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 - Uses the **saveHandler** function to save a user's study session data to the local cache whenever a user navigates out of the timer page	**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:	**Collaborators:**
• Wraps around signup, login, SignupSuccessCard	• Signup • Login • SignupSuccessCard

Class name:	
Logout	
(Frontend)	
Parent Class (if any): N/A for react	
Subclass(if any): N/A for react	
Responsibilities:	**Collaborators:**
• 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	• AuthProvider

Class name:	
Navbar	
(frontend)	
Parent Class (if any): N/A for react	
Subclass(if any): N/A for react	
Responsibilities:	**Collaborators:**
• 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	• BrowerRouter

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

Class name:	
SelectedTag	
(Frontend)	
Parent Class (if any): N/A for react	
Subclass(if any): N/A for react	
Responsibilities:	**Collaborators:**
• Holds a tag dropdown that has all the tags that the user has added • A button to add tags that stores the added tag in database as well by calling the backend • Allows users to select a tag that they want • Returns the selected tag in the callback of component • User can unselect their selected tag	• TagRoute(backend) • Tag(database) • FocusTime(frontend)

Class name:
TImerDashboard
(Frontend)

Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
- 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 - Responsible for pomodoro timer flow - Calls **addStudySession** to save study session to the backend - Handles saving study session data to local cache when user reloads page/navigates away using the **useSaveDataOnReload** hook and **saveDataToCache** function - Handles changing the timer durations by selecting the gear cog icon, which opens a dialog of input fields - Calls **saveHandler** and **handleDurationChange** to update the timer with new durations.	- Timer - SelectedTag - DashboardContainer

Class name:
Timer
(Frontend)
Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
• Calculates time track of time • Contains timer functions that are called by timerdashboard such as start, pause, reset • Passes time to ClockFace	• ClockFace

Class name:
ClockFace
(Frontend)
Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
• Displays time from Timer	• Timer

Class name:
AuthProvider
(Frontend)

Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
- Provides attributes that check whether user is logged in (auth token, isLoggedIn) - Provides login function - Provides logout function	ProtectedRoute

Class name:
ProtectedRoute
(Frontend)

Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
Checks if user is logged in and prevents access to component it wraps around, if not	DashboardContainer

Class name:
AppContainer (Frontend)

Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
• Contains entire app • Contains Router which routes to every page in the app	• AuthProvider • Router

Class name:
BrowserRouter (Frontend)

Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
• Routes user to specific pages in the app	• AuthProvider

Class name:
Login (Frontend)

Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
- Contains user login form - Logs user in - Sends request to backend	AuthProvider

Class name:
ToDoDashboard
(Frontend)

Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
- Fetches user's list of tasks from database, displaying them in a table (if the user has any) with their corresponding subtasks. - Allows functionality for deleting a task. - Update and display the progress of each task - Display the task's tag (if any) - Calculates the completion percentages of each task based on the completed subtasks.	- AddTask - AuthProviderUtils **SortTable**

Class name:	
AddTask	
(Frontend)	
Parent Class (if any): N/A for react	
Subclass(if any): N/A for react	

Responsibilities:	**Collaborators:**
- Adds a task for a user given a task title. - Optionally, the user can set a due date tag - Optionally, the user can add a list of subtasks - Optionally, the user can select/create a tag and assign it for the task	- ToDoDashboard - AddSubTask - SelectedTag

Class name:	
AddSubTask	
(Frontend)	
Parent Class (if any): N/A for react	
Subclass(if any): N/A for react	

Responsibilities:	**Collaborators:**
- User creates a subtask with a 'title' - Table of subtasks is displayed in the AddTask component - User can delete a subtask before submitting it.	AddTask

Class name:
Charts
(Frontend)

Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
- Compiles data about user habits and shows them as charts - Shows the charts mentioned in the collaborators - Has a carousel for navigating between charts	- User (database) - TaskCompletionChart - FocusTime

Class name:
FocusTime
(Frontend)

Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
- Displays the number of hours user has focused in the past 7 days - Has a tag dropdown to filter the data by the tag that was assigned to the timer - On X axis it has the day of the week - On Y axis it has number of hours	- Charts(frontend) - SelectedTag(frontend) - ChartsRoute(backend)

Class name:
TaskCompletion Chart
(Frontend)

Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
• Displays the % completed tasks by tag	• ChartsRoute • ChartsDashboard • Todo List

Class name:
SortTable
(Frontend)

Parent Class (if any): N/A for react
Subclass(if any): N/A for react

Responsibilities:	Collaborators:
• Provides sorting for the task table based on the selected column header(Title, Tag, Due date, or Completion percentage).	• TodoDashboard

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: /tag [POST]: Makes a new tag /tag/<tag_id> [PATCH]: Edits existing tag based on ID /tag [GET]: Gets all of a users tags /tag/<tag_id> [DELETE]: removes the tag of the id	**Collaborators:** Tag (database)

Class name: ChartRoutes (Backend)	
Parent Class (if any): N/A **Subclass(if any): N/A**	
Responsibilities: /chart/completion [GET]: Gets user task completion percentage based on completed tasks per tag. /chart/weekly/ [GET]: Gets minutes studied that week. /chart/weekly/<tag_id> [GET]: Gets minutes studied that week of that tag.	**Collaborators:** - Tag (database) - User (database) - Study (database) - Task (database)

Class name:	
TaskRoutes	
(Backend)	
Parent Class (if any): N/A	
Subclass(if any): N/A	
Responsibilities:	**Collaborators:**
- Routes to preform CRUD operations on Tasks: /task [POST] creates a new Task object and add it to the user specified in the /task [GET]: Returns a JSON array of Tasks for the User specified in the header. /task/<task_id> [PATCH]: changes the status of a Task /task/<task_id> [DELETE]: Removes a Task from the user	- User (database) - Tasks (database) - Protected Route - Tag (database)

Class name:	
StudyRoutes	
(backend)	
Parent Class (if any): N/A	
Subclass(if any): N/A	
Responsibilities:	**Collaborators:**
/study [POST]: 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	- Study (database) - User (database)

Class name:
AuthRoutes
(backend)

Parent Class (if any): N/A
Subclass(if any): N/A

Responsibilities:	Collaborators:
/signup [POST] - 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 [POST] - 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	User (database)

Database

Class name:	
User	
(database)	
Parent Class (if any): N/A	
Subclass(if any): N/A	

Responsibilities:	**Collaborators:**
• Manages a user of the application • Has a list of Tasks, study sessions, Friend users.	• Task • User • Auth

Class name:	
Task	
(database)	
Parent Class (if any): N/A	
Subclass(if any): N/A	

Responsibilities:	**Collaborators:**
• 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	• User • Tasks • Tag

Class name:	
SubTasks (database)	
Parent Class (if any): N/A **Subclass(if any): N/A**	

Responsibilities:
 	
   • Manages a subtask of a task
   • Stores the text, and status of the
      SubTask
	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. We have updated the diagram based on added functionality.
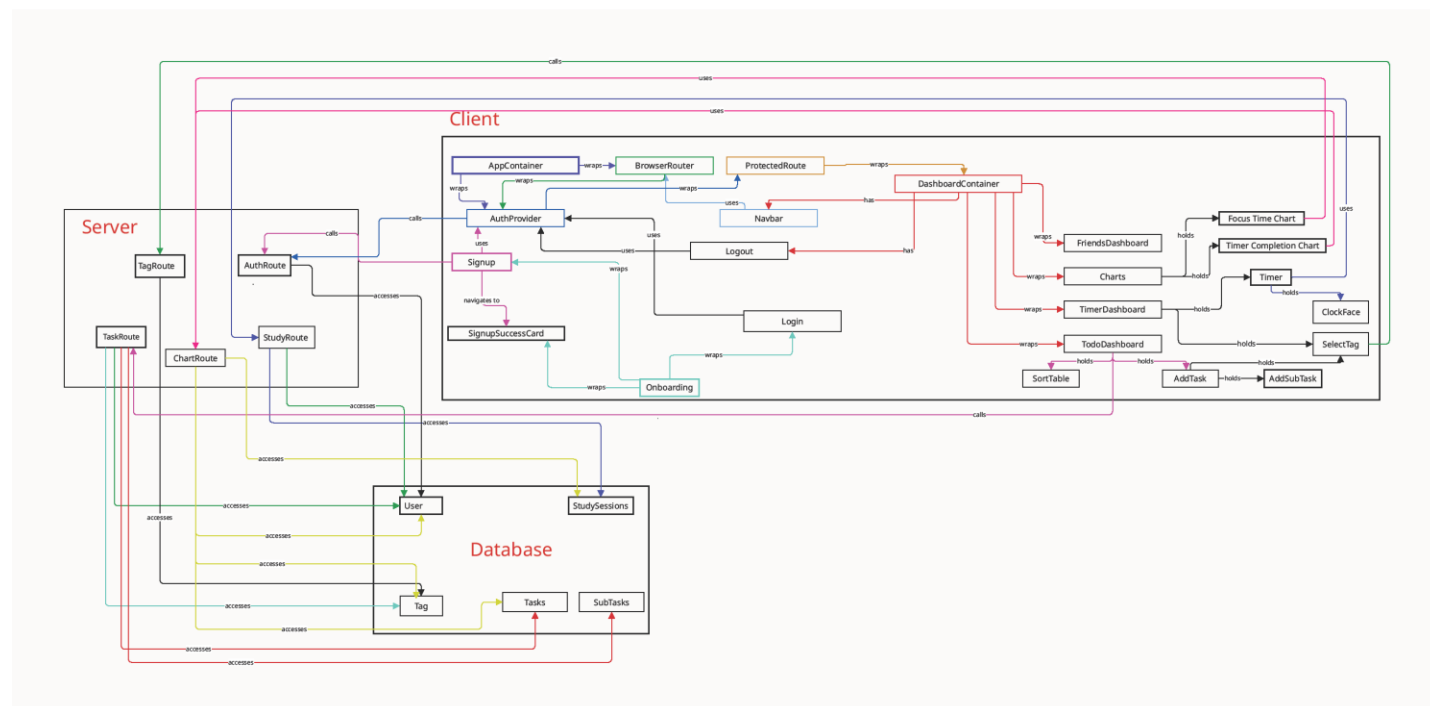
Errors and exceptions

This is the list of errors and how we handle them

Error handling:

- User Authentication: Logging in (Backend)
  - o Request: User logs into their account with an invalid email address or password
  - o Response: Error code 401 - "Invalid email address or password"
- User Authentication: Signing up (Backend)
  - o Request: User signs up with an existing email address (Account already created with email address)
  - o Response: Error code 400 - "Email is already registered"
- Adding Task (Backend)
  - o Request: User has invalid jwt token in request header
  - o Response: Error code 401- "invalid session token"
  - o Request: Malformed Task data in JSON body
  - o Response: Error code 500: "error adding task"
- Getting Task (Backend)
  - o Request: User has invalid jwt token in request header
  - o Response: Error code 401- "invalid session token"
  - o Request: Error retrieving User or Tasks from database
  - o Response: Error code 500: "Error getting tasks"
- Editing Task (Backend)
  - o Request: User has invalid jwt token in request header
  - o Response: Error code 401- "invalid session token"
  - o Request: Malformed Task data in JSON body
  - o Response: Error code 500: "error editing task"
- Removing Task (Backend)
  - o Request: User has invalid jwt token in request header
  - o Response: Error code 401- "invalid session token"
  - o Request: Malformed Task data in JSON body
  - o Response: Error code 500: "Error removing task"
- Adding SubTask (Backend)
  - o Request: User has invalid jwt token in request header
  - o Response: Error code 401- "invalid session token"
  - o Request: Malformed SubTask data in JSON body or url
  - o Response: Error code 500: "Error adding subtask task"
- Editing SubTask (Backend)
  - o Request: User has invalid jwt token in request header
  - o Response: Error code 401- "invalid session token"
  - o Request: Malformed SubTask data in JSON body or url
  - o Response: Error code 500: "Error editing subtask 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 500: "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 500: "Error getting Study Session"
- Getting Task Completion percentage (Backend)
  - Request: User has invalid jwt token in request header
  - Response: Error code 401- "invalid session token"
  - Request: Error calculating completion percentage
  - Response: Error code 500: "Error calculating completion"
- Getting Weekly Study Session Data (Backend)
  - Request: User has invalid jwt token in request header
  - Response: Error code 401- "invalid session token"
  - Request: Error calculating weekly session data
  - Response: Error code 500: "Error calculating completion"
- 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 500: "Error making Tag"
- Getting 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 500: "Error making Tag"
- Editing 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 or url
  - Response: Error code 500: "Error editing Tag"
- Removing 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 or url
  - Response: Error code 500: "Error removing 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)
- SelectedTag (Frontend)
  - The use gets the following warning message under the tag title if they try to add a tag that already exists.
    - "this tag exists."

- Add Task(Frontend)
  - The user gets the following warning message under the task title input if they try to add a task that already exists.
    - "The task already exists"
- Add Subtask(Frontend)
  - The user gets the following warning message under the task title input if they try to add a subtask that already exists.
    - "The subtask already exists"
- TimerDashboard (Frontend)
  - The user gets the following warning message in the dialog container if they to update the timer duration length (minutes or seconds) to a value greater than 59.
    - "Value cannot be greater than 59!"