“Planner’s Paradice” System Design Documentation

Contents

Class Descriptions

Below are our CRC cards describing each of the Collections/Components of our application. We have five main collections: Users, Tasks, Projects, Notes, and Extensions and four components that control how they are accessed.

Class Name: User	
Parent Class: N/A	
Responsibilities: • The user knows their name, email, password, and country • The user has a unique ID • The user has a do not disturb status (on/off) ○ They also have a count for how long this is active for • The user has points they accumulate by completing tasks • The user also has a list of extensions (widgets/backgrounds) they have unlocked using their points	**Collaborators:** • Extension

Class Name: Project	
Parent Class: N/A	
Responsibilities: • Projects have a unique identifier • Projects have a list tasks that make up the project's subtasks • Projects have a completion date, indicating when the user wants to complete the project by • Projects have a progress field indicating how far along the project is • Projects have a title so the user can identify it • Projects have a user ID, associating them to their owner	**Collaborators:** • User • Task

Class Name: Note	
Parent Class: N/A	
Responsibilities: • Notes have content, i.e. what the user has noted down • Notes have a title • Notes have a user ID to identify its creator/owner • Notes have an ID to uniquely identify the note	**Collaborators:** • User

Class Name: Task	
Parent Class: N/A	
Responsibilities: • Tasks have a unique identifier • Tasks have content, i.e. a description of what the user wants to do • Tasks have a completion date i.e. the day they get checked off • Tasks have a deadline i.e. a planned day to be completed for • Tasks have a priority level i.e. how urgent it is to complete • Tasks have a has been reschedule date indicating whether the task has been rescheduled or not • Tasks have a user ID to associate them to their author • Tasks have a field indicating whether they are part of a project	**Collaborators:** • User • Project

Class Name: Extension	
Parent Class: N/A	
Responsibilities: • Stores the extension's ID	**Collaborators:**

Class Name: DAO Interface	
Parent Class: N/A **Sub Classes:** DAO User, DAO Notes, DAO Tasks, DAO Projects	
Responsibilities: • Update, create, and get items given their ID	**Collaborators:**

Class Name: DAO Notes	
Parent Class: DAO Interface	
Responsibilities: • Add a note for a user given the notes content, title and username • Update a note with new content given its owner's user ID and its ID	**Collaborators:** • Note • User

Class Name: DAO Tasks	
Parent Class: DAO Interface	
Responsibilities: • Add a task for a user given the task's content, completion date, priority, and username • Update a task with its new content, or priority given its owner's user ID and the task's ID • Mark a task as completed priority given its owners user ID and its ID • Change the completion date for a task and mark it as rescheduled priority given its owner's user ID and its ID	**Collaborators:** • Task • User

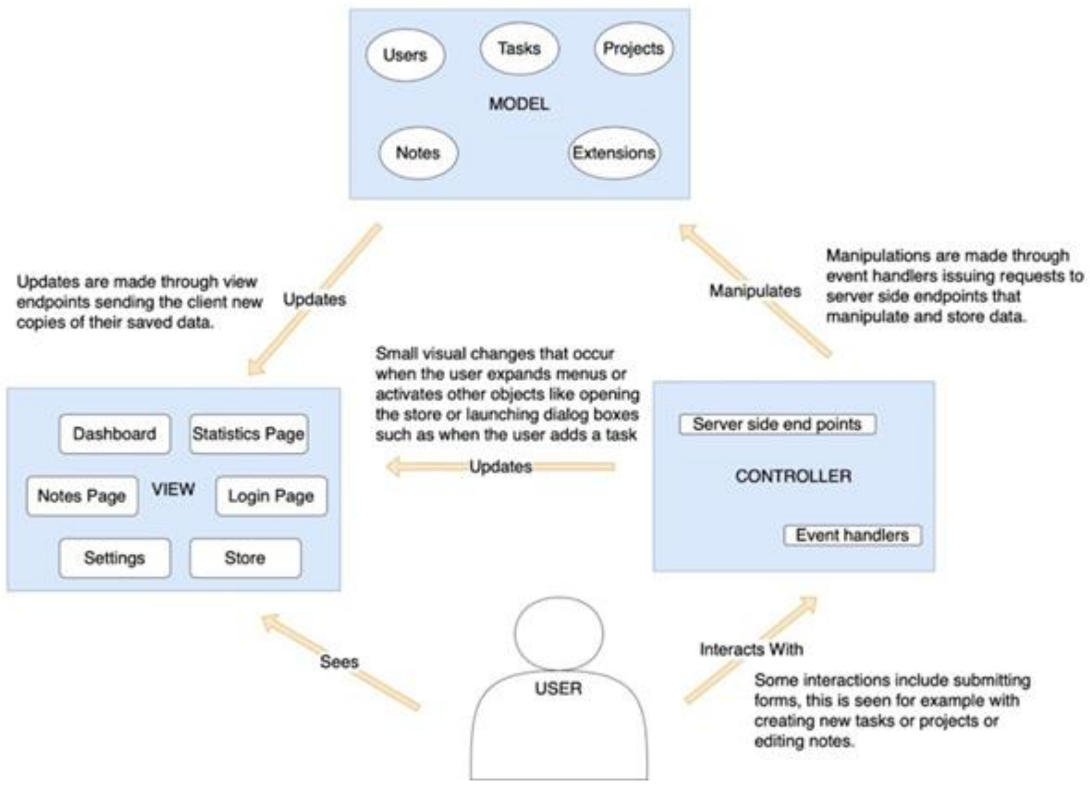
Class Name: DAO Projects	
Parent Class: DAO Interface	
Responsibilities: • Add a project for a user given the projects title, subtasks, completion date and the user's username • Update a project with new subtasks, or changed subtasks given its ID and its user ID • Mark a project's subtask as completed given the project's ID, subtask ID, and the user ID ○ This will update the project's progress as well • Update a project with a new completion given its ID and the user ID • Update a project with a new title given its owner's user ID, and the project's ID	**Collaborators:** • Project • User • Task

Class Name: DAO User	
Parent Class: DAO Interface	
Responsibilities: • Create a user given their email address and password • Get a user given their username and password • Update a user given their username • Unlock widget or background for the user given the user's ID, and the extension's ID • Enable do not disturb for the user given the user's ID	**Collaborators:** • User • Extension

System Interaction

Our system interacts with its environment as follows. Firstly, since it is designed to be a Google Chrome new tab extension it should be able to work on any OS as long as the user is using the Chrome browser. It will work with Chrome as follows; when the user launches their browser or opens a new tab they will be presented with our application. It is interacting with Chrome by replacing the default new tab or new window menus. This may work as follows; when a user launches their browser, they will be greeted with our application and if they are not logged in, they will be asked to login, or register with our app if they have yet to do so. Upon logging in they will see their dashboard on our app and can use it from there. Every time the user launches a new tab they will be greeted with this same dashboard as long as they are signed in. Secondly, it interacts with its environment through its interactions with the network and our server. For example, it interacts with our server as when our application is launched our server will send the browser the required JavaScript code to be run and processed on the client's browser. Any requests made from the app through the client's browser are also sent to our servers. Our system is dependent on their network connection and our server being active. We are also dependent on our servers being able to process the requests using Express JS and save any needed data on our MongoDB database. Our app will also use Auth0 as a middleman service to interface between our clients and our backend server code to handle the authentication of users. This is how our system will interact with other systems and how we assume these operations will work.

System Architecture


Users
Tasks
Projects
MODEL
Notes
Extensions
Updates are made through view endpoints sending the client new copies of their saved data.
Updates
Manipulates
Manipulations are made through event handlers issuing requests to server side endpoints that manipulate and store data.
Small visual changes that occur when the user expands menus or activates other objects like opening the store or launching dialog boxes such as when the user adds a task
Updates
Dashboard
Statistics Page
Notes Page
VIEW
Login Page
Settings
Store
Server side end points
CONTROLLER
Event handlers
Sees
USER
Interacts With
Some interactions include submitting forms, this is seen for example with creating new tasks or projects or editing notes.

System Decomposition

The Model
The Model is the database of our server that stores all entities. Each entity in the database belongs to one of five collections: User, Task, Note, Project, or Extension. These collections make up the different models of our application; see our CRC cards to learn about their attributes.

The View
The view of our app is what the user sees and how the information is presented to them. Our system's view includes multiple pages. These pages include the Dashboard displaying tasks and projects, the notes page displaying notes, the store panel where the user can buy extensions/widgets using their points, the settings page, the statistics page, and others.

The Controller
This is a combination of two groups of functions in our application. The first group is the client side functions/event handlers that update or create data. They will do some pre-data manipulation or data gathering before sending the requests off to the server. The second group is the server side endpoints. These endpoints will receive requests from the client side event handlers and will further process it before storing it in the server, or updating the data in the server.

Error Handling
To handle exceptional circumstances our app is designed to always fail gracefully, and never fail in a way that would reveal underlying implementation details or aspects of the app the user otherwise shouldn't see. Some exceptional circumstances that may occur that we are designing our system to handle are the following: invalid form input, network errors, server errors, failed authentication, and invalid requests. To describe how these errors are handled we will split them into categories and give examples of them below.

Invalid Authentication And Invalid Form Input
To handle invalid authentication and invalid form input we will display red text over the field describing an error, and disable submission of the form. For example, if someone were to try to register with an invalid email address we will say "Invalid email" in red text over the input box. Or if someone were to try to enter too many characters in a task's content box we will say "Too many characters" over the task input box. These are a few examples of how our system would handle these kinds of exceptions.

Failed Authentication

Failed authentication is when a user tries to sign in with a username or password that is not in our database. To handle failed authentication we will display a red error message on screen saying that the username or email does not exist and ask the user to try a different email or password or prompt them to register.

Network Errors, Server Errors, and Invalid Requests

To handle network errors, server errors, and invalid requests we will treat them all quite similarly. All of which upon occurring will simply display a red toast box in the top right corner of the display with a messaging describing the error. In the case of a network error this message will say, "something went wrong, please try again." For example, if someone tries to submit a new task but the request timed out we will display a red toast saying "something went wrong, please try again", also their form data will still be present and they won't need to enter it again in this case. In the case the error is server related we will prompt the user saying "we've encountered a problem processing your request, our engineers are on it!" in the red toast, and we will not clear their form data. Lastly, if the error is due to an invalid request, that is the user has found a way to get around our form validation and submitted the form they will be prompted with an error message saying "invalid request, the data you have entered is incorrect, please try something else" in the red toast, and their form data will be reset. This is how our system will handle these three types of exceptions.