React Native 2 α

New Attempt

Due Nov 19 by 11:59pm **Points** 6 **Submitting** a text entry box
Available after Nov 9 at 6pm

Homework: React Native 2 α (6 Points Total, 2 Points Extra Credit)

This assignment will continue on the essentials of React Native, Expo, and RESTful APIs through the implementation of our prototype from React Native 1 β. While we do not require you to implement React Native 2 α exactly as you prototyped it, you should draw inspiration from your prototype. Specifically, you will be building a day view, a goals comparison view, and a way to create/update/delete both exercises. Additionally, starter code has been provided, which is an example solution for React Native 1 α. Note that this starter code is *optional*, you may disregard it and replace it with your own code from React Native 1 α if you wish.

Setup Information

PLEASE READ: If you have not submitted your React Native 1 α yet, DO NOT accept the GitHub Classroom invitation for React Native 2 α. Your React Native 1 α submission will be rejected if it is turned in after you accept the invitation below.

GitHub Starter Code (https://classroom.github.com/a/UeT-a7CC) (Submit your React Native 1 α before accepting the GitHub Classroom invitation!)

Note: This starter code is *optional*, you may disregard it and continue with your own code if you wish. Because of differences in expo and node/npm versions, you may have to install, update, remove, and change some of the packages used.

Submission Details

IMPORTANT: DO NOT rely on web browser simulation of Expo when testing your app; it is unable to detect potential errors that may arise when actually run on a mobile device. You may end up losing points if some features do not work properly on an emulator/physical device.

When pushing your work to the repository, don't forget to include the `.gitignore` file automatically generated from Expo. React Native is intended to build applications that work across all platforms, though we realize that you may only be able to test on Android or iOS due to your system

constraints. **Please specify in your Canvas submission which operating system you have tested on.**

When your assignment is ready for grading, please **submit your tested operating system, repository name, and latest commit hash** from GitHub Classroom. e.g. `android, react-native2-alpha-osori, 81d0462`.

Program Requirements

Problem 1 (0.5 Points)

Provide the user with an option to sign out and return to the login screen at any point while they are logged in to the application.

- Helpful Links: **Header buttons** (https://reactnavigation.org/docs/header-buttons)

Problem 2 (2 Points + 0.25 Extra Credit Points)

Provide the user with a "Today View" with the ability to view their exercises logged for today. The ways to create, update, and delete exercises will be detailed in Problem 4.

- The user should be able to see their exercises for today, including the name, duration, and the total number of calories burned for each exercise.
- The Today View should be updated in sync with the Exercises View; if a change is made to an exercise in the Exercises View, it should be reflected in the Today View. Similarly, if an exercise is added or created in the Exercise View, it should be added or deleted in the Today View.

(0.25 Extra Credit Points) The user should also be able to see their meals for today, including the foods and the total number of calories and macronutrients for each meal. **The extra credit problem must be completed first to be eligible for these points.**

Problem 3 (1 Point + 0.25 Extra Credit Points)

Provide the user with the ability to compare today's activity minutes versus their daily goal activity minutes. This can be made a part of the "Today View" from Problem 2 or be made as a separate view.

- The user should be able to compare their total daily activity minutes (calculated as the sum of all exercise activity) versus their daily goal activity minutes.
- The view should be updated in sync with the Profile View; if a change is made to the user's goals, it should be reflected in their comparison.

(0.25 Extra Credit Points) The goals comparison should also show the difference in their overall calories and macronutrients. **The extra credit problem must be completed first to be eligible for**

calories and macronutrients. **The extra credit problem must be completed first to be eligible for these points.**

Problem 4 (2 Points)

Provide the user with the ability to create a new exercise or edit/delete any of their past exercises. An exercise consists of a name (e.g., "Jogging"), duration, date, and the number of calories burned; see the API for further details.

- The user should be able to see and specify the name, duration, and the number of calories burned for each exercise.
- The user should be able to indicate if the exercise was done at the current time or at some other time.

Final Criteria (0.5 Point)

- Appropriately utilize icons or graphics to visually aid in the user experience; for example, you may want to have a graphic of a person exercising in the exercises view or a graphic of food in the meals view. **Check the approved libraries (https://hci-curriculum-uwmadison.github.io/CS571/packages.html)** for pre-made icons and graphics! **(0.25 points)**
- Use React Navigation in order to help the user navigate between views; this could include tabs, drawers, stacks, or any of the navigation techniques outlined in the **React Native 2 lecture (https://hci-curriculum-uwmadison.github.io/CS571/lectures/08-Build-React-Native-2.pdf)**. **(0.25 points)**

Extra Credit Problem (1.5 points)

Provide the user with the ability to create a new meal, or edit/delete any of their past meals. A meal consists of a name (e.g., "Lunch"), a date, and a list of foods; see the API for further details.

- The user should be able to see and specify 0-to-many foods that they ate for that meal.
- The user should be able to see the total calories and macronutrients consumed for that meal.

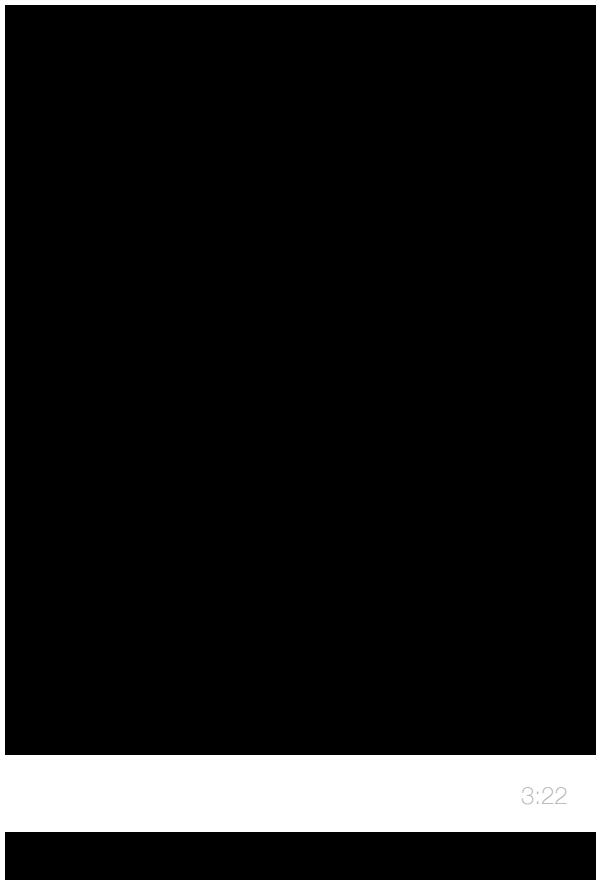
- The user should be able to indicate if the meal was eaten at the current date and time or at some other date and time.

Example Walkthrough

The below video walks through a scenario where...

1. Sign up for a new account

2. Log in with the new account
3. See blank today view
4. Specify an activity goal
5. See activity goal reflected in the today view
6. Add an exercise for the current day
7. See the exercise in the today view
8. Add an old exercise from another day
9. Do not see the old exercise in the today view
10. Update today's exercise and see updates reflected in the today view
11. Delete all exercises, and add a single exercise for today
12. See the exercise in the today view
13. Sign out from one screen
14. Log back in
15. Verify progress is saved
16. Log out from another screen



0:00 3:22 1x

API Information

Import.png

For your convenience, a collection (v2.1) of the Postman requests can be downloaded here (https://canvas.wisc.edu/users/146319/files/22892231/download?verifier=ZupPmBuSQ5tmuYkIvoe81I3PskqJATa6xxntCp5V&download_frd=1) . These can be imported into your environment, see these instructions (https://learning.postman.com/docs/getting-started/importing-and-exporting-data/) . Also refer to this detailed documentation here (https://documenter.getpostman.com/view/2613990/UVC5Enhe) .

The following API can be accessed at `http://cs571.cs.wisc.edu:5000`.

Route	Auth Required	Token Required	Get	Post	Put	Delet
/login/	✔		✔			
/users/				✔		
/users/`<username>`		✔	✔	✔	✔	✔
/activities/		✔	✔	✔		
/activities/`<activity_id>`		✔	✔		✔	✔

Route	Auth Required	Token Required	Get	Post	Put	Delete
/meals/		✔	✔	✔		
/meals/`<meal_id>`		✔	✔		✔	✔
/meals/`<meal_id>`/foods/		✔	✔	✔		
/meals/`<meal_id>`/foods/`<food_id>`		✔	✔		✔	✔
/foods/			✔			
/foods/`food_id`			✔			

Auth and Tokens

For this API, users need to provide credentials in order to access information specific to themselves. They get these credentials by requesting tokens, which are short-lived codes which tell the server that you are who you are saying you are, without having to provide a username and password each time. The steps to get these tokens are outlined below.

Signup

This can be done with a `POST` request to the `/users` route. You will need to tell the API a bit about the user. You should provide this data in the message body (stringified) in the following form:

```
{username:<str>,                 // Required
 password:<str>,                 // Required
 firstName:<str>,                // Optional
 lastName:<str>,                 // Optional
 goalDailyCalories:<float>,      // Optional
 goalDailyProtein:<float>,       // Optional
 goalDailyCarbohydrates:<float>, // Optional
 goalDailyFat:<float>,           // Optional
 goalDailyActivity:<float>       // Optional
}
```

Only the `username` and `password` fields are required. Don't worry about the other ones for creating a user, as they can be updated later with `PUT` requests.

If the user is successfully created, you will receive a positive message back from the server. You will then need to log in with that user.

Login

You can do this via the `/login` route with a `GET` request. You will need to send the username and password in the authorization header using the header `Authorization` with value 'Basic {base64enc(username:password)'. e.g. 'Basic bXl1c2VyOnBhc3MxMjM0' for a hypothetical user 'myuser' with password 'pass1234'.

You will receive back a token that you can use to access information from the API. The token you receive can then be added in the `x-access-token` header.

User

Users cannot query `/users`, since that would involve exposing all the other users' data. Instead, they must get/modify the information for their user separately. They do this by using

must get/modify the information for their user separately. They do this by using the `/users/<username>` route, where `<username>` is filled in with their actual username. Suppose your username is `Fred639`, then you could fetch (`GET`) `/users/Fred639` to get the information about yourself, but only if you provide the right token. Likewise, you can `PUT` to `/users/Fred639` to modify your goals, by providing the changes in the form of a `json`. You can modify the following fields:

- `pasword`
- `firstName`
- `lastName`
- `goalDailyCalories`
- `goalDailyProtein`
- `goalDailyCarbohydrates`
- `goalDailyFat`
- `goalDailyActivity`

Additionally, you can delete unused users using the `DELETE` method on the `/users/<user_id>` route. As articulated in the Final Notes, please be a good citizen and clean up after yourself.

User Activities

You can use the routes `/activities` and `/activities/<activity_id>` in conjunction with relevant `GET`, `POST`, `PUT` and `DELETE` methods. The request and response structure of each of the individual activities follows the form...

```
{
 id:<int>,
 name:<str>,
 duration:<float>, // Minutes
 date:<isostring>,
 calories:<float>
}
```

In particular...

A `GET` to `/activities` will return a list of all that user's activities.

A `POST` to `/activities` will create a new activity for that user.

A `GET` to `/activities/<activity_id>` will get the details for a specific activity.

A `PUT` to `/activities/<activity_id>` will update the details for a specific activity

A `DELETE` to `/activities/<activity_id>` will delete a specific activity.

Note that we do *not* pass the `username` to these endpoints. The API is able to determine the appropriate user through the specified `x-access-token`

Extra Credit: User Meals

Each user can track their meals across days. These are interacted with using the `/meals` routes. Despite not including the user in the route, you will only be able to retrieve the information for your current user, since the token tells the API who you are. Unlike `/users`, you can request the full set of meals using a `GET` on `/meals`. This will return an array-like object like this:

```
{meals:[
 {id:<int>,
  name:<str>,
  date:<isostring>
 },
 {id:<int>,
  name:<str>,
  date:<isostring>
 }]
}
```

You can also access an individual meal by `meal_id` (`id`), using `GET` on `/meals/<meal_id>`. This will return the single meal data object:

```
{id:<int>,
 name:<str>,
 date:<isostring>
}
```

Posting to `/meals` creates a new meal. If not specified in ISO format, the `date` will default to the current date and time. You cannot specify the `id`, only the `name` and `date`. Using the `PUT` method is possible for the `/meals/<meal_id>` route, such that you can modify the `name` and `date`.
Using `DELETE` on an individual meal removes the item (and all associated foods).

Extra Credit: User Foods

You may notice that the meal doesn't have any food data. That is because they are accessed with a deeper route, `/meals/<meal_id>/foods` and `/meals/<meal_id>/foods/<food_id>`. These behave similarly to the meals, where using `/meals/<meal_id>/foods` returns the list of all foods associated with that meal:

```
{foods:[
 {id:<int>,
  name:<str>,
  calories:<float>,
  protein:<float>,
  carbohydrates:<float>,
  fat:<float>
 },
 {id:<int>,
  name:<str>,
  calories:<float>,
  protein:<float>,
  carbohydrates:<float>,
  fat:<float>
 }]
}
```

A single food can be accessed by `id`, using `/meals/<meal_id>/foods/<food_id>`, returning the single object:

```
{id:<int>,
 name:<str>,
 calories:<float>,
 protein:<float>,
 carbohydrates:<float>,
 fat:<float>
}
```

Note that this does not return the `measure` of the foods. Like meals, you can modify all attributes with `PUT`, other than the `id`. Using `DELETE` on an individual food removes the item.

Extra Credit: Food Library

A list of foods can be accessed with `GET` methods to `/foods` and `/foods/<food_id>`. They appear similar to foods in meal, with an additional attribute of `measure`, which tells you how much of each the nutritional values are associated with, and what the grouping name is called. For example, bread comes in `slices`:

```
{id:2,
 name:"whole wheat bread",
 measure:"slice",
 calories:69.0,
 protein:3.6,
 carbohydrates:12.0,
 fat:0.9
}
```

Final Notes

While the server API articulated above does adhere to standard conventions in security, you should not assume that it is completely secure. As such, do not use sensitive usernames and passwords for your testing. If you need to create a number of users for testing your login/signup, please be a good citizen and `DELETE` your unused users, so that the server does not get bogged down.

React Native 2 α

Criteria	Ratings			Pts
P1: Signout	**0.5 pts** **Full Marks** The user is able to signout of the application.	**0.5 to >0.0 pts** **Partial Marks** There is an attempt at functionality to sign the user out of the application.	**0 pts** **No Marks** Little to no attempt is made to sign the user out of the application.	0.5 pts
P2: Today View	**2 pts** **Full Marks** There exists a functioning view in which the user can see their activities for today. For each exercise, the user is able to view their name, duration, and the total number of calories burned. The activities are updated in sync with other views.	**2 to >0.0 pts** **Partial Marks** There exists a semi-functioning view in which the user can see their activities for today. For each exercise, the user may be able to view their name, duration, and the total number of calories burned. The activities may be updated in sync with other views.	**0 pts** **No Marks** Little to no attempt is made for a Today View	2 pts
P3: Goals Comparison	**1 pts** **Full Marks** There exists a functioning view in which the user can compare their goals to today's progress. The comparison shows the difference between today's progress and the goal value.	**1 to >0.0 pts** **Partial Marks** There exists a semi-functioning view in which the user can compare their goals to today's progress. The comparison shows the difference between today's progress and the goal value, but it may not be accurate.	**0 pts** **No Marks** Little to no attempt is made for a Goals Comparison View	1 pts

Criteria	Ratings			Pts
P4: Exercise Management	**2 pts** **Full Marks** There exists functionality for a user to be able to add new exercises or edit/delete existing exercises. Each exercise is allowed to have a name, duration, and the number of calories burned. A date and time can also be specified for each exercise.	**2 to >0.0 pts** **Partial Marks** There exists a semi-functional functionality for a user to be able to add new exercises or edit/delete existing exercises. Each exercise is allowed to have a name, duration, and the number of calories burned. A date and time may also be specified for each exercise.	**0 pts** **No Marks** Little to no attempt is made for being able to manage exercises	2 pts
FC1: Icons/Graphics	**0.25 pts** **Full Marks** The application properly utilizes icons or graphics in at least one view.	**0 pts** **No Marks** The application does not utilize icons or graphics at all.		0.25 pts
FC2: React Navigation	**0.25 pts** **Full Marks** The application makes use of React Navigation; using tabs, a drawer, stacks, or any navigation technique outlined in the React Native 2 lecture.	**0 pts** **No Marks** The application does not make use of React Navigation.		0.25 pts
				Total Points: 6