Design Document for CyTrack

Group 3_Mahdi_3

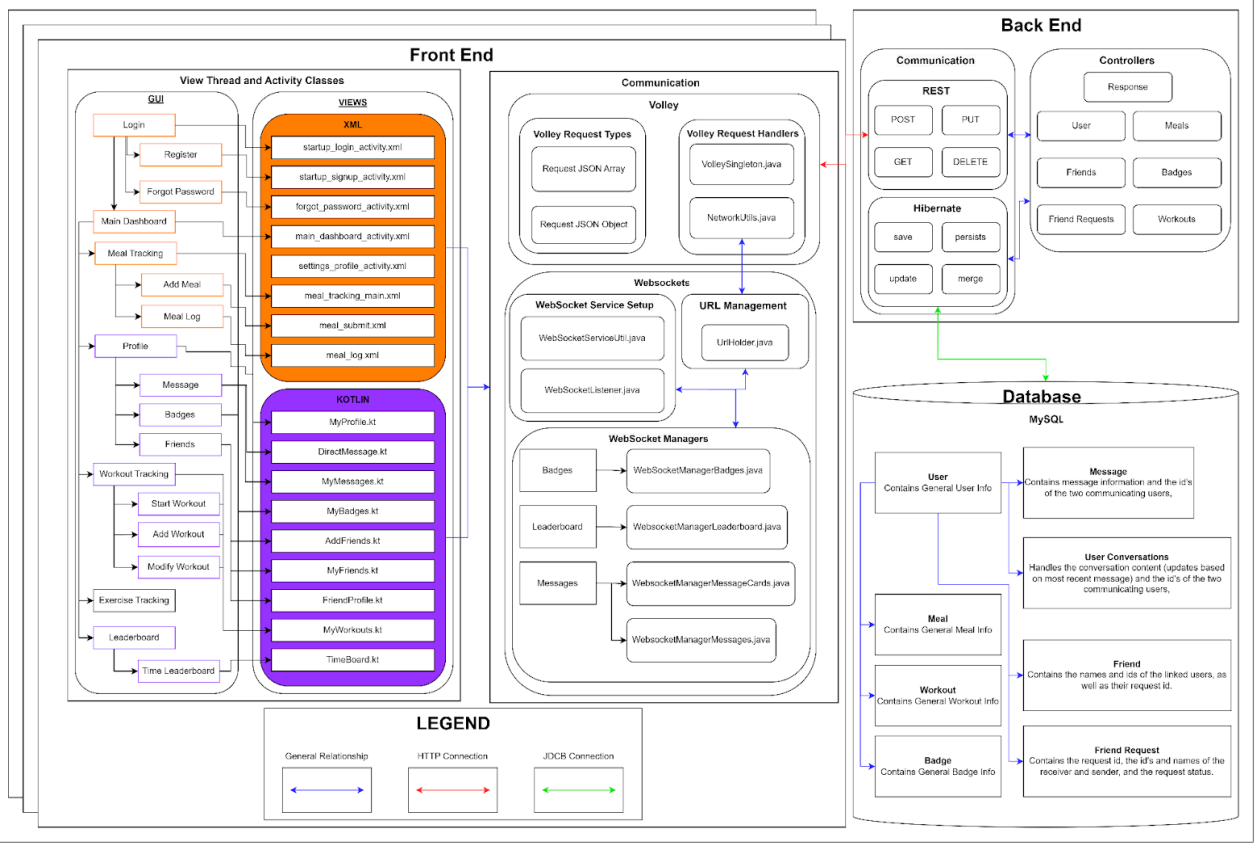
Andrew Boun: 25 % contribution

Kai Quach: 25% contribution

Eduardo Barboza-Campos 25% contribution

Ethan Cabelin: 25% contribution

CyTrack


Front End
View Thread and Activity Classes
GUI
Login
Register
Forgot Password
Main Dashboard
Meal Tracking
Add Meal
Meal Log
Profile
Message
Badges
Friends
Workout Tracking
Start Workout
Add Workout
Modify Workout
Exercise Tracking
Leaderboard
Time Leaderboard
VIEWS
XML
startup_login_activity.xml
startup_signup_activity.xml
forgot_password_activity.xml
main_dashboard_activity.xml
settings_profile_activity.xml
meal_tracking_main.xml
meal_submit.xml
meal_log.xml
KOTLIN
MyProfile.kt
DirectMessage.kt
MyMessages.kt
MyBadges.kt
AddFriends.kt
MyFriends.kt
FriendProfile.kt
MyWorkouts.kt
TimeBoard.kt
Communication
Volley
Volley Request Types
Request JSON Array
Request JSON Object
Volley Request Handlers
VolleySingleton.java
NetworkUtils.java
Websockets
WebSocket Service Setup
WebSocketServiceUtil.java
WebSocketListener.java
URL Management
UrlHolder.java
WebSocket Managers
Badges
WebSocketManagerBadges.java
Leaderboard
WebsocketManagerLeaderboard.java
Messages
WebsocketManagerMessageCards.java
WebsocketManagerMessages.java
LEGEND
General Relationship
HTTP Connection
JDCB Connection
Back End
Communication
REST
POST
PUT
GET
DELETE
Hibernate
save
persists
update
merge
Controllers
Response
User
Meals
Friends
Badges
Friend Requests
Workouts
Database
MySQL
User
Contains General User Info
Message
Contains message information and the id's of the two communicating users,
User Conversations
Handles the conversation content (updates based on most recent message) and the id's of the two communicating users,
Meal
Contains General Meal Info
Workout
Contains General Workout Info
Friend
Contains the names and ids of the linked users, as well as their request id.
Badge
Contains General Badge Info
Friend Request
Contains the request id, the id's and names of the receiver and sender, and the request status.

FrontEnd

LoginActivity

- LoginActivity creates a page displaying the following entry options
    - TextInputEditText: Username
    - TextInputEditText: Password
    - Button: signup_button
    - Button: login_button
    - Button: forgotPassword_button
- When the user logs in they can input their username and password into the fields and submit them by pressing “login”. Users cannot log in unless all fields are filled. From there, a call is sent to our NetworkUtils class, a helper class that handles all the JSON requests sent from the front end. The call attempts to POST the mapped fields in a JSON format and GET matching user information from the database. If the user does not exist, they must create an account to POST a new entry into our users table via the signup page. If the user does exist then they will be directed to the MainDashboard. If the user doesn’t remember their password, they send a PUT request to their account via the forgot password page.

SignUpActivity

- SignUpActvitity creates a page displaying the following entry options
    - TextInputEditText: InputUsername
    - TextInputEditText: InputPassword
    - TextInputEditText: InputPasswordAgain
    - TextInputEditText: InputFirstName
    - TextInputEditText: InputLastName
    - TextInputEditText: InputAge
    - Spinner: genderSpinner
    - Button: SignUp
    - Button: Goback
- From the log-in page, the user can access the Sign-Up Activity. From here they can input the necessary fields for their account. All fields must be filled and valid to sign up and post a new user into the database. If sign-up is successful the user is redirected to the login page.

FrontEnd

ForgotPasswordActivity

- ForgotPasswordActivity creates a page displaying the following fields
    - TextInputEditText: InputUsername
    - TextInputEditText: InputPassword
    - TextInputEditText: InputPasswordAgain
- From the log-in page, the user can access the Sign-Up activity. From here they can input the necessary fields for their account. All fields must be filled and valid to PUT the new password into the linked user in the database. If the reset is successful the user is redirected to the login page.

MainDashboardActivity

- MainDashboardActivity creates a page that provides access to all our other pages like WorkoutTracking, MealTracking, Leaderboard, Profile, and a top bar showing the User's name and Streak.

MealTrackingMain & (MealSubmit and MealLog)

- MealTrackingMain is accessible from the Main Dashboard and gives access to Meal creation, Meal Logs, and adding a favorite Meal. Albeit not yet implemented, the page will also show the latest meal added from the log.

MealSubmit

- MealSubmit is accessible from the Meal Dashboard, and creates a page displaying the following fields…
    - TextInputEditText: MealName
    - TextInputEditText: MealCalories
    - TextInputEditText: MealProtein
    - TextInputEditText: MealCarbs
    - Button: Submit
- When the user opens this page, they can create a meal by filling out all the necessary fields like their Meal's name and nutrition information. If all fields are filled out they can submit this request, which passes a call to our NetWorkUtils to POST a new meal into our Meals Table. The formatting for this data is done via a HashMap that collects the fields. These parameters are passed into the NetworkUtils class to create and send a new JSON object in a specified method type and URL.

FrontEnd

MyProfile

- MyProfile is accessible from the Main Dashboard and gives access to Friends, Messages, Badges, and a view of the User's received friend requests.

Friends

- Friends is accessible from your profile and gives access to your current conversations, an entity pulled from our database. Only conversations with a matched Friend are displayed. This entity gives data that lets us display a card featuring the name of the friend our user is messaging, a button to redirect us to the saved conversation, and a dropdown menu to delete the friend. Deleting the friend sends a DELETE request to the Friends table, and locates the paired user id parameters of the requestee and their friend to delete the entry from "Friends".

DirectMessage

- DirectMessage is accessible from the Friends Page and Messages. The display uses a MutableList and a Websocket to GET the conversation information live between the two senders. At the bottom of the page, the user can enter and send a message to the receiver. Whenever the conversation is updated with new appended messages, the WebSocket verifies whether it can post and access the updated data, and outputs our specified error or success logs. All messages are saved and appended into a MutableList as a Message Object.

Messages

- Messages are accessible from Profile and give access to all of your current conversations, an entity pulled from our database. Similarly to Friends your conversations are displayed in a card and can redirect you to a DirectMessage with the selected user. A dropdown menu to delete the conversation is available. Deleting the conversation sends a DELETE request to the conversations table, and locates the paired user ID parameters of the requestee and their receiver.

Badges

- Badges are accessible from the Profile. The display uses a MutableList and a Websocket to GET the user's badge information live. Whenever the list is updated with new appended badges, the WebSocket verifies whether it can access the updated data, and outputs our specified error or success logs. All badges are saved and appended into a MutableList as a Badge Object.

FrontEnd

MyWorkouts

- MyWorkouts is accessible from the Main Dashboard. When opened, the page GETs the User's "Workout" data and displays how many calories they have burned, and the time they have worked out. At the bottom, the user has a mutable list that displays all of their workouts. This data is pulled from the Workout Table, and selected based on a linked userID key field. Above the mutable list are two buttons, "Start Workout" and "Add Workout", these two pages allows the user to create and POST workouts.

StartWorkout

- StartWorkout is accessible from MyWorkouts and acts as a dynamic option for adding a workout. The display showcases four entry options…
    - MutableLabel: ExerciseType
    - MutableLabel: Duration(Minutes)
    - MutableLabel: Calories Burned
    - MutableLabel: Date
    - Button: Start Workout
    - Button: End Workout
- The user must input all of the necessary fields to post this workout. On "Start Workout" the workout is posted to a special URL where the backend starts ticking down a timer. On "Workout End" the timer is stopped and the duration is updated with a POST.

AddWorkouts

- AddWorkout is accessible from MyWorkouts and acts as a more formal option for adding a workout. Similarly to MyWorkouts, a mutable list of all the user's workouts is displayed. Each card can also be clicked to modify the workout information and PUT the new information into the database. A "Add Workout" button is displayed at the bottom of the screen, and showcases three entry options…
    - MutableLabel: WorkoutName
    - MutableLabel: Duration(Minutes)
    - MutableLabel: Calories Burned
- The user must input all of the necessary fields to POST this workout, which will be displayed on the mutable list in MyWorkouts and AddWorkouts on reload.

Leaderboards

- Leaderboards are accessible from the Main Dashboard. The display uses a MutableList and a Websocket to GET the Leaderboard information of the selected leaderboard. For now, the only leaderboard implemented displays a high-low hierarchy of users ordered by the highest time spent working out. Whenever the leaderboard is updated with new user entries, the WebSocket verifies whether it can access the updated data, and outputs our specified error or success logs. In its current implementation, a Leaderboard is completely reloaded from our view with the new leaderboard data.

Backend

Communication

The backend uses CRUDL mappings to update the database based on information sent to the given mappings' URL. These include:

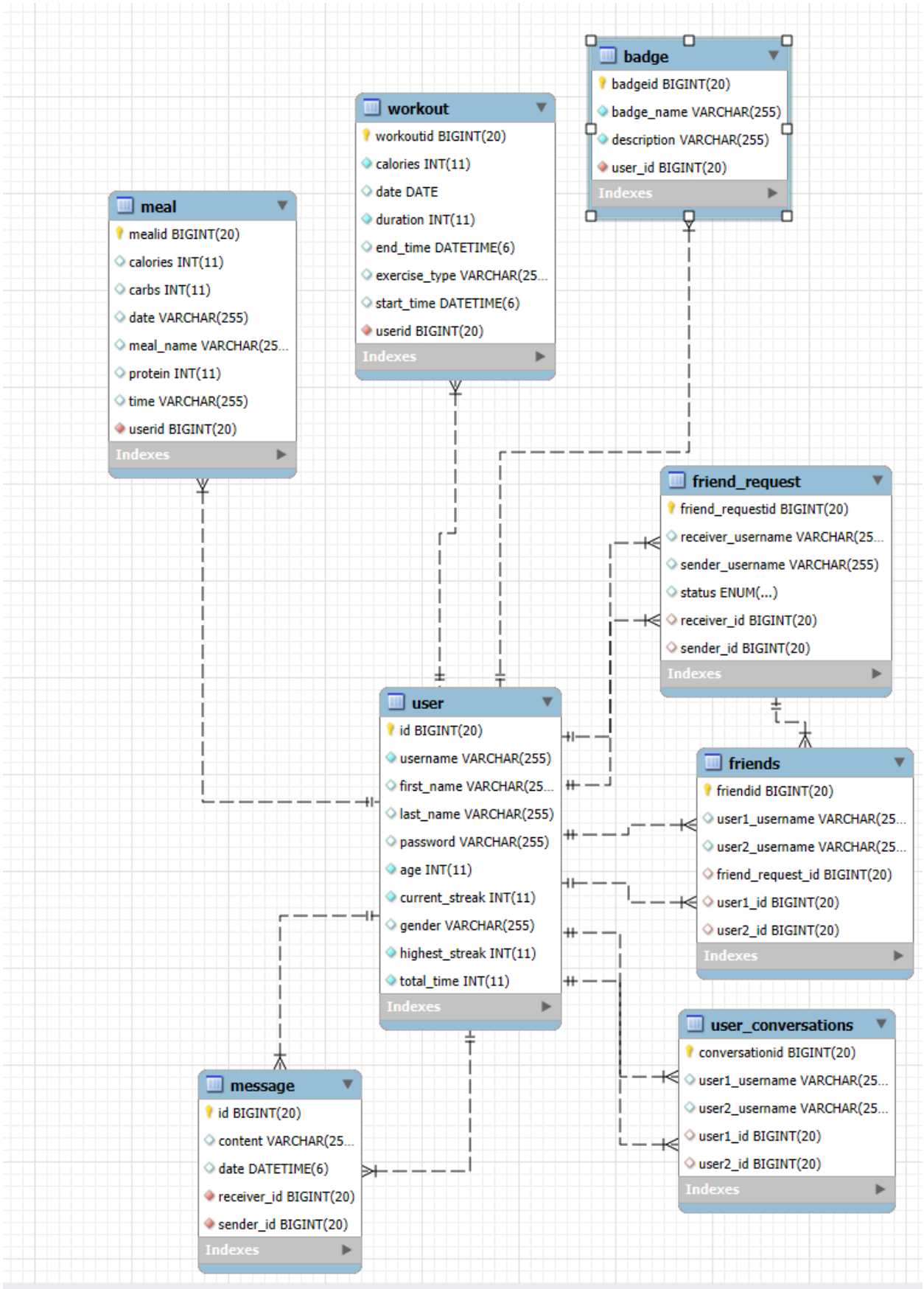
- Post: Sends information on an item to be added to the database.
- Get: request information, often with an identifier for the specific item requested from the database.
- Put: send information to update a specific item in the database
- Delete: send an identifier to delete a specific item from the database

Controllers

The controllers contain mappings for communications between the frontend and the entities in our database.

These entities include:

- User: Uses the above mappings for operations with users. Contains one to many relationships with Workout, Meal, Badge, and FriendRequest. Also has a many to many relationship with Friend.
- Workout: Contains info for a User's specific workout. Users can "log/start" their workouts, which automatically calculates duration, updates fields for related entities, and awards badges.
- UserConversations: Handles conversations betweens two users. Constantly updates with the most recent message between the two users.
- Friends: When a friend request has been accepted, it is then sent to the friends database that displays the two users who are friends.
- FriendRequest: Handles sent requests between two users. Holds data from the receiver and sender. Includes the statuses PENDING, ACCEPTED, and DECLINED.
- Message: Currently, the messaging system handles messages between two users. The database includes information about who sends the messages, who is receiving the message, when it was sent, and the content of the message. (The goal is to include group chats).
- Meal: Has many to one relationship with users. Users can "log" their meals for a given date for tracking. Contains nutritional info, like calories and protein.
- Badge: Has many to one relationship with users. Users are automatically awarded badges upon meeting specific criteria, such as reaching milestones for total workout time.


badge
badgeid BIGINT(20)
badge_name VARCHAR(255)
description VARCHAR(255)
user_id BIGINT(20)
Indexes
workout
workoutid BIGINT(20)
calories INT(11)
date DATE
duration INT(11)
end_time DATETIME(6)
exercise_type VARCHAR(25...
start_time DATETIME(6)
userid BIGINT(20)
Indexes
meal
mealid BIGINT(20)
calories INT(11)
carbs INT(11)
date VARCHAR(255)
meal_name VARCHAR(25...
protein INT(11)
time VARCHAR(255)
userid BIGINT(20)
Indexes
friend_request
friend_requestid BIGINT(20)
receiver_username VARCHAR(25...
sender_username VARCHAR(255)
status ENUM(...)
receiver_id BIGINT(20)
sender_id BIGINT(20)
Indexes
user
id BIGINT(20)
username VARCHAR(255)
first_name VARCHAR(25...
last_name VARCHAR(255)
password VARCHAR(255)
age INT(11)
current_streak INT(11)
gender VARCHAR(255)
highest_streak INT(11)
total_time INT(11)
Indexes
friends
friendid BIGINT(20)
user1_username VARCHAR(25...
user2_username VARCHAR(25...
friend_request_id BIGINT(20)
user1_id BIGINT(20)
user2_id BIGINT(20)
Indexes
user_conversations
conversationid BIGINT(20)
user1_username VARCHAR(25...
user2_username VARCHAR(25...
user1_id BIGINT(20)
user2_id BIGINT(20)
Indexes
message
id BIGINT(20)
content VARCHAR(25...
date DATETIME(6)
receiver_id BIGINT(20)
sender_id BIGINT(20)
Indexes