CSCI 0320 Project Specs

Task 1 Introduction

a) General Information:

Project name:

AiROBIC

Team members:

1. cbaker20: backend, working on developing the markov model/graph APIs
   - week 1 time: 10 hours (developing the initial Markov model/graph APIs)
   - week 2 time: 8 hours (refining APIs and addressing any issues)
   - week 3 time: 5 hours (finalizing APIs and assisting with integration)
2. avonbism: frontend, working on design and CSS
   - week 1 time: 12 hours (designing the user interface and setting up CSS framework)
   - week 2 time: 10 hours (refining design, addressing feedback, and optimizing CSS)
   - week 3 time: 5 hours (finalizing design and assisting with integration)
3. gmilovac: backend, working on developing user data storage and its interaction with the API as well as preconceived workout data
   - week 1 time: 10 hours (developing user data storage and initial interaction with the API)
   - week 2 time: 8 hours (implementing preconceived workout data and refining user data storage)
   - week 3 time: 5 hours (finalizing data storage and assisting with integration)
4. mwinter2: frontend, working on integrating the front and back ends of the project
   - week 1 time: 5 hours (gaining familiarity with the frontend and backend components)
   - week 2 time: 10 hours (initiating integration of the frontend and backend)
   - week 3 time: 8 hours (finalizing integration and addressing any issues)

Total estimated time: 96 hours

b) Purpose

Our project aims to simplify the indoor rowing workout process by providing personalized training plans. We conducted user interviews and examined existing solutions (like large fitness

suggestion apps), which revealed that current systems are limited in generating personalized plans for indoor rowing based on user-specific parameters. This leads to sub-optimal training outcomes for individuals of all experience levels who wish to have a plan that considers their specific goals and training level.

To address this issue, our platform uses a hidden Markov model to generate comprehensive plans that consider user parameters such as goals and training level. Users can easily customize and save their plans on the front-end, which eliminates the need for manual plan generation that is prone to errors and time-consuming. Our project adds value by providing a more efficient and accurate way to generate personalized training plans. This expands access to personalized plans for indoor rowing, making the experience more enjoyable and effective for users. Our motivation for choosing this project stems from the lack of existing solutions for personalized training plans in indoor rowing. We understand the challenges that rowers face when trying to improve their performance, which is why we believe this project can make a significant difference for us and the wider indoor rowing community.

c) Intended Audience and Intended Use

Our product targets individuals who have access to indoor rowing equipment and are interested in improving their performance through personalized training plans. We chose to focus on this group based on our personal experience as rowers and feedback from other rowers who expressed a need for plans that consider their specific parameters.

The app is designed to streamline indoor rowing training by providing users with the ability to generate or modify their training plans based on their progress. Depending on their goals and training level, users may use the app at least once a week. The app is for personal use and is not necessarily intended for professional use, although we may add features in the future that cater to professional rowers, such as performance tracking and analysis.

In addition to the target audience, gym owners, coaches, and personal trainers could in the future also use the app to generate personalized training plans for their clients or members. To accommodate these stakeholders, we could add features that allow them to manage multiple user accounts and generate plans for their clients. The app will be user-friendly and easy to navigate, ensuring that users of all backgrounds can use it.

d) Scope

While the project aims to provide personalized training plans for indoor rowing, there are certain features that we are not planning on implementing due to time and resource constraints. These features include:

1. Multi-sport integration: Some users may want to generate training plans for multiple sports but to keep the application simple we have decided to initially focus on rowing.

2. More detailed information for rowing, especially expanding into strength training with plans for weightlifting as well as on-water rowing.

3. A library of videos that can teach people how to improve their form across all the sports the site could offer.

4. Integration with fitness tracking devices or apps: While this feature can provide additional insights into the user's performance, it requires additional development and testing time. Therefore, we have decided to exclude this feature from the initial version of the app.

5. Social features: We understand that some users may want to share their progress or connect with other rowers using the app. However, implementing social features such as chat or forums can be time-consuming and requires additional resources.

6. Performance tracking and analysis: While providing detailed performance tracking and analysis can be beneficial for professional rowers, it is outside the scope of the initial version of the app.

We may consider adding these features in the future if there if we have the time towards the end of the project. However, for the initial version of the app, we will focus on providing a simpler solution for personalized training plans for indoor rowing.

e) User Stories

Backend:

1. User Story 1: As a developer, I can call on the backend API giving a shell of a schedule along with other user-specific parameters in order to receive a filled out workout plan from a sorted bank of workouts.

2. User Story 2: As a developer, I can call on the backend API to register a new user with username/password.

3. User Story 3: As a developer, I can call on the backend API to gather existing user workout plans given a logged in user.

4. User Story 4: As a developer, I can call on the backend API to load new data about a workout plan (i.e. completion status, etc.).

5. User Story 5: As a developer, I can call on the backend API to generate workout plans of varying lengths and intensity, using a hidden markov model or a random walk through a graph.

6. User Story 6: As a developer, I can use the generic hidden Markov model class to build hidden Markov models of my choosing, and read their results into classes of my choosing.

7. User Story 7: As a developer, I can use the random walking class to randomly walk through a graph of my choosing.

8. User Story 8: As an end user, I can view different types of indoor rowing workouts and how they are categorized, as well as some example graphs and models that might be used in workout plan generation.

9. User Story 9: As a developer, I can pass in a strategy class to either the hidden Markov model or the graph random walk class that will tell this model what shape its output should be and how it should be loaded (from these classes' default outputs).
10. User Story 10: As a developer, I can view documentation/README/tests to fully understand the backend code.
11. User Story 11: As a developer, I can dynamically change which user is currently accessible by API calls to the server.
12. User Story 12: As an end user, my username and password are carefully protected from security attacks on the back end.

Frontend:

1. User Story 1: As an end user, I can register with a new username and password and have my data saved for my next session on the site.
2. User Story 2: As an end user, I can get a reasonable workout plan by inputting my desired workout schedule and goals.
3. User Story 3: As an end user, I can modify a calendar-based workout plan by asking for a week to be lighter/heavier.
4. User Story 4: As an end user, I can view past workout plans in a history tab.
5. User Story 5: As an end user, I can add data for a workout session I have completed into a displayed past workout plan.
6. User Story 6: As an end user, I can try out the goal input, where I input a target workout I hope to complete and receive a workout plan that matches my schedule designed to maximize a score on this workout.
7. User Story 7: As an end user, I can opt to have more or less variation of workouts (more randomness involved).
8. User Story 8: As an end user, I find the UI and logo visually pleasing.
9. User Story 9: As a blind end user, I can navigate through the site using VoiceOver without confusion (with shortcuts).
10. User Story 10: As an developer, I can easily host the web program and understand the code via documentation, README, and tests.

Task 2 Overall Description

a) User Needs

Here are some brainstormed questions for user research:

1. What are your fitness goals, and how do you currently track your progress toward them?
2. How do you currently generate training plans for indoor rowing, and what are the limitations of the current process?
3. What parameters are essential to you when creating a personalized training plan for indoor rowing? Examples include training level, goals, time commitment, etc.
4. How frequently do you participate in indoor rowing, and how often do you modify your training plan?
5. What are the most significant challenges you face when trying to improve your indoor rowing performance?
6. How do you prefer to access and interact with digital tools related to fitness and training?

Drawing upon our comprehensive user research, we have obtained vital understanding of the requirements and desires of indoor rowing enthusiasts, empowering us to tailor our project to better accommodate their needs. By engaging with two primary user groups – seasoned rowers and newcomers to the indoor rowing scene – we have amassed invaluable data about our target audience.

Our findings indicate that those new to indoor rowing frequently encounter challenges in crafting a suitable program for themselves amidst the abundance of available advice. While the internet offers a plethora of resources on enhancing rowing skills, users have voiced discontent with the daunting task of locating a customized plan that accommodates their distinct skill level, capacity, and time limitations. For instance, beginners may lack the knowledge to structure their workouts or identify specific exercises to elevate their performance. Conversely, experienced rowers may possess greater understanding of training but still grapple with time constraints to devise and adhere to a personalized regimen.

Conversely, seasoned rowers also seek programs tailored to their unique needs. Although they might possess a deeper understanding of training and have the ability to create a program utilizing online information, the majority of available resources cater to beginners. These experienced rowers may have distinct goals or specific aspects they wish to enhance, requiring a program that acknowledges their individual needs and abilities. Our project aims to address this by offering customized training plans designed specifically for the objectives and capabilities of seasoned rowers, enabling them to optimize their training and reach their performance milestones.

b) Assumptions

Here are some assumptions for the project:

- Users will be willing to provide their personal information, such as their training level and goals, to generate a personalized training plan.

- Users will have access to an indoor rowing machine.
- Users would like more variable training plans than they already have.
- Users generally care about indoor rowing.
- We will be able to learn a new login/database system.
- There are ways to display a schedule well with React.

These assumptions are based on a combination of research and personal experience and are primarily geared towards individuals who participate in rowing and wish to improve their ability on the indoor rower, as well as those who prefer indoor rowing over other forms of exercise for general health. It is assumed these users also seek personalized training plans to improve their performance.

Our user research indicated above that we will have users that generally care about indoor rowing, and also would like more variable training plans. On the other hand, we hope that we will be able to live up to our own technical assumptions; these are outside of the scope of our users.

These assumptions really center those who value indoor rowing, as our users will likely all have this attribute. In this way, this user group might be overly centered.

In addition, our second assumption in particular may cause harm to those who may not have access to the equipment relevant to our workout plans. By providing a more accessible plan in future iterations of the project, we might mitigate the negative impact of this assumption.

c) Dependencies

The project relies on several software and technology components, including React TSX for the front-end, Java for the back-end, and a hidden Markov model to generate personalized training plans. We will also be using databases (maybe SQL) to store user information and training plans, as well as some authentication software to encrypt our data. Additionally, the project will require a stable internet connection and a compatible web browser to access the app.

In terms of non-technical dependencies, the project relies on user input to generate personalized training plans. Therefore, user cooperation and willingness to provide accurate information is crucial for the app to function properly. The app will collect user data such as name and email for account creation and subscription purposes.

The relevance of the app may rely on a social and cultural context, as the popularity and accessibility of indoor rowing may vary depending on the region and cultural norms. Therefore, we will need to consider the social and cultural context of our target audience when designing and marketing the app. There are no significant financial dependencies for the project, as all software we need is available under an educational license, such as JetBrains.

Task 3 System Features and Requirements

a) Module Design

We hope to collect and store user data such as: usernames, passwords, workout preferences (mostly discrete or bounded continuous variables, nothing too personal), schedule, and workout data (if they choose to input it). In order to store our data, we might use a database library like SQL – however, we are open to suggestions from our mentor and aren't super experienced with this. In addition, our user authentication might interact with google authentication for safer transfer of user data. At a high level, or project will collect this user data outlined above, and use our models on the backend to give users personalized workout plans.

The User Database module will exist on the backend, and store all of this user data in a database. This database will interface with other backend modules by having a query function and a modify function of some kind; these will allow the database to both be queried and altered based on API calls to the models/login from the front end. This module could be considered as part of the Controller, as the user will have some interaction with it. It will also have API calls register and login, which take in an encrypted username/password.

The Models module will contain both the hidden Markov model and graph as described in previous sections. It will use user data (via database interface described above) to generate workout plans, and return these workout plans to the front end via a serialized JSON object that the front end will need to parse. The main Markov model API call will likely take in a schedule parameter of some kind, as well as user preferences like light/heavy and a goal workout. These parameters will inform the model on how to generate a workout plan, which it will serialize and return to the front end. The object to be serialized will likely be a List¡Week¿ object, where each week will contain a Day, and that Day will contain a set of ordered workouts. We might also have a second Markov model API call, where instead of the days of the week being hidden states, we will use workout categories as hidden states. This call might take in a schedule, as well as user preferences on variety (as a percentage, will determine probabilities of unconventional state transitions) and workout categories. In addition, the graph API call will also take in a schedule parameter, but otherwise run fairly independent. All of these API calls will return a serialized List¡Week¿ of some kind. For fairly obvious reasons, this is part of the Model – the end-user cannot interact with it. We could see an argument for this module being split in two – graph and Markov model.

The Workout Data module will contain all of our preconceived workouts/sensible workout combinations in the form of JSON, to be easily serialized into Java objects. This module is part of the Model, and will primarily have interaction with the Models module via a class that will have a method that gets a random workout of a particular type.

The Login module will contain all pages for logging in users on the front end, as well as registering them. This module is primarily the Controller, but is also part of the view. This module will need to rely on the API parameter interfaces of the User Database module.

The Homepage module will contain a page that allows users to generate workout plans based on their preferences. This module will need to rely heavily on the API parameter interface described in the models module. This will primarily be part of the View/Controller.

The History module will allow users to look back at their past workout plans and input their own data, which will interact with the User Database module (API call containing updated JSON for a

given workout plan, with user modifications). This will primarily be part of the View/Controller.

All of the History, Homepage, and Login modules might be split into two – one component handling the View, and the other handling the Controller (API calls). Each module would contain sub-modules, one containing relevant React components and the other containing relevant API-calling and mock functions.

b) Data Requirements

We need to collect the following data:

- Username/Password - we need this to log users in, and save their workout plans. Our software might still work without this; users just would not be able to see past plans. We will ask for consent when registering users, and tell them not to use identifying information.
- Schedule - we need this to make workout plans for users that are sensible for them. Without this, we could use a default schedule, but it would severely reduce how helpful our app is. We will warn users before inputting their data that we will collect their schedule, but that it will only be tied with their username and password, which should not contain identifying information.
- Past workout plans - we need this data so that users can have access to their past interaction with the app. Without saving the data, users might forget workout plans, but the software would still work in some way. We will warn users before making a plan that it will be saved to their account.
- Voluntary user data about completed workouts - without this data, our software would still work; however, users might find it nice to see their data for each workout when looking in their plan history. Users will be warned that their workout data will be attached to their username.

None of this data is publicly available, and all of it will be used directly in our front end display! So, we really would like to have all of this data collected – none of it is too sensitive, so this should be feasible.

Each of the frontend submodules that call on the backend APIs will be using user data in some way – we will make this secure by (hopefully) encryption pre-API call. From here, the User Database module will store user data, using a package that will store the data securely (we believe SQL is secure or something like it, but would like to consult with our mentor first). Finally, the Models module processes data, and will encrypt before sending the data back to the frontend.

c) Risks

There are a number of risks related to the data we store, but these two are the only ones we feel are truly relevant to our users:

1. Leakage of user logins (usernames/passwords reveal identifying information). We can mitigate this risk by encrypting our data during API calls and trying not to ask for any personal data.

2. Leakage of user schedules/workout data. Professional athletes using our site may have their workout speeds leaked, which could change the outlook of their competitors in some way our site does not intend. We can mitigate this, like in our first risk, by encrypting their workout data.

Are there accessibility risks involved in the kind of presentation or interaction you will provide? E.g., are you concerned about repetitive stress caused by your UI design, or about the way your page works making it difficult for users to employ screen readers or magnifiers? How can you adjust the design of your View to account for this accessibility need and others?

There are a number of accessibility risks involved in our presentation. First, if a user produces a large workout plan, they might have trouble reading through all of it at once. In order to fix this issue, we might try to find a way for users to navigate through the header of each workout to get a quick sense of their new schedule. We also worry that navigating our inputs well to create a plan might be tricky for blind users – we will try to designate a very specific div that takes in all user input. Each option for data entry will be fairly uniform.

In addition to accessibility and security risks, our site has two main social risks. First, it might run into a lot of competition with other companies that already have significant infrastructure to make models like ours for many sports (however, we believe our indoor rowing knowledge will outdo this competition). In addition, our project also risks encouraging users to do more strenuous or complicated workout plans than really makes sense for them. Those hoping to making rowing less of an arms race (as it seems to be these days) might oppose a project like this.

d) Testing Plan

User Database Module:

In this module, we will primarily test the following properties:

1. User registration: Ensure that new users can be registered and added to the database with unique usernames, valid passwords, and other required information. In particular, we will test that some "register" function and API call will be able to add a new user to the database. This will be done with unit testing and integration testing, both with mocks of users attempting to register with their usernames and passwords. We will need to watch out for cases of invalid/collision usernames, as well as invalid passwords.

2. User login: Test that users can log in with their registered credentials and that invalid credentials are rejected. We will test that the "login" function of the database wrapper can log in a user, as well as integration test the "login" API endpoint. This will also require mocking of attempts to login, particularly for the integration testing. We will need to watch out for cases of incorrect passwords, non-existent usernames, etc. (appropriate API/function response).

3. Data retrieval: Verify that user-specific data can be correctly retrieved from the database. We will unit test that a "query" function of the database wrapper works, and that the API calls for different data all work as expected. We will need to watch for the cases where no user is logged in, and cases where the data does not yet exist.

4. Data modification: Check that user data can be updated accurately, including workout preferences, schedule, and workout data. We will unit test that the "modify" function of the database wrapper works, as well as integration test each data modification endpoint. We will need to watch for the cases where users are not logged in, or when the data field doesn't necessarily exist or is not of the proper shape.

5. Data security: Assess the security of the stored data, such as password encryption and secure data transfer using mock authentication services. We will do this with integration testing, although we are not exactly sure how to test whether the encryption works correctly.

Models Module:
For this module, we will focus on testing the following aspects:

1. Markov model output: Validate the generated workout plans based on user input and preferences, ensuring that the plans are consistent and follow the expected structure. We will both unit test that each way of generating a workout plan using the Markov model works as expected, as well as unit test the base Markov model class. We will also integration test the endpoint that returns the results of a model run, and ensure that these results can be easily deserialized. Finally, we will need to watch out for cases where there is an invalid schedule inputted, or when certain defaults need to be used (parameters left out or invalid).

2. Graph model output: Confirm that the graph-based workout plans meet user requirements and maintain a logical progression. We will test this in the exact same way as the Markov model.

3. Serialization and deserialization: Test the correct conversion of data between serialized JSON objects and Java objects. We already have most of the infrastructure for this from our collective Maps work – just unit tests that we can easily read JSON into objects.

4. Model interaction: Ensure that the models interact seamlessly with the User Database and Workout Data modules. Here, we will unit test calling on an instance of the Markov model / graph classes (at a higher level) that have access to the database. This will require mocking of the database responses. We will need to watch out for failed database queries.

Workout Data Module:
In this module, the primary testing points will be:

1. Preconceived workouts: Test the accuracy and consistency of preconceived workouts and workout combinations. This will be a user based test; we will have different users try our site in development and see what they think of the plans.

2. Data serialization: Verify that the workout data can be easily serialized into JSON and deserialized into Java objects. This will mostly take the form of unit testing our classes that model a "workout".

3. Interaction with Models module: Ensure that the module can properly provide workout data to the Models module as needed. This will take the form of unit testing; we will use some of our Models classes and mock the response of this module, as well as vice versa.

Login, Homepage, and History Modules:
For these modules, we will test the following components:

1. User interface: Assess the usability and user experience of the Login, Homepage, and History pages. This will mostly take the form of user interviews, as well as React tests that use accessibility labels.

2. API calls: Test the interaction between the frontend and backend through API calls, ensuring proper data transfer and processing. This will primarily require mocking of backend API responses; we must ensure to test a number of failed API calls to see how the page responds.

3. User input validation: Confirm that user input is validated before being sent to the backend. This will take the form of unit testing; we will need to try many invalid user inputs.

4. Error handling: Check that the modules can handle errors gracefully and provide helpful feedback to users. This will take the form of more React tests under errored states of the web page that we will attempt to induce with bad mock data from the API responses.

Integration Testing:
After testing individual modules, we will perform full-fledged integration testing to ensure that all modules work together seamlessly:

1. Data flow: Test the data flow between frontend and backend modules, as well as between different backend modules.

2. User experience: Verify that the end-to-end user experience is smooth and meets the project requirements.

3. Error handling: Confirm that errors at any stage of the process are handled gracefully without causing disruptions to other modules.

These are all mostly outlined in the above tests – we're not sure if we need this end-to-end testing, and are not totally sure how to automate it. We would greatly appreciate any input!