customization can leave an application looking cluttered, garish, and unprofessional. Giving users the option of custom backgrounds or a full color picker could lead to an inconsistent UI that doesn't scale properly to different screen sizes. We will aim for the middle by allowing for customization with a limited palette of colors and options, maintaining a consistent user experience.

How should we handle permissions?

- Let everyone have the same permissions
- **Allow managers to have heightened permissions**
- Have a strict hierarchy with several levels of permissions

We will allow for managers to have higher permissions than the rest of the team. Allowing everyone access to all features and settings of the project is simplest for users to understand, but isn’t very powerful for managers who want to have more control over their projects. Although a big focus of our application is professional teams where presumably everyone on the team is trustworthy, we also want our application to be useful for casual groups and school projects. School projects especially may involve strangers, so managers will want to limit them from having heightened permissions. By default the project creator will be the sole manager and can promote other members to manager as well. Managers will have the ability to edit project settings, add and remove members, and view overall statistics for their team.

Design details



Classes

1. User
   a. User class will include ID number, username, current teams, number of tasks completed, number of comments, number of tasks currently assigned, score, and email.
   b. The user class will maintain statistics for the user and provide an identity with which to join teams, create tasks, post comments, and maintain queues.

2. Project
   a. Project class will include member list, number of open tasks, number of completed tasks, history of completed tasks, list of open tasks, and team name.
   b. The project will be the primary place for users to collaborate with colleagues and friends.

3. Task
   a. Tasks will include creation date, id number, title, comment history, assignment history, completion status, priority level, due date, and the project it belongs to.
   b. Tasks may be created by team members and assigned to team members by managers. They include a comment history where users can leave comments or questions about the task.

4. Manager
   a. The manager class will be a superclass of a user. It will provide additional functionality and permissions such as ability to add users to a project, remove users from a project, set priority levels, change deadlines on a task, and receive suggestions on which team member is best suited for completing a specific task.
   b. Managers will act as the owners of a team.

5. Comments
   a. Comments will include messages, ID of commenter, and message dates.
   b. Comments can be left by team members and team managers and are located on the task information screen.

6. Queues
    a. Queue class will contain name and task list.
    b. The purpose of queue is to separate tasks into separate lists

7. History
    a. The history class will contain lists of completed tasks and comments.
    b. The history class serves as a list of all completed tasks so that they can be easily indexed.

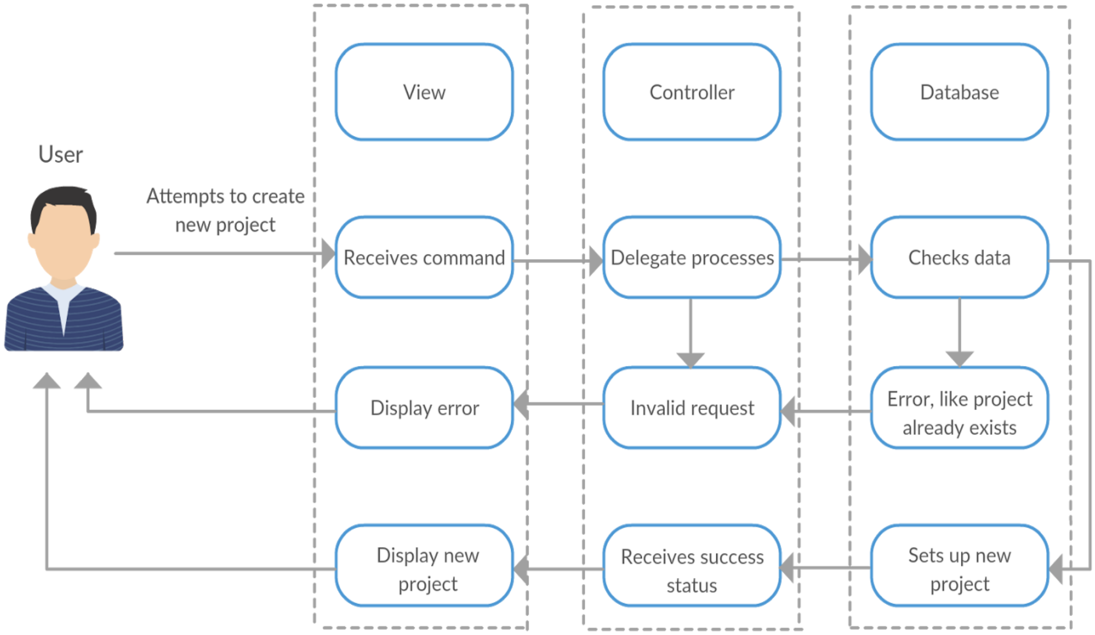
8. Calendar
    a. Calendar class will contain dates, tasks, and maintain task due dates.
    b. The calendar will essentially become a container for tasks and will show visually when things are due.

9. Profile
    a. The profile class will be a superclass for the user class. It will also have a profile picture, number of tasks completed, and number of things the user is currently working on.
    b. The profile class will allow other users to see current workflows for other users.

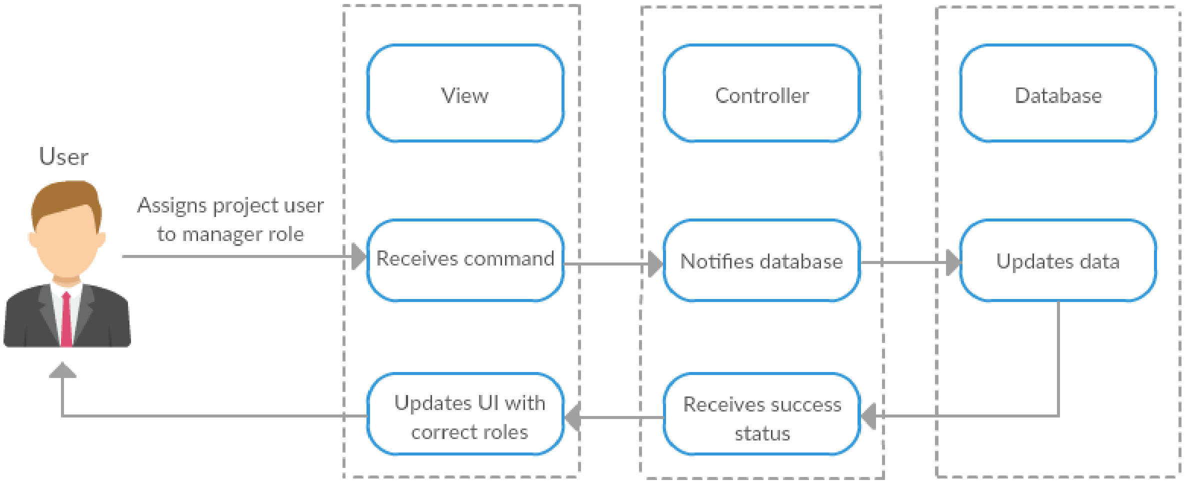
10. Dashboard
    a. The dashboard class will be used to visually represent team tasks.

Sequence Diagrams

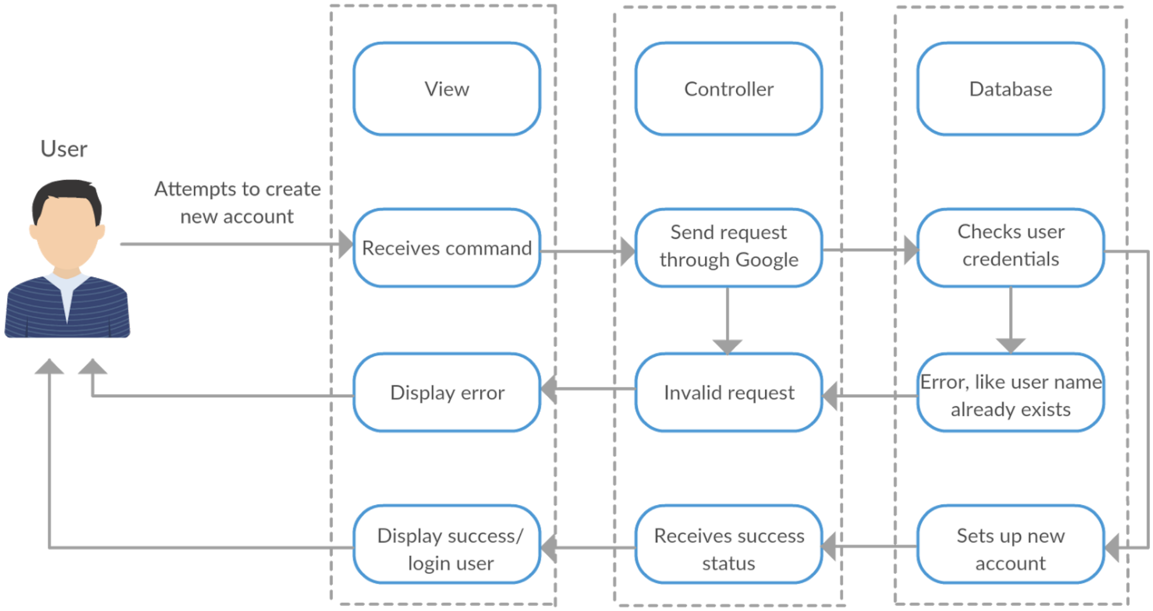
User creates a new project:



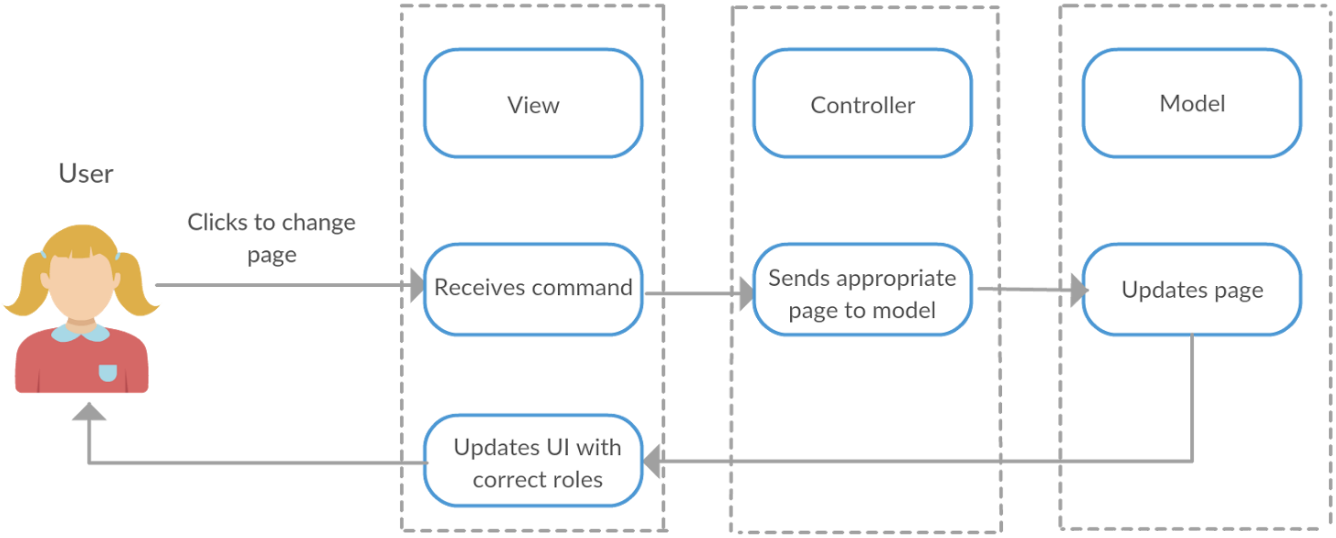
Manager promotes team member in a given project:



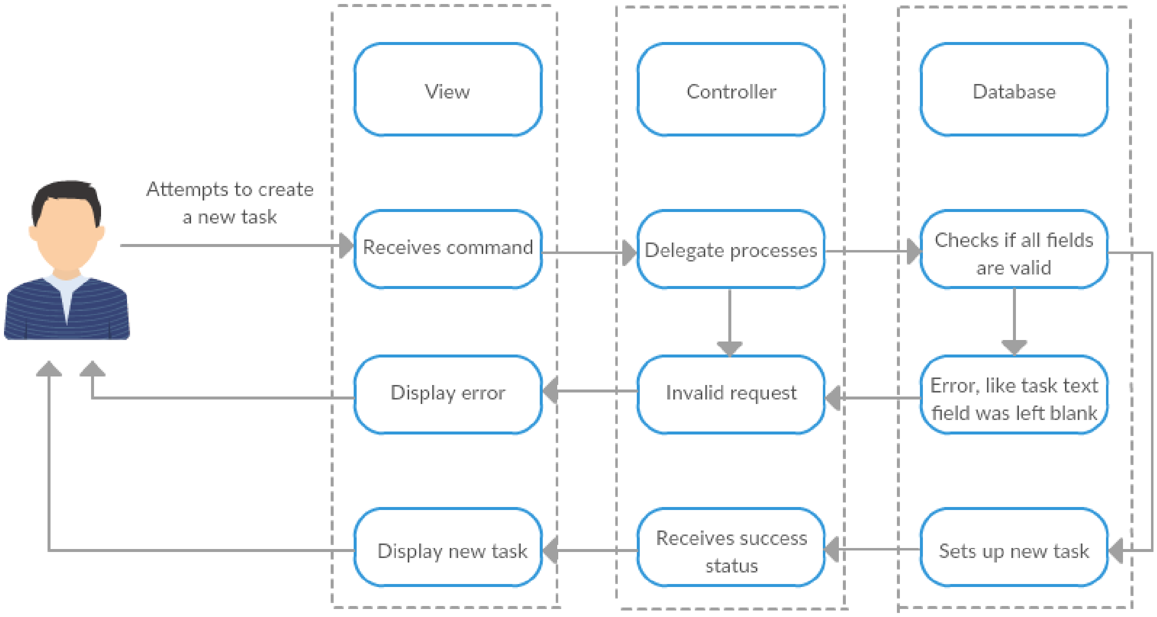
User creates an account/Signup Process:



User switches screens:

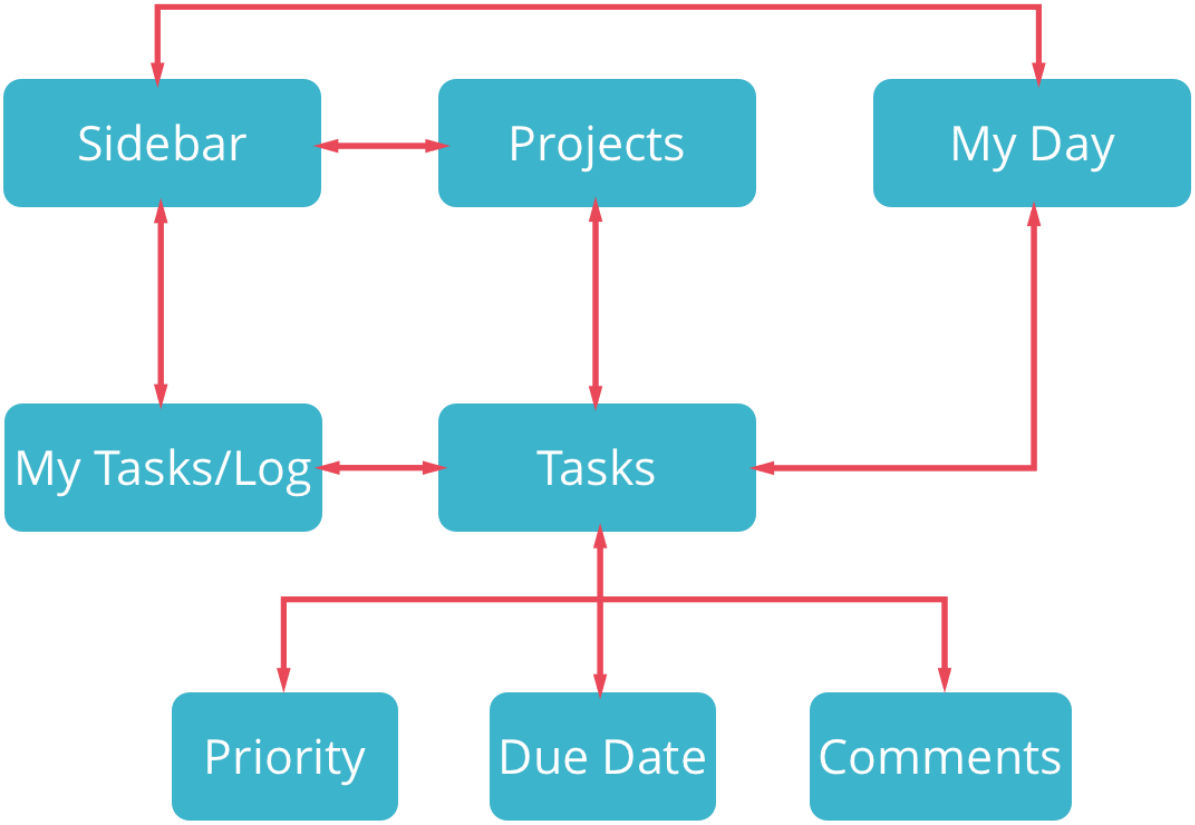


User creates a new task for their project:



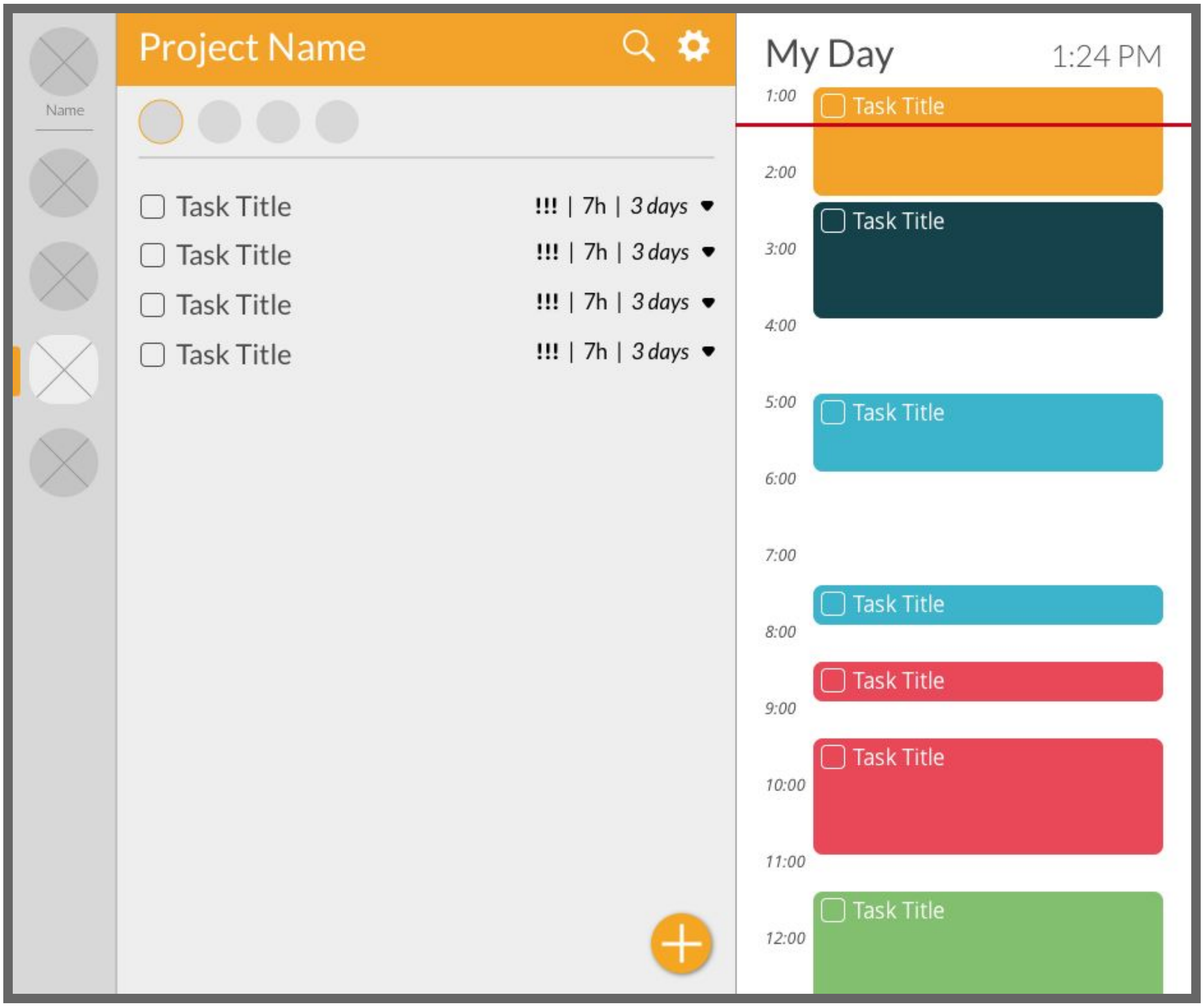
Navigation Flow Map

We built our user experience with an emphasis on beneficial, simplistic behaviors for ease-of-use. The ‘My Day’ view displays a user’s current planned day and can be expanded to view all tasks in addition to each individual project. Tasks can easily be dragged between views and projects, and clicking a task reveals additional important information. It is designed to be responsive to both desktop and mobile use by incorporating buttons that work with both a cursor and touch interactions. Below is an abstracted version of the navigation flow. As you can see, all components can be accessed from the multiple views.

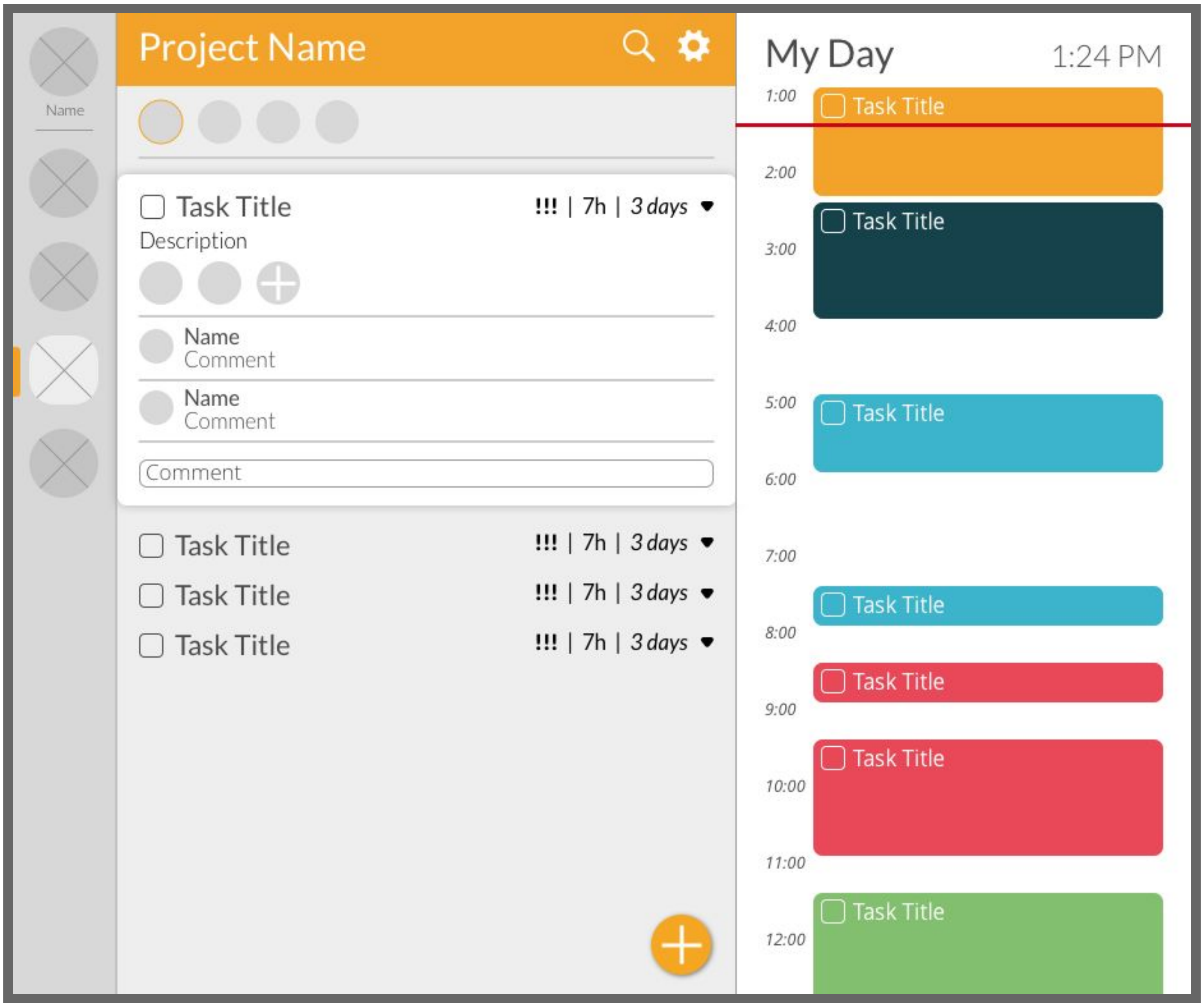


UI Mockups

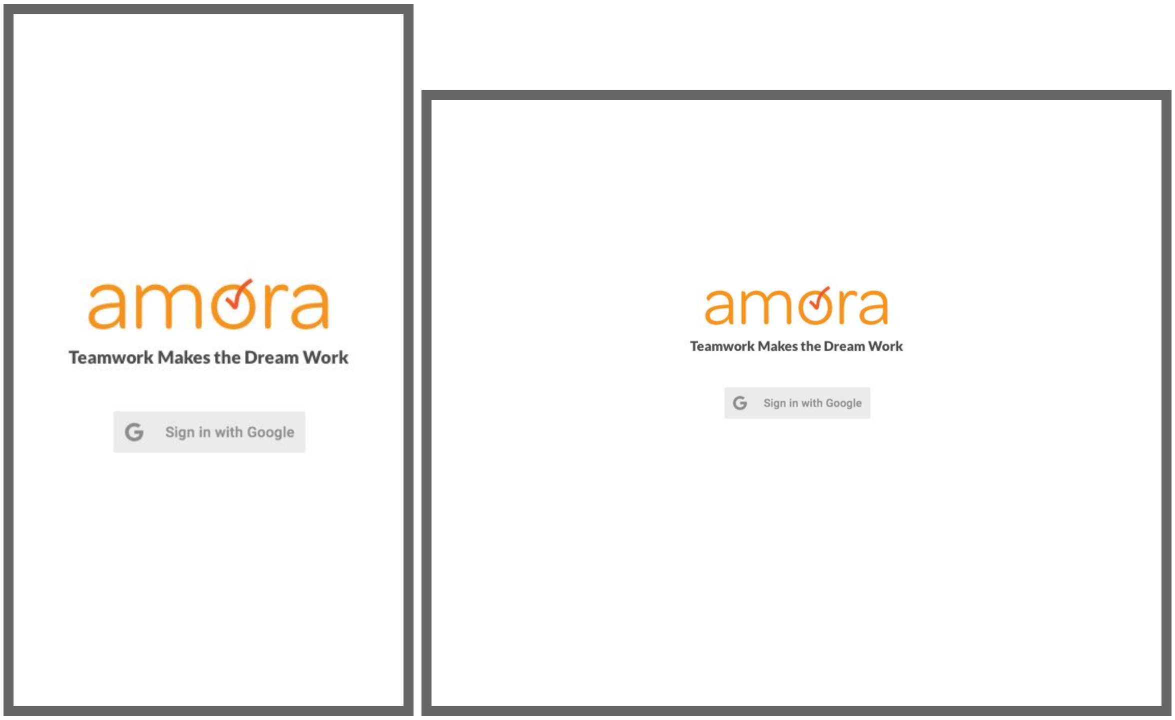
We designed our user interface to be simplistic in its display of information. We use contrasting colors to differentiate tasks and projects, and use increasing brightness to shift a user's focus to the necessary components of the screen.



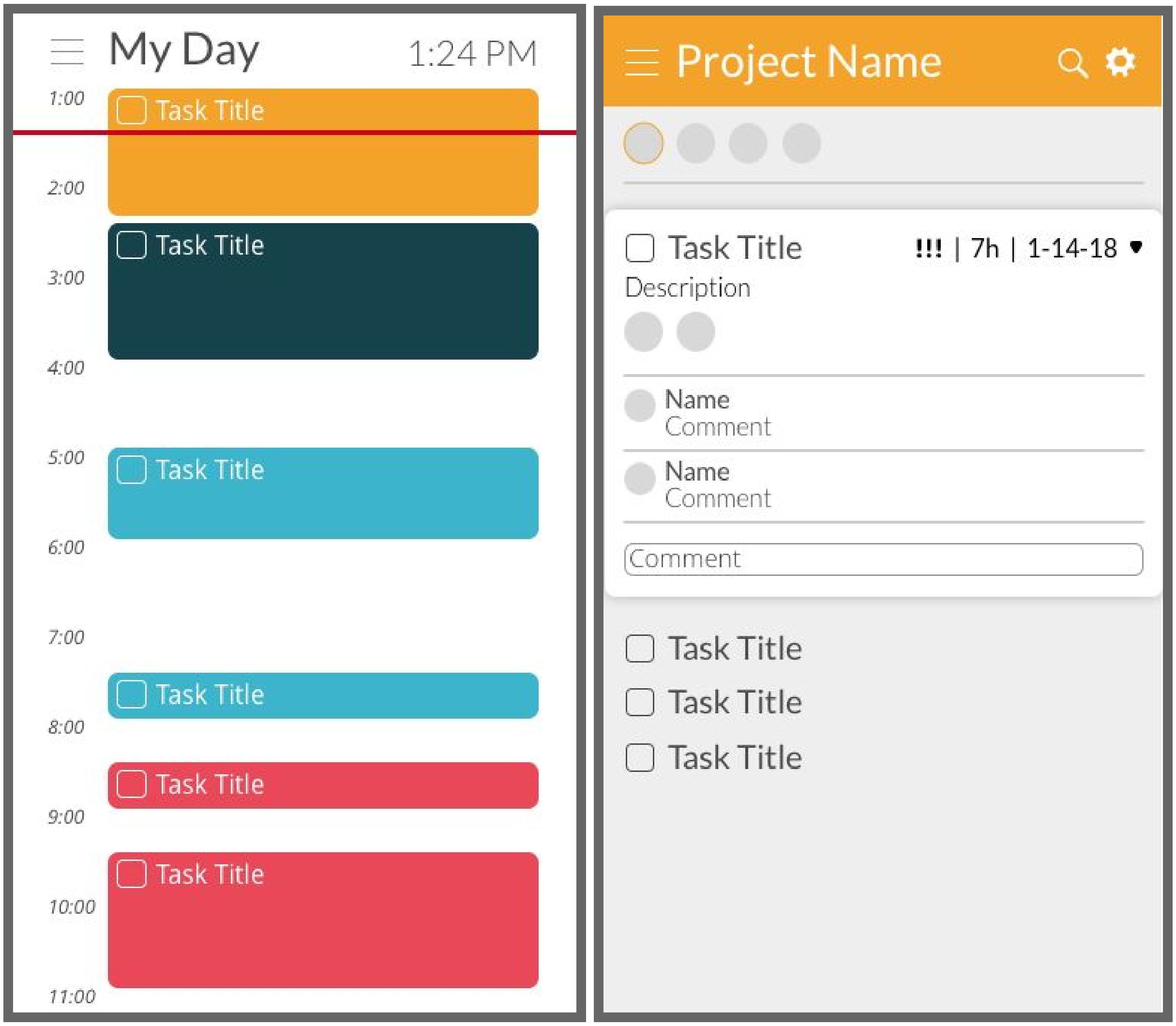
This is the main view for the desktop user interface. Projects are displayed on the left sidebar, the current project is displayed in the center, and the 'My Day' view is displayed on the right-hand side. Users can drag tasks between views and directly into their days.



Upon clicking on a task, a user is presented with more information on the selected task, such as the description, assigned members, comments, priority, etc. Managers are displayed with a colored outline and are presented with more customization options.



The login screen is simplistic and directs a user to sign in with Google credentials.



The design easily translates to a mobile device. Instead of all components being displayed, each portion of the screen is displayed. A hamburger button in the top left corner reveals the projects and 'My Day' screen.