System design document

BY TEAM WABI SABI

System Design document

Wabi Sabi

Table of Contents

CRC cards

The cards below show the components of our project broken down into frontend, backend and database.

For more information for backend APIs you can also check swagger in our GitHub.

the documentation for frontend and backend components are placed in Doc/Sprint1/documentation and then you can select frontend or backend folders to see the documentation.

Frontend

Class name: DashboardContainer (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Displays the different dashboards based on the path user is on • Displays logout dropdown function • Displays navigation bar • Wraps around dashboards	**Collaborators:** • Logout • Navbar • TimerDahboard • TodoDashboard • ChartsDashboard • FriendsDashboard

Class name: OnBoarding (frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Wraps around signup, login, SignupSuccessCard	**Collaborators:** • Signup • Login • SignupSuccessCard

Class name: Logout (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • has a dropdown menu that when clicked on the logo opens • has logout as an option in dropdown menu • when user clicks on logout it logs user out by calling logout in AuthProvider	**Collaborators:** • AuthProvider

Class name: Navbar (frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Holds buttons for each available dashboard • When user clicks on a button for a dashboard navigates user there • When user is on a dashboard keeps the button of that dashboard as SelectedButton and all other buttons as UnselectedButton	**Collaborators:** • BrowerRouter

Class name: Signup (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Has a form that receives email password and confirm password • Checks if confirm password is the same as password if not shows error message • When user submits calls backend signup API • If SignupAPI sends error, displays error • If signup is successful navigate user to Signupsuccesscard	**Collaborators:** • Signup in Auth Routes (backend) • SignupSuccessCard

Class name: SignupSuccessCard (frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • shows a success message • allows user to navigate back to login to login with their newly created account	**Collaborators:** •

Class name: TImerDashboard (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Holds Timer • Contains timer function buttons Such as Starts, Pauses, Skips, Resets Timer • Contains different timer options such as pomodoro, short break and long break as buttons	**Collaborators:** • Timer

Class name: Timer (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Calculates time track of time • Contains timer functions that are called by timerdashboard such as start, pause and reset • Passes time to ClockFace	**Collaborators:** • ClockFace

Class name: ClockFace (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Displays time from Timer	**Collaborators:** • Timer

Class name: AuthProvider (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Provides attributes that check whether user is logged in (auth token, isLoggedIn) • Provides login function • Provides logout function	**Collaborators:** • ProtectedRoute

Class name: ProtectedRoute (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Checks if user is logged in and prevents access to component it wraps around, if not	**Collaborators:** • DashboardContainer

Class name: AppContainer (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Contains entire app • Contains Router which routes to every page in the app	**Collaborators:** • AuthProvider • Router

Class name: BrowserRouter (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Routes user to specific pages in the app	**Collaborators:** • AuthProvider

Class name: Login (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Contains user login form • Logs user in • Sends request to backend	**Collaborators:** • AuthProvider

Class name: ToDoDashboard (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Fetches user's list of tasks from database, displaying them in a table (if the user has any). • Allows functionality for deleting a task. • Update and display the progress of each task.	**Collaborators:** • AddTask • AuthProviderUtils

Class name: AddTask (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Adds a task for a user given a task title. • Optionally, the user can set a due date tag and subtasks	**Collaborators:** • ToDoDashBoard

Class name: ChartsDashboard (Frontend)	
Parent Class (if any): N/A for react **Subclass(if any): N/A for react**	
Responsibilities: • Compiles data about user habits and shows them as charts	**Collaborators:** • User (database)

Class name: FriendsDashboard (Frontend)	

Parent Class (if any):
 	
Subclass(if any):	

Responsibilities: • Shows a list of friends a user has • Allows user to add a friend using the friend's email or id • Allows user to delete a friend	**Collaborators:** • User (database)

Backend

Class name: TagRoutes (Backend)	
Parent Class (if any): N/A **Subclass(if any): N/A**	
Responsibilities: • Functions for adding tags given it's description • Function for getting list of tags	**Collaborators:** • Tag (database)

Class name: TaskRoutes (Backend)	
Parent Class (if any): N/A **Subclass(if any): N/A**	
Responsibilities: • Routes to preform CRUD operations on Tasks: • /tasks/add: creates a new Task object and add it to the user specified in the • /tasks/get: Returns a JSON array of Tasks for the User specified in the header. • /tasks/edit: changes the status of a Task • /tasks/rm: Removes a Task from the user	**Collaborators:** • User • Tasks • Protected Route • Tag

Class name: StudyRoutes (backend)	
Parent Class (if any): N/A **Subclass(if any): N/A**	
Responsibilities: • study/add: Adds a study session based on the time started, time ended, and tag given from the JSON body • Study/get: Returns a JSON array of study sessions for the user in the auth	**Collaborators:** • Study • User

Class name: AuthRoutes **(**backend)	
Parent Class (if any): N/A **Subclass(if any): N/A**	
Responsibilities: • /signup ○ Create a User, given an email and password ○ Create a User, given an email and password ○ Checks if User already exists with email when signup is called ○ Inserts new User into database • /login ○ Given email and password, retrieve User from database ○ Checks if User exists when login called ○ Checks if password is correct ○ Generate and returns a JWT authorization token	**Collaborators:** • User

Database

Class name: User (database)	
Parent Class (if any): N/A **Subclass(if any): N/A**	
Responsibilities: • Manages a user of the application • Has a list of Tasks, study sessions, Friend users.	**Collaborators:** • Task • User • Auth

Class name: Task (database)	
Parent Class (if any): N/A **Subclass(if any): N/A**	
Responsibilities: • Manages a single task for a user to complete • Stores the text, due-date, tag, and status of the Task • Also stores sub-tasks within an array	**Collaborators:** • User • Tasks • Tag

Class name: Study (database)	
Parent Class (if any): N/A **Subclass(if any): N/A**	
Responsibilities: • Manages a study session taken by a user • Maintains the start and end times • Optional tag for the study session	**Collaborators:** • User • Tag

Class name: Tag (database)	
Parent Class (if any):N/A **Subclass(if any): N/A**	
Responsibilities: • Knows the tag id • Knows its description	**Collaborators:** • User

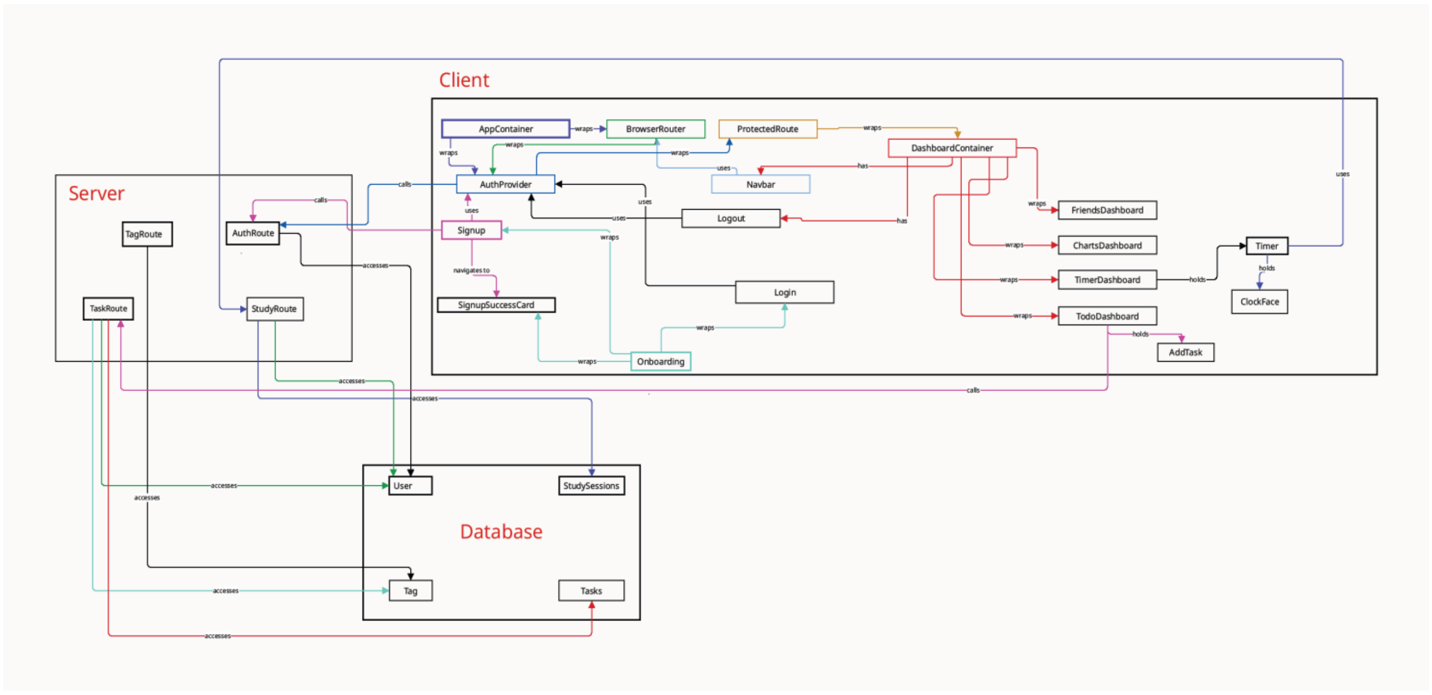
Software Architecture Diagram

For details regarding parameters and each class please check the CRC cards and the documentation in GitHub.

For backend APIs you can also check swagger in our GitHub and the documentation is placed in Doc/Sprint1/documentation and then you can select frontend or backend.

We are using three-tiered architecture as mentioned in Sprint 0 handout.
https://www.linuxjournal.com/article/3508

P.S.: We have added a separate pdf called system architecture to show this diagram separately for a better quality.



Errors and exceptions

This is the list of errors and how we handle them

Error handling:

- User Authentication: Logging in (Backend)
  - Request: User logs into their account with an invalid email address or password
  - Response: Error code 401 - “Invalid email address or password”
- User Authentication: Signing up (Backend)
  - Request: User signs up with an existing email address (Account already created with email address)
  - Response: Error code 400 - “Email is already registered”
- Adding Task (Backend)
  - Request: User has invalid jwt token in request header
  - Response: Error code 401- “invalid session token”
  - Request: Malformed Task data in JSON body
  - Response: Error code 400: “error adding task”
- Getting Task (Backend)
  - Request: User has invalid jwt token in request header
  - Response: Error code 401- “invalid session token”
  - Request: Error retrieving User or Tasks from database
  - Response: Error code 400: “Error getting tasks”
- Editing Task (Backend)
  - Request: User has invalid jwt token in request header
  - Response: Error code 401- “invalid session token”
  - Request: Malformed Task data in JSON body
  - Response: Error code 400: “error editing task”
- Removing Task (Backend)
  - Request: User has invalid jwt token in request header
  - Response: Error code 401- “invalid session token”
  - Request: Malformed Task data in JSON body
  - Response: Error code 400: “Error removing task”
- Adding Study Session (Backend)
  - Request: User has invalid jwt token in request header
  - Response: Error code 401- “invalid session token”
  - Request: Malformed study session data in JSON body
  - Response: Error code 400: “error adding study sessions”
- Getting Study Session (Backend)
  - Request: User has invalid jwt token in request header
  - Response: Error code 401- “invalid session token”
  - Request: Error retrieving User or Study Sessions from database
  - Response: Error code 400: “Error getting Study Session”
- Setting Tag (Backend)

    - Request: User has invalid jwt token in request header
    - Response: Error code 401- “invalid session token”
    - Request: Malformed Tag data in JSON body
    - Response: Error code 400: “Error making Tag”
- Signup (Frontend)
    - If the email address is already being used we display email is already registered
    - If the password and confirm password don’t match we say passwords must match.
- To do list (frontend)
    - The user gets the following the warning message under the task title input if they submit a task without a task title (Frontend):
        - “Please enter a task”
- User Authentication: Logging In (Frontend)
    - Invalid email/password: User input incorrect email or password (email does not exist/password is wrong)