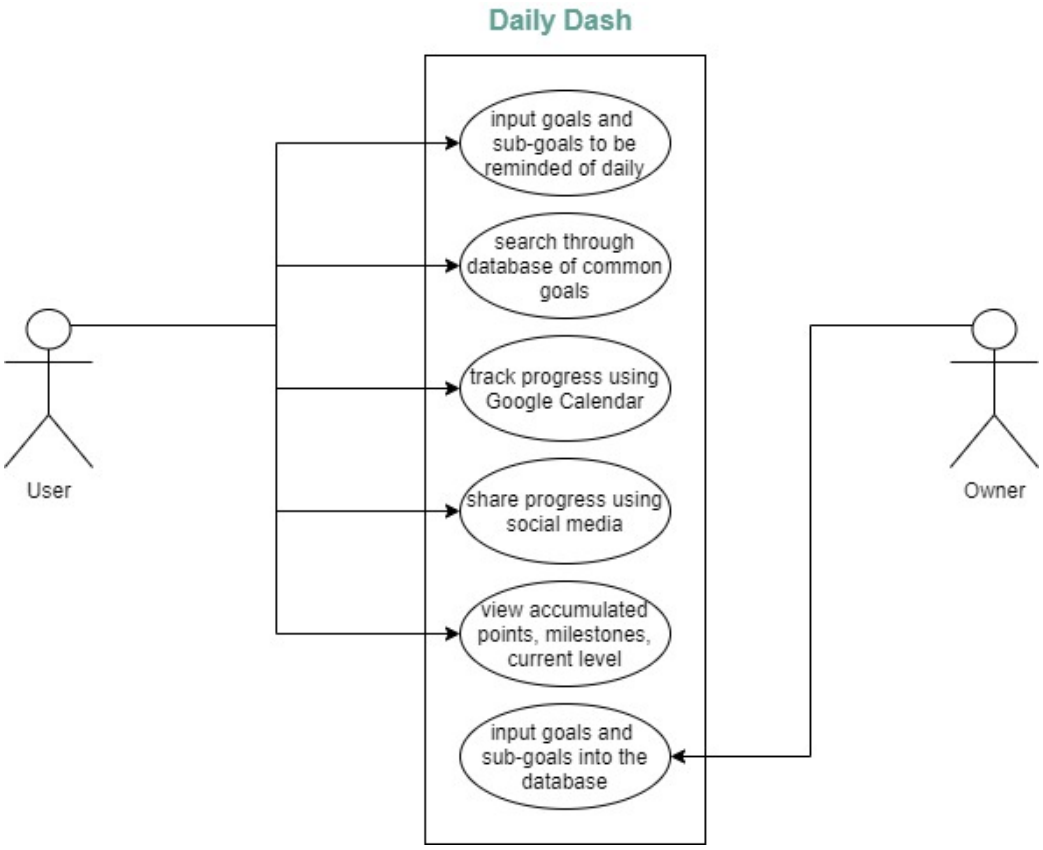
M3: Project Requirements

1. Description:

Daily Dash believes small efforts everyday accumulate to yield life-changing results. Daily Dash empowers users across all walks of life to reach their goals, whether it's to become skilled at public speaking or to adopt a healthy diet. The app provides encouragement at every step of the way via a fun, game-like, points system. It also syncs with Google Calendar to schedule rewards, breaks, and milestones. Using push notifications, the app reminds users to do their tasks everyday. The app also provides a database with sample changes users can implement while also accepting user input.

2. Use Case Diagram



3. Non-functional Requirements

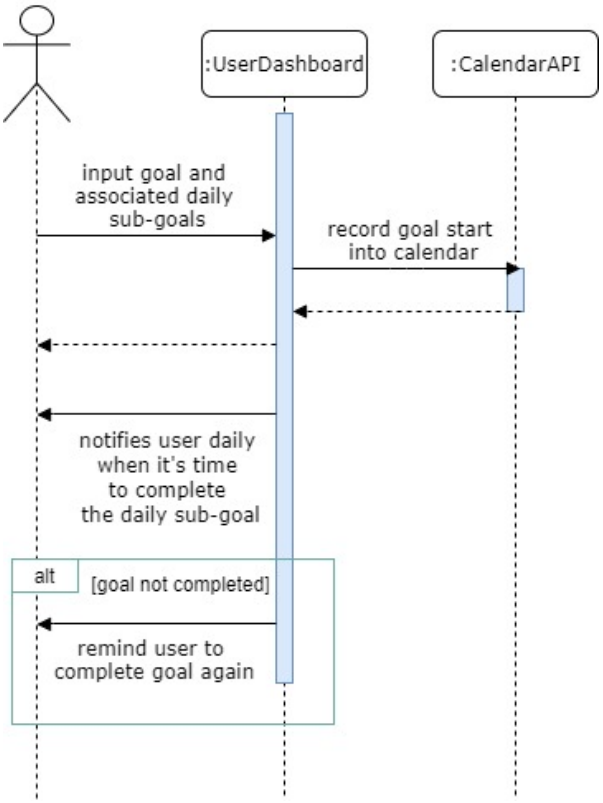
Requirement	Reasoning
The user should not need more than 5 clicks for any action.	Older users who aren't proficient with technology may have a hard time navigating the app otherwise.

The font and graphics should be at least size 12.	Older users might have trouble reading small text.
Milestones and points should be awarded with each task completion. Milestones will be easier to achieve at the beginning and harder later on. They should be challenging enough to keep users engaged.	If milestones and points are too hard to achieve, users will lose motivation. If they are too easy, users will become bored.
Breaks should be scheduled at least once a week.	Breaks too often will disrupt users' momentum. Breaks too seldom will tire users out. 1-2 breaks a week will allow users to rest without disrupting users' rhythms.
Database queries and updates should take less than 2 seconds.	Users don't want to wait a long time while browsing the database or inputting their goals.

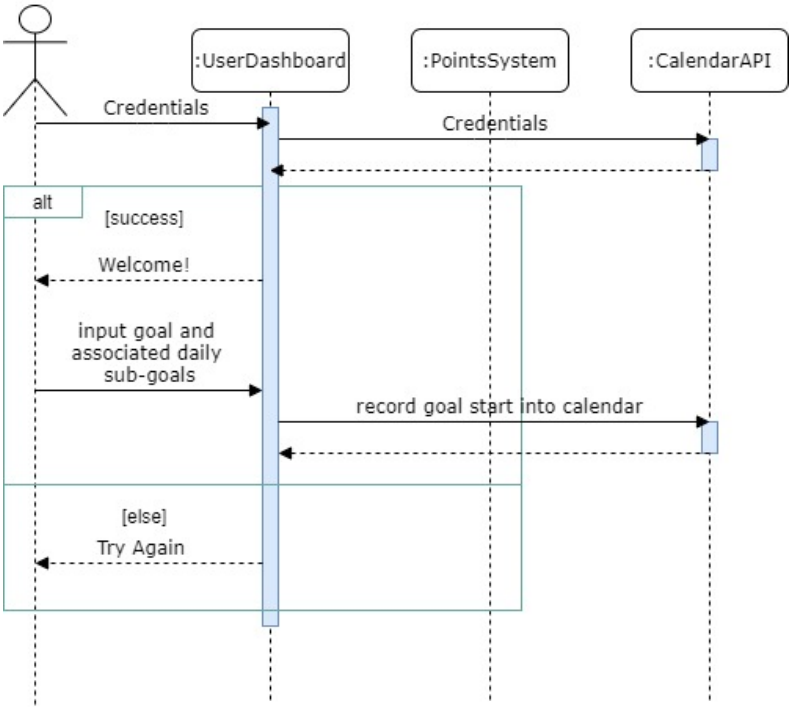
4. **Main components of your system**
    - **Database of common goals:** The app will have a static, unchanging database of common goals and subgoals that users can select or draw inspiration from. For example, a common goal in the database will be "Adopt a healthy diet," and a sub-goal for this goal will be "Cook one meal everyday instead of eating out."
    - **Calendar using Google Calendar:** The synced calender will track when users create goals, achieve awards, level up, and complete their goals. It will also prompt users to add a note with each milstone, perhaps to remark their feelings or reflections about their goal.
    - **User dashboard:** This is where users input new goals, view their goals, and view their daily to-do list comprised of the sub-goals. It will have a todo list type of interface; users can check off daily goals they've completed. Users can also view their current level, current points, and points until next level. The dashboard will also send notifications to the user.
    - **Trophy case:** This screen displays all the awards/milestones users have achieved so far. An example of a milestone could be "30-day streak!," awarded to users if they've accomplished a sub-goal every day for 30 days.
    - **Points System:** The non-trivial backend logic keeps tracks of the user's points and level, awarding milestones when appropriate.

5. **Sequence diagrams for 3 central use cases:**
    - **with external API call**
    - **with logic / calculations**
    - **with live updates (push notifications)**

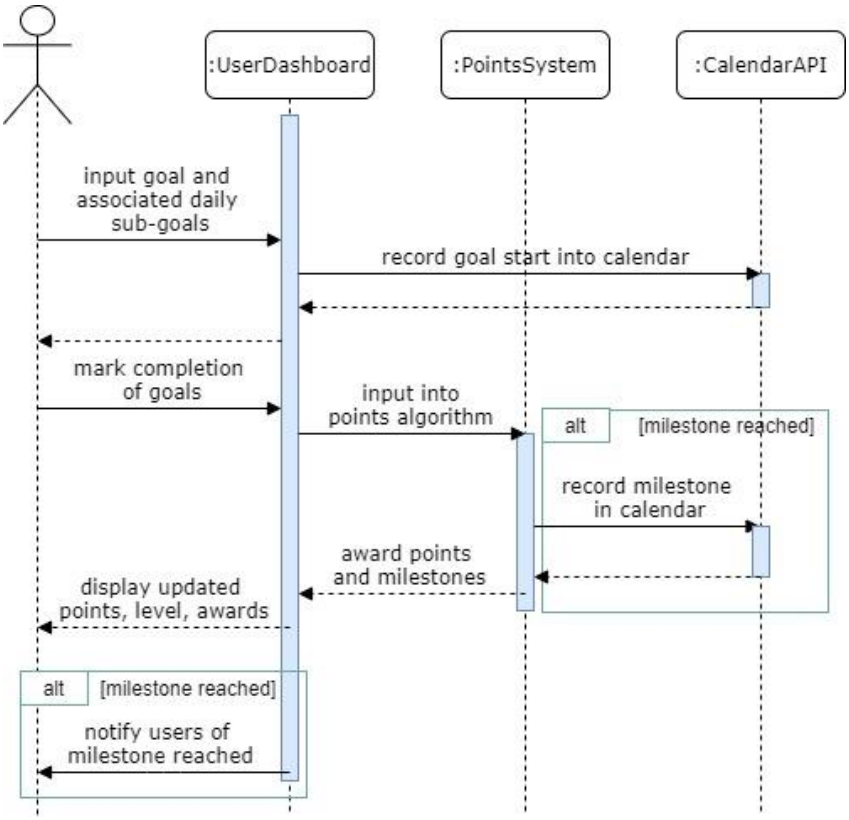
Live Updates



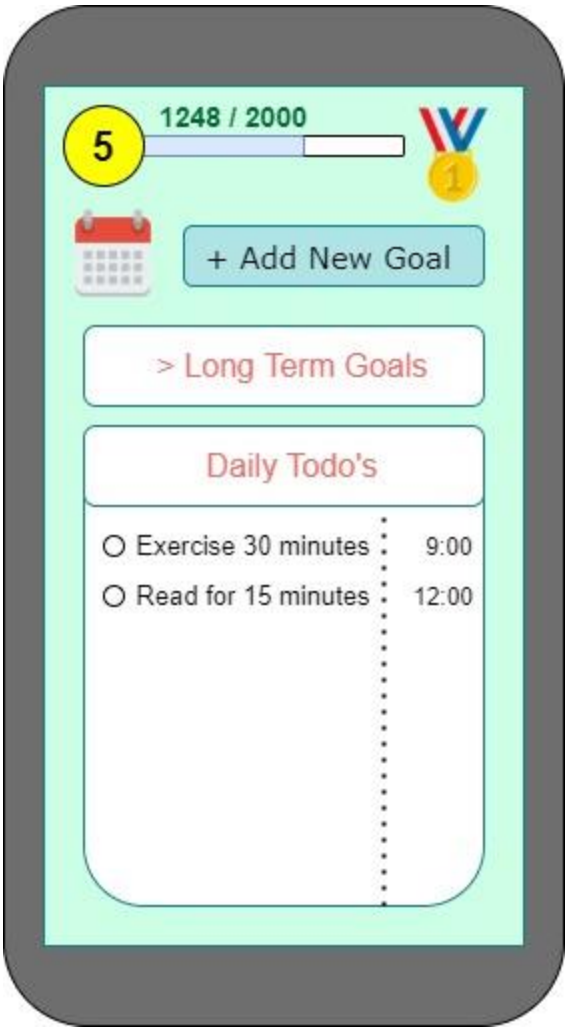
Connection to External API



Non-Trivial Logic



6. **Sketch of the main screen of your app**


5
1248 / 2000
+ Add New Goal
> Long Term Goals
Daily Todo's
Exercise 30 minutes
9:00
Read for 15 minutes
12:00

M4: Design

1. **The main modules of your front-end and back-end components. Explain in one to two sentences the purpose of each module [10 points] o NOTE: identified modules may – but do not have to – directly correspond to the modules you identified in M3. That is because M3 focused on the conceptual system decomposition and M4 focuses on concrete architectural decomposition. o ADVICE: remember the single responsibility and least knowledge principles**

Front End Components:

Calendar: The synced calender will track when users create goals, achieve awards, level up, and complete their goals. It will also prompt users to add a note with each milstone, perhaps to remark their feelings or reflections about their goalDaily todo list

Long-term goals list: Goals that users input, reminds users what their destination is.

Awards cabinet: This screen displays all the awards/milestones users have achieved so far. An example of a milestone could be "30-day streak!," awarded to users if they've accomplished a sub-goal every day for 30 days

Progress meter: A bar at the top of the screen notifying users of how many points they have and how many points they are away from levelling up.

Daily todos: A list of daily sub-goals users need to do in order to reach their long term goals.

Back End Components:

Database of common goals: The app will have a static, unchanging database of common goals and subgoals that users can select or draw inspiration from. For example, a common goal in the database will be "Adopt a healthy diet," and a sub-goal for this goal will be "Cook one meal everyday instead of eating out."

Points Algorithm: The non-trivial backend logic keeps tracks of the user's points and level, awarding milestones when appropriate.

Database of users and their goals: individual to each user, contains the information displayed on the dashboard and calendar, used as input into points system.

2. **Interfaces of each component, including parameters and return values. Explain in one to two sentences the purpose of each interface [20 points] o ADVICE: remember the fan-in/fan-out principle**

Calendar
Event getEvent(Date)
Void putEvent(Date, Event)

Long-term goals list
Void putGoal(String)

Daily todo list
Void putSubGoal(String)

Awards cabinet
Void putAward(Award)

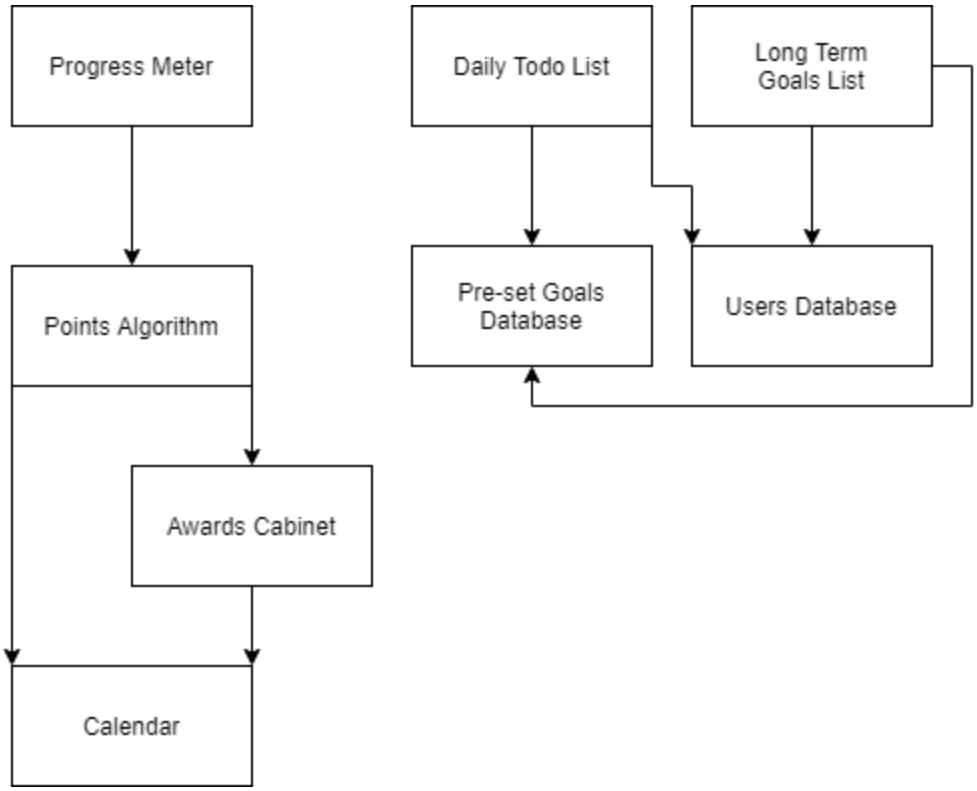
Progress meter
Void putPoints(Points)
Int getPoints()

Database of common goals
Goal getGoal(Query)
Void putGoal(Goal)

Points Algorithm:
Bool levelUp(int Points, User, Goal)
Int pointsAwarded(Goal)
Award getAward(User)

Database of users and their goals:
User getUser(id)
Goals[] getUserGoals(Query)
SubGoals[] getSubGoals(Query)

3. **A diagram showing dependencies between the modules. The "boxes" in this diagram should correspond to the identified modules; the links should show calls between the modules and should show directionality [15 points] o I.e., AàB means that a module A calls B via at least one interface defined in module B. AßàB means that module A calls B via at least one interface defined in module B, and module B calls A via at least one interface defined in module A o NOTE: this is not a class diagram**



4. **Architectural patterns used for the front-end and back-end components. Explain in one to two sentences why you picked these patterns [10 points]**

Overall pattern: Client-server. I am choosing this pattern because it's straightforward and effective in small applications such as document sharing, or in my case, todo list type applciations.

Back-end pattern, front-end pattern: **Model-view-controller pattern.** This pattern allows modularity and decoupling of code, making the development process more organized and the finished code more easy to understand. It also allowed effective code reuse.

5. **Languages and frameworks used (Android/Java or React-Native, MongoDB or MySQL, Cloud provider, etc.). Explain in one to two sentences why you picked them and what alternatives were available [15 points]**

Java: I want to use Java for my app's backend because it is dynamic, robust, and distributed. It's a very safe language that also supports beautiful and useful UI/UX designs. The alternative is Android. I picked Java because it's more flexible in that I can layer any combination of frameworks on top of it.

Node.js: I am using node.js for the backend because it is easy to learn, easy to scale, and is known for offering high performance. There is also a large and active support community. Other alternatives, like PERL, are not as user friendly.

React-Native: React Native is my choice due to its superior performance, increased flexibility, and it is an affordable and efficient solution for app development. Alternatives don't have as many benefits, such as Flutter.

MongoDB: I appreciate MongoDB's tableless, document organization as it allows me to be flexible in describing my data. I do not like the ridigity and upfront commitment of SQL options like MySQL.

Microsoft Azure: I like that Microsoft Azure is secure and easily customizable, making it easy and comfortable to use. Alternatives include Google Cloud and AWS.

6. **Plans to realize each non-functional requirement from M3. Explain in one to two sentences [15 points]**

The user should not need more than 5 clicks for any action: Use React Native to create easy-to-use user interfaces that don't have a lot of components.

The font and graphics should be at least size 12: Use only fonts size 12 and up in react native.

Milestones will be easier to achieve at the beginning and harder later on: Implement using the non-trivial points algorithm – award more points in earlier levels and less points in later levels.

Breaks should be scheduled at least once a week. Breaks too often will disrupt users' momentum: Use the calendar to put in a pre-determined break once a week.

Database queries and updates should take less than 2 seconds: Make efficient queries in MongoDB using well thought out logic implemented in MQL.

7. The algorithm used in your "complex logic" use case. Specify its inputs, outputs, and main computational logic [15 points]

Inputs: User, Goal, current points

Outputs: New number of Points

Main computational logic: Looking at which level the user is at, how many days the user has been pursuing this goal, and the current number of points, the logic will output a new number of points. This method / logic will be called upon the completion of daily sub-goals and upon the completion of long term goals. Based on the new points amount calculated, the logic will then determine whether the user level is increased and what awards the user is eligible for. The logic will award more points for lower levels and less points for higher levels to make the "game" increasing challenging to users!