M4

Amanda Lazar

Project Description

Many cyclists are aware of the importance of keeping their bike in good shape for riding. However, without software, it is difficult to keep track of bike/parts usage and time-consuming to update maintenance history lists. Bike Checkup makes these tasks easier by providing cyclists a secure place to keep track of all bikes and their components, as well as create their own bike maintenance schedule.

M4 Questions

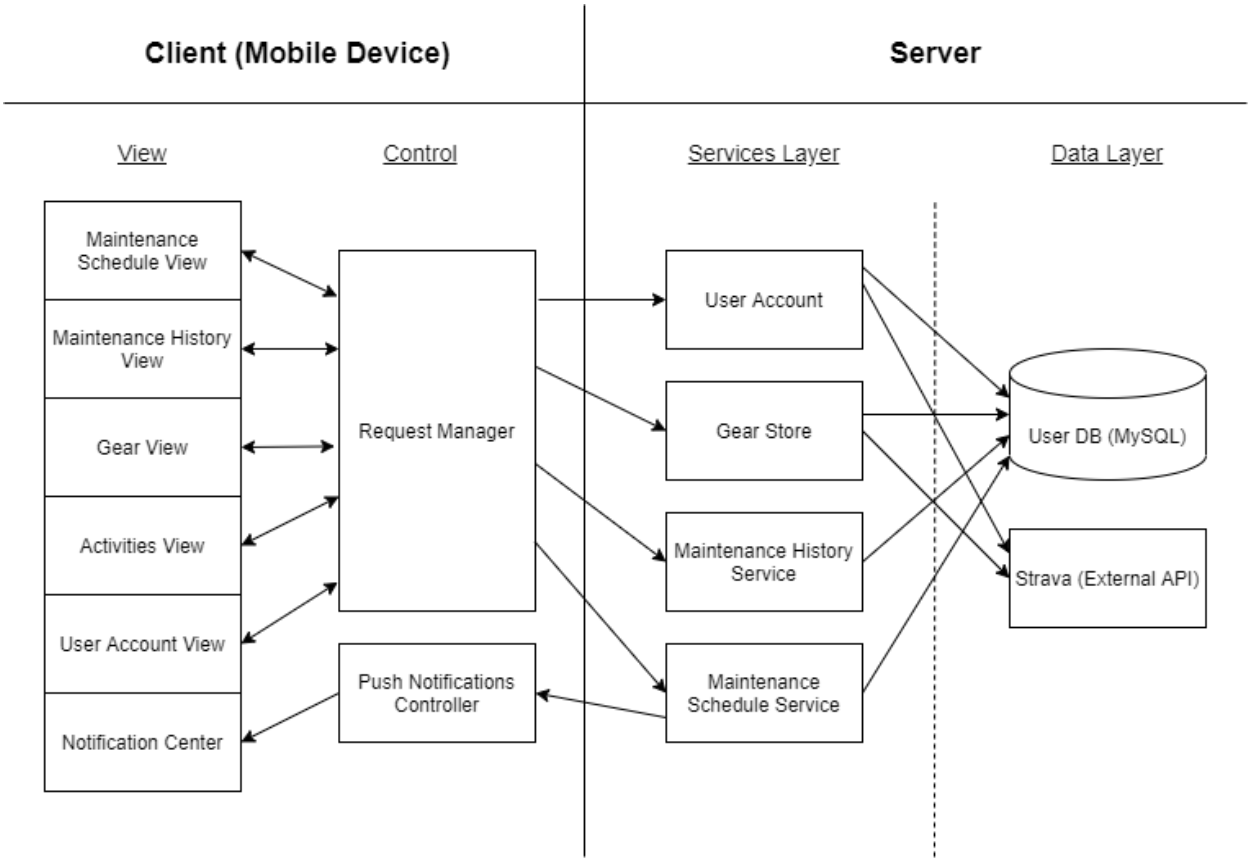
1, 2. Main system modules and interfaces for each:

- **Maintenance Schedule View**
  - View that shows a user's maintenance schedule. Here, the user can also add, remove, and modify the tasks that comprise their maintenance schedule.
  - void addScheduledTask(MaintenanceTask newTask)
  - void removeScheduledTask(MaintenanceTask taskToRemove)
  - void updateScheduledTask(String oldMaintenanceTaskName, MaintenanceTask updatedTask)
- **Maintenance History View**
  - View that shows a user's maintenance history. Here, the user can log maintenance completed.
  - void addCompletedTask(MaintenanceTask completedTask)
  - void removeCompletedTask(MaintenanceTask completedTask)
- **Gear View**
  - View that shows a user's gear, including bikes and their components. Here, the user can add and replace components corresponding to each Strava-imported bike.
  - void addComponent(UserBike bike, BikeComponent component)
  - void replaceComponent(UserBike bike, String componentType, BikeComponent newComponent)

- **Activities View**
  - View that shows a list of a user's recent Strava cycling activities, including relevant information of time and date, distance, and bike used.
  - Activities[] getNewActivities(void)
- **User Account View**
  - View that shows a user's account information, including their connected Strava account and notification preferences.
  - void changeConnectedAccount(void)
  - void updateScheduledMaintanenceNotification(String notice)
- **Notification Center**
  - View that shows a user's recent notifications.
  - void dismissNotification(Notification notification)
- **Request Manager**
  - Handles front-end requests by relaying information to the back-end.
  - void routeToService(Data requestData)
- **Push Notifications Controller**
  - Controller responsible for real-time notifications made by the app, particularly triggered by due maintenance schedule tasks.
  - boolean notificationsDue(void)
  - void triggerNotification(Notification notification)
- **User Account**
  - Service responsible for Strava authentication and user account-related logic.
  - UserID connectStravaAccount(void)
  - void registerUserAccount(DBRecord userEntry)
  - StravaGoal getStravaCyclingGoal(void)
  - void updateScheduledMaintanenceNotification(DBRecord userEntry)
- **Gear Store**
  - Service responsible for updates to user gear and gear status, including Strava-imported bikes and components, as well as polling Strava for updates to gear and activity data.
  - void addComponent(DBRecord bike, DBRecord component)
  - void replaceComponent(DBRecord bike, DBRecord component, DBRecord newComponent)

    - StravaActivity[] getNewActivities(void)
    - int getBikeOdometerValue(UserBike bike)
- **Maintenance Schedule Service**
    - Service responsible for updates to a user's maintenance schedule. Contains the complex logic for predicting maintenance due-dates based on user distance and/or time goals.
    - void addScheduledTask(DBRecord newTask)
    - void removeScheduledTask(DBRecord taskToRemove)
    - void updateScheduledTask(DBRecord oldMaintenanceTask, DBRecord updatedTask)
    - Map<MaintenanceTask, PredictedDueDate> predictDueDates(UserGoal goal)
- **Maintenance History Service**
    - Service responsible for additions to a user's maintenance history.
    - void addCompletedTask(DBRecord completedTask)
    - void removeCompletedTask(DBRecord completedTask)
- **User DB (MySQL)**
    - Relational database responsible for storing all user-related information. A table containing users will link to other tables such as bikes, components, maintenance schedule, maintenance history, and activities.
    - void insert(SQLQuery query)
    - void remove(SQLQuery query)
    - void modify(SQLQuery query)
    - DBRecord find(SQLQuery query)
- **Strava (External API)**
    - Third-party API which, after user authentication, will provide user data including bike gear and odometer data, activity data, and optionally goal/stats data.
    - GET /gear/{id}
    - GET /activities/{id}
    - GET /athletes/{id}/stats

3. High-level app architecture:



4. Architectural patterns used for the front-end and back-end components:
    - Client-Server
        - I chose this pattern because it is inherent to the architectural design of our projects, being that we have a mobile application acting as our client and our backend serving this frontend.
    - Model-View-Controller (MVC)
        - I chose this pattern because it is intuitive to me and my previous experience with app development, namely having the “middleman” of a controller handling interactions between frontend (views) and backend (models).

5. Language and framework decisions:
    - Database choice: MySQL
        - I chose to use a relational database over a non-relational database because the data I will be storing can be stored in a structured way, and

given the scope of my project there is no need for additional flexibility. The structure of relational databases are simple and uniform, therefore supporting the “KISS” design principle. In addition, I will not be working with massive amounts of data, so a non-relational database is not advantageous to my particular project in this way.

- Front-end choice: React Native
    - I chose React Native over native Android because prior research into my app idea revealed that similar apps do exist, however no one app supports both iOS and Android. Therefore by going with React Native, I will be giving my app a competitive advantage if I end up releasing to the app stores.
- Cloud provider choice: AWS (Amazon)
    - I chose AWS over other leading cloud platforms such as Azure (Microsoft) and Google Cloud because it has an established reputation as the oldest and most experienced player in the cloud market. As one of the oldest cloud providers, AWS has established a larger customer base, as well as larger trust and reliability factors. Across multiple detailed comparisons of cloud providers, Amazon tends to edge out the rest. I have also not had experience using AWS, to this is a great opportunity for me to learn how it works.

6. Plans to realize each non-functional requirement in M3:
    - Security: App's Strava API "secret" used for authentication is stored in a file that is encrypted prior to being pushed to the source tree. This can be accomplish using the “git-secret” tool.
    - Usability: App design takes accessibility into account by having sufficient contrast, acceptable touch target sizing and spacing, and dynamic text sizing capabilities. To accomplish this, the app will be developed in accordance with the targets outlined in Material Design's accessibility guidelines, namely choosing a color scheme using Material Design's Color Tool to achieve sufficient contrast, having touch targets be at least 48 x 48 dp in size and separated by at least 8 dp of space, and marking text and their associated containers to be measured in scalable pixels to enable system font size [in an Android app].

7. Algorithm used in my “complex logic” use case:
    - Inputs:
        - A user-set goal for distance or time cycling per a given time interval
    - Output:
        - Predicted due-date of all user maintenance tasks
    - Main computational logic:
        i. Acquire cycling goal data either from Strava (if the user has a premium Strava account with this field set) or from an in-app prompt (for example, “30km per week”)
        ii. Normalize goal data (for example, into “km per day”)
        iii. For each maintenance task in the user’s maintenance schedule:
            a. Get the task’s user-chosen recurrence interval (for example, “every 100km” for the “oil chain” task)
            b. Normalize task recurrence interval into complementary units to normalized goal value
            c. Using the normalized goal and recurrence interval values, calculate the expected amount of time until the task is due
            d. Based on the current time/day, convert the above value to an actual due-date for the maintenance task