**System Architecture**

Our architecture is designed to support mobile clients, with our model backed entirely by a backend web service hosted in the cloud. All data and other model information are requested by the mobile application in real time; our architecture is designed around this principle.

***Database Schema***

Our system needs to store information about users, workouts, workout templates, and conversations between users. We are storing all of this information in a PostgreSQL database.

The following is a rough database schema using a relational database system. It is not final, but is expected to be very close to what we finally use.

**TABLE User (**

**name: text**

**user id: int primary key**

**)**

**TABLE Trainer INHERITS User (**

**specialty: text**

**)**

**TABLE Trainee INHERITS User (**

**goals: text**

**weight (lb): int**

**height (inches) : int**

**)**

**TABLE Connections (**

**userid: int**

**connection id: int**

**)**

**TABLE Devices (**

**userid: int**

**deviceid: int (or text, depending on how device id might be formatted)**

**)**

**TABLE trainerTraineeRelation (**

**trainerId (user id): int**

**traineeId (user id): int**

**)**

**TABLE traineeWorkout (**

**trainee id (user id): int**

**workout id: int**

**)**

**TABLE msg (**

**msgid: int**

**author (user id): int**

**time: time**

**message text: text**

**)**

**TABLE conversation (**

**conversation id: int**

**member1 id (user id): int**

**member2 id (user id): int**

**)**

**TABLE msgToConversation (**

**message id: int**

**conversation id: int**

**)**

**TABLE workoutTemplate (**

**workout template id: int primary key**

**title: text**

**author (user id): int**

**)**

**TABLE workout (**

**workout id: int primary key**

**title: text**

**time: time**

**)**

**TABLE exerciseMetaData (**

**exercise metadata id: int primary key**

**name: text**

**instructions: text**

**photolink: text**

**category (used to distinguish what type of exercise): int**

**quantityType1: real**

**quantityType2: real**

**quantityType3: real**

**)**

**TABLE exercise (**

**exercise id: int primary key**

**exercise template id: int**

**Progress: int**

**)**

**TABLE exerciseTemplate (**

**exercise template id: int primary key**

**exercise metadata id: int**

**category (used to distinguish what type of exercise): int**

**defaultquanity1: real**

**defaultquanity2: real**

**defaultquanity3: real**

**)**

**TABLE templateRelation (**

**workout template id: int**

**exercise template id: int**

**order: int**

**)**

**TABLE workoutExerciseRelation (**

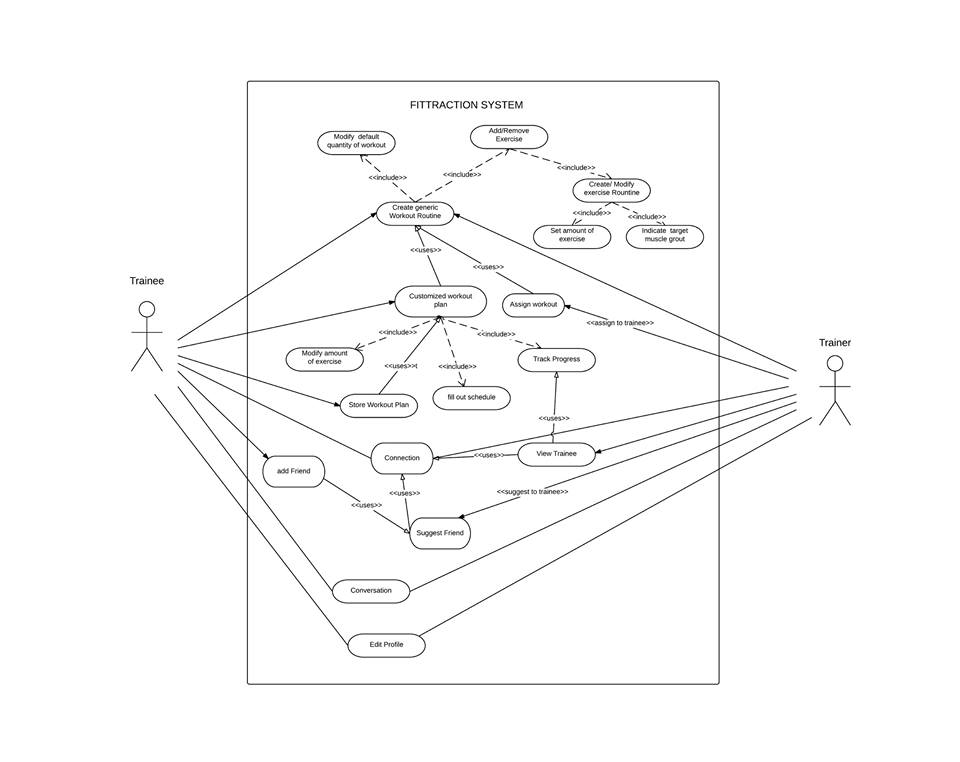
**workout id: int**

**exercise id: int**

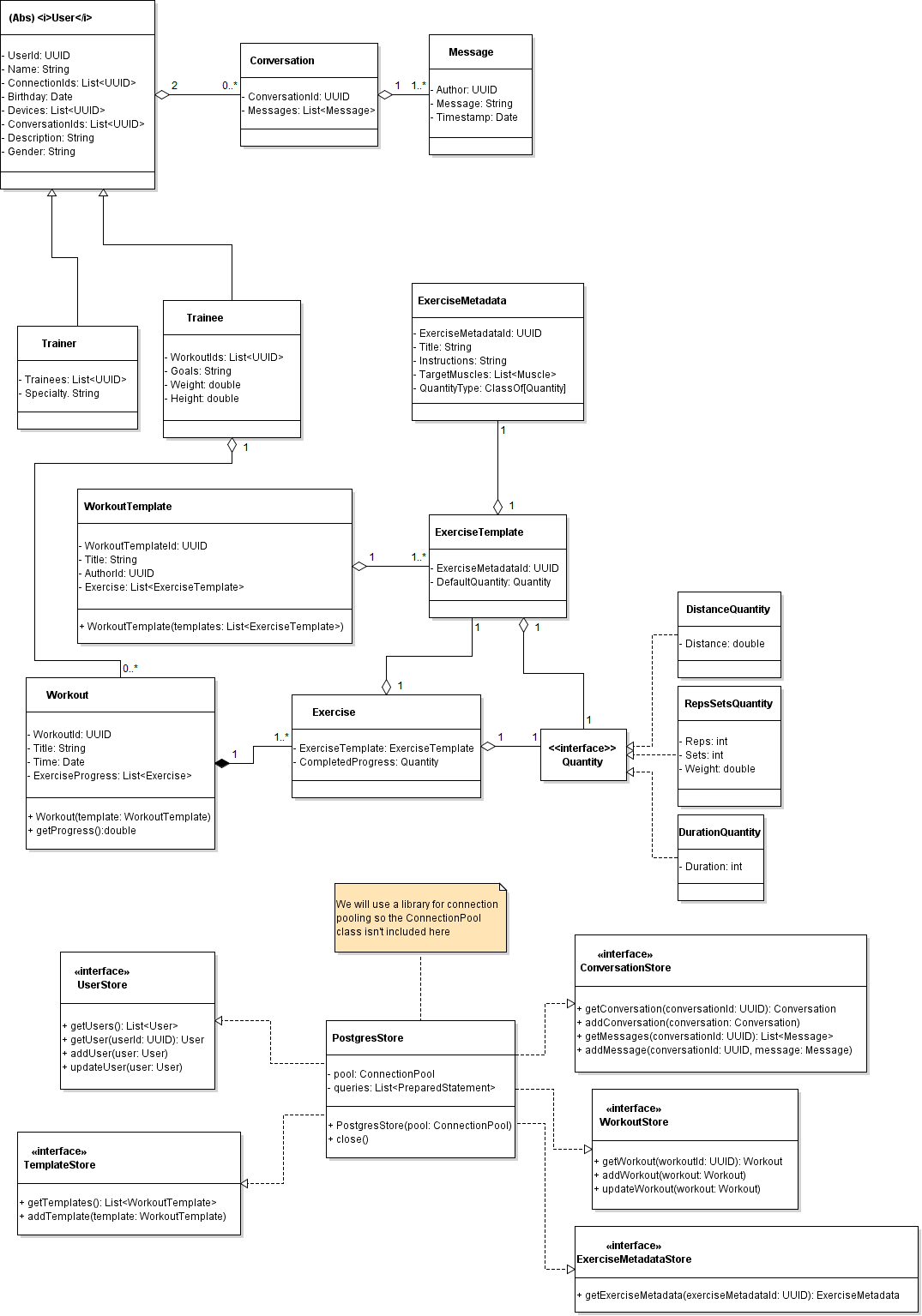
**)**

***Diagrams***

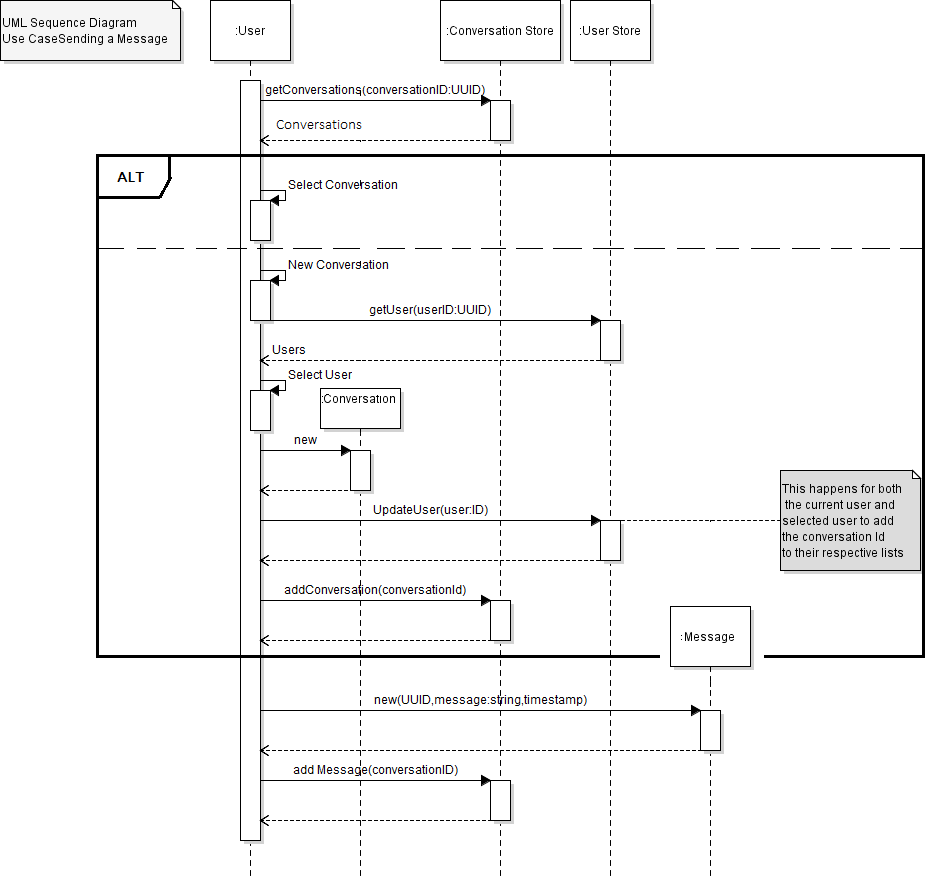
Use Case Diagram



Class Diagram



Sequence Diagrams



This Sequence Diagram describes the use case of sending a message to another user and the process of selecting that user. Accompanying it is pseudo code for the sequence.

SEND MESSAGE:

CALL get Conversations list

DISPLAY list of Conversations

READ user input

IF (INPUT select conversation Conversation)

READ user input for conversation

ELSE //user selected new conversation

CALL get Users //Get users's contacts

DISPLAY Users

READ user input for other User

NEW Conversation

CALL Update this User // store the new identifier to conversation object

CALL Update this User // store the new identifier to conversation object

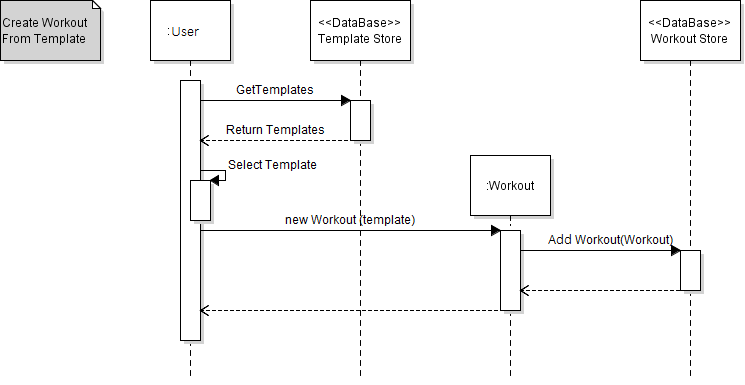
CALL add Conversation // store the conversation object

DISPLAY for Text input

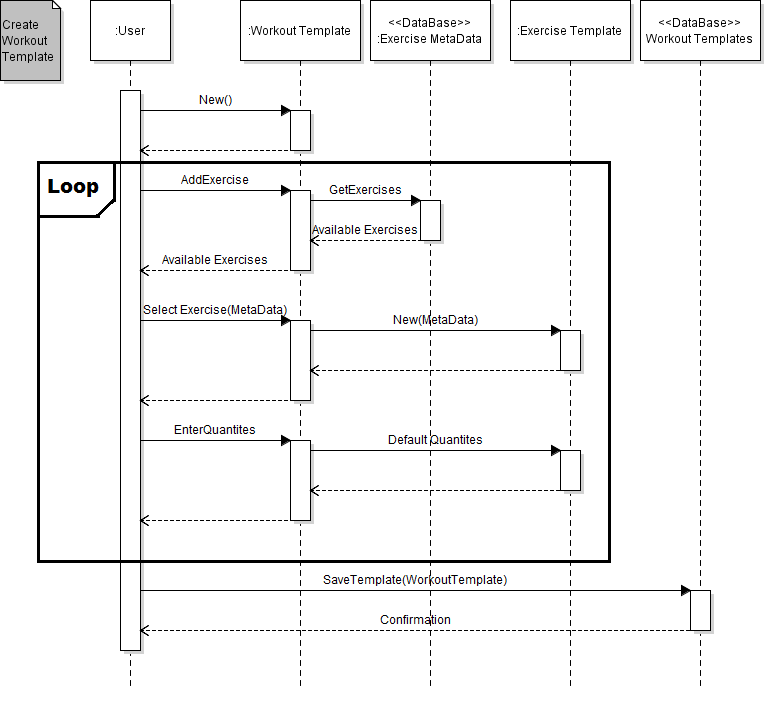
READ user input text

CALL New Message (User,Text,Time) // create new message item

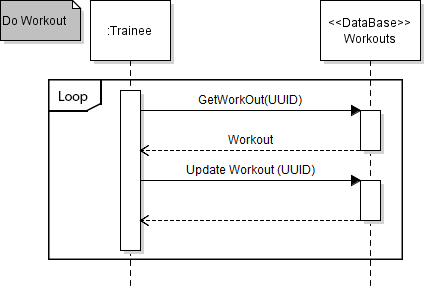
CALL addMessage // add the message to the list of stored conversations



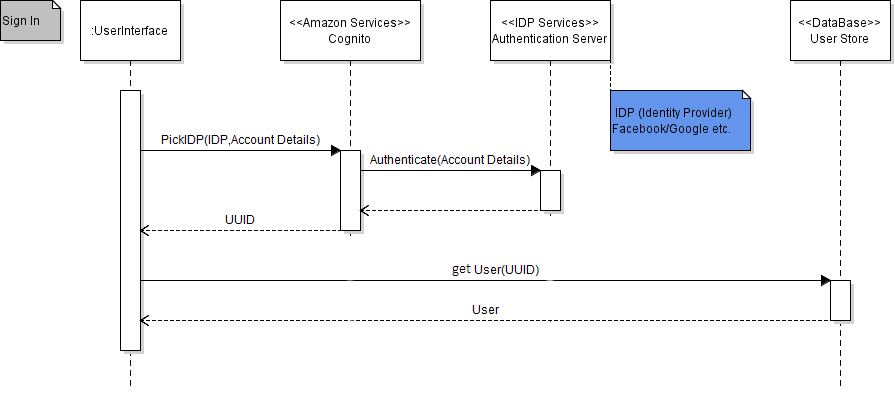
This Sequence Diagram describes the use case of creating a workout from a template workout.



This Sequence Diagram describes the use case of creating a workout template, setting the default quantities (reps/sets/distance) per exercise and adding it to the database.



This Sequence Diagram describes the use case of the user doing a workout/s.



This Sequence Diagram describes the use case of a user signing into the application.

***Design Decisions***

During the course of the design exercise, we encountered a few design decisions with different potential solutions; one such example was whether to store personal workout data on the mobile device only or whether to persist it to the backend. While storing it locally would be faster and reduce network bandwidth, sharing the information between people is much harder and there is a less-clear single source of truth. As such, we chose to persist it to the backend.

Another decision was whether or not to allow templates to be parameterized (ie, a template can support multiple values for fields such as sets and reps) or whether to create multiple templates for each instance. Creating more templates is easier to do and simpler to store; however, we decided that it was critical to be able to create a single workout template with variable fields for people of different athletic level. Because of this, we went with the latter.

**Process**

***Risk Assessment***

Tight Schedule

* + **Likelihood (High):** It is evident that we don’t have a lot of time to complete the product, that paired with lack of experience will lead to potential deadline issues.
  + **Impact (Medium):** We believe that we would still be able to handle all of the base functionality within the timeframe given, If we focus on completing our two most important features first, it is less likely that we will have a nonfunctioning deliverable by our deadline.
  + **Evidence:** This is based off of the expected amount of time the team can work on the project in addition to the need to address our inexperience and other obligations to complete the project within 5 weeks.
  + **Steps To Reduce Likelihood:** To reduce the likelihood and impact of this risk, we will first focus on completing at least our two most important features, to avoid having multiple incomplete features, and maintain a flexible schedule to address problems as they arise.
  + **Plan For Detecting The Problem:** To detect the problem we will perform frequent progress checks and stand-up meetings to reevaluate where the project is, and modify our schedule to accommodate changes necessitated by new information,
  + **Mitigation Plan:** To mitigate the effect of a time crunch, we will schedule more time for work as we near the end of each sprint, and make whatever time sacrifices necessary (eating, sleeping) in order to meet our deadlines at the end.

Unfamiliar Areas of the Product (Scala, Titanium, Docker, AWS)

* + **Likelihood (High):** Our team has little experience in mobile application design so we picked certain software due to recommendations rather than experience.
  + **Impact (High):** By being unfamiliar to with software tools, we spend more time researching and understanding the software tools than necessary which will shorten our coding time available.
  + **Evidence:** Each of our team members have presented the amount experience that we have in our software development career. Though all of us are proficient in coding, only a couple have a little experience in mobile applications.
  + **Steps To Reduce Likelihood:** Each team member are responsible for researching the software on their own time and ask questions to other team members if needed. We aim to continue to help each other in understanding the software tools if a team member is proficient in a tool.
  + **Plan For Detecting The Problem:** We will establish clear communication between all team members. During our weekly meetings we will discuss our progress and what’s blocking us. By doing so, the whole team will know each individual's progress and problems.
  + **Mitigation Plan:** If unfamiliarity occurs, the team members will need to take time off from implementation to solve the issue or ask other team members who understands the issue. If the issue involves every team member, we will focus on a meeting that will resolve these issues.

Unintuitive UI

* + **Likelihood (Medium):** Our team has limited design experience so there is a possibility that we are not able to combine our features into an intuitive UI.
  + **Impact (Medium):** It is very important to have a simple UI that incorporates all of our features that caters to the customer however we decided that for the scope of this class our first priority is functionality, making this a medium impact item.
  + **Evidence:** This is based off a survey of the team asking for design experience and an ordering of priorities to establish what we would consider a successful project.Of which, the team reported having close to zero previous experience with a focus on UI.
  + **Steps To Reduce Likelihood:** We are planning on having a meeting dedicated completely to establishing color schemes and a common UI layout theme. We will review pushed code week to week to assess adherence to the theme.
  + **Plan For Detecting The Problem:** We plan to detect this problem by using hallway testing to test for usability and appearance.
  + **Mitigation Plan:** We will change our UI theme if it does appear too bloated to users. In changes to UI theme we will always air towards simplicity.

Distributed Backend Architecture (AWS) Setup Difficulties

* + **Likelihood (Medium):** We are using various parts of AWS for very specific parts of our application (general computing, authentication, text messaging, etc). Each part has a learning curve and separate configuration that will take time to understand its usages.
  + **Impact (Medium):** The various AWS components are highly correlated with our individual features; for instance SNS is only required for notifications. As such, trimming features should directly trim the number of services we need to understand.
  + **Evidence:** Simply browsing through all of the services offered in AWS and the tools that we decided to use, we can see that there are a lot of different technologies that will be pertinent to our success. However after some basic experimentation and browsing Amazon’s documentation, all of these services are well documented (with documentation geared towards unexperienced groups) which include examples and demos; Amazon has high-quality support for teams in our situation, which is why we don’t feel the risk is high.
  + **Steps To Reduce Likelihood:** To reduce the likelihood of the number of services being a problem, we are going to frontload implementing the more core, application-wide services (EC2, RDS, Cognito) and keep the more specific services (SNS, S3) farther back in the priority chain.
  + **Plan For Detecting The Problem:** We will detect whether if problems with our backend technology will occur based on whether we are meeting our sprint goals. If standing up these services is clearly harder than we originally estimate, we will know within the sprint cycle that the issue occurs and start our mitigation plan.
  + **Mitigation Plan:** The mitigation plan if the number of services is overwhelming is simply to trim features such that the required number of services decreases. There isn’t really a way around this; the services were selected in order to support our feature set, so to remove services we must also remove features.

Frontend: Titanium Cross Platform Complexities

* + **Likelihood (High):** Titanium is a hybrid mobile app Javascript framework that generates native mobile platform code that can be used in Android, iOS, Windows, and other mobile platform. The problem is that each platform will have specific functionalities exclusive to them that Titanium can’t cover that will require the team to take extra care to include. For example: Android has a back button, but iOS doesn’t.
  + **Impact (Medium):** For the most part the our apps will share the same core logic flow. The majority of the code we write will be the same. The biggest impact will be tackling specific details that are different between mobile platforms.
  + **Evidence**: Titanium’s motto is “write once, adapt everywhere”, which means that differences in mobile platform is already expected to occur.
  + **Steps to reduce likelihood:** The problem of generating native code is inherent in all cross platform frameworks the only two ways to avoid the problem is to 1) write app natively for each platform, which would defeat the purpose of using Titanium or 2) select specific features for the app that won’t have any crossover differences, which would restrict what could be created. Unfortunately there is little we can do to reduce the likelihood of this problem occurring.
  + **Plan for detecting the problem:** Many of the specific API calls in Titanium will mention specific feature capability of a mobile platform. By doing so, we will know what features we have to consider for different platforms. Otherwise we’ll compile a list of general differences between platforms and use our own intuition/experience during our group meetings to discuss differences in mobile platforms.
  + **Mitigation Plan:** Because the problem isn’t new. We will see what other people have done in the past to tackle any problems that had native differences.Worst case scenario we write our code in Titanium so that it would be functional but not look native. ex: add a back button in the UI for iOS but also include it in for Android.

The risks selected have changed in order to be more specific, the general risk of having 5 sprints has evolved to address the risk of running out of time, lack of familiarity with tools has become more specific to address the risks in using AWS and in using Titanium, which blends with the previously identified high complexity risk. In addition to these, we’ve identified the risk of having an unintuitive or suboptimal UI.

***Project Schedule***

**External Milestone** Internal Milestone

\*Workout feature represents two key features of creating and sharing/tracking

|  |  |  |
| --- | --- | --- |
| **Milestone** | **When** | **Tasks Due (Further divided Below)**   * **Dependencies** |
| Weekly Reports | Tuesday, weekly | Write up report from meeting |
| Hello Worlds | Evening 4/25 | Set up your Tools and Environments and get a initial commit on the repo |
| Sprint 1 Start | 2:30 pm, April 27, 2015 | Zero Feature completed   * Hello Worlds |
| Zero Feature DUE | 11:00 pm, May 1, 2015 | Turn in Zero Feature   * Zero Feature Complete |
| Finish User Profiles | 2:30 pm, May 4, 2015 | Sign up/in feature  Profile Pages   * Zero Feature |
| Finish Workouts Feature | 2:30 pm, May 11, 2015 | Create Workout Templates  Create Workout  Execute Workout  Finish Buildable Beta |
| Beta Release DUE | 11:00 pm, May 15, 2015 | Turn in Beta   * Finish Workouts Feature |
| Finish Messaging Feature | 2:30 pm, May 18, 2015 | Implement Message store   * Backend environment setup   Finish Feature Complete   * Workouts * Profiles * Messaging |
| Feature Release DUE | 11:00 pm, May 22, 2015 | Turn in Feature Complete   * Finish Feature Complete |
| Finish Release Candidate | 2:30 pm, May 25, 2015 | Fix Bugs in Feature Release   * Turn in Feature Complete   Make Release Candidate Buildable |
| Release Candidate DUE | 11:00 pm, May 25, 2015 | Turn in Release Candidate   * Turn in feature complete |
| Project Due | 11:00 pm, June 5, 2015 | Fix Bug in Feature Release |

**Tasks To Completing Features**

|  |  |  |
| --- | --- | --- |
| **(Feature) Task** | **Effort** | **Dependencies** |
| Zero Feature Shipped | Whole Team 1 work week | People getting familiar with their respective tool set (Hello World) |
| (Profiles) Get Notion of Users in Database | Backend team (4 students) 1 week | Zero feature shipped |
| (Profiles)Notion of User in Front end | 1 student work week | Zero feature shipped |
| (Profiles)UI pages for Sign in/up sequence | 2 student work weeks | Zero feature shipped |
| (Profiles) Set up UI for landing/profile pages | 2 student work weeks | Zero feature shipped |
| Test Profile Feature | 1 student work week | (Profile) tasks |
| Ship Profile Feature | 1 student work week | Test profile feature |
| (Workouts) Get notion Workouts/Exercise Templates in the DB | 2 student work weeks | Profiles |
| (Workouts) Get Notion of Workout in DB | 1 student work week | Profiles  Workout Template |
| (Workouts) Get Notion of Exercise Meta Data in DB | 1 student work week | Profiles  Exercise Template |
| (Workouts) Get Notion of Exercise in DB | 1 student work week | Profiles  Exercise Template  Exercise Meta Data |
| (Workouts) Get Notion of ExerciseMeta in DB | 1 student work week | Profiles |
| (Workouts)Implement UI for Workout Creation | 2 Student Work Weeks | Zero Feature Release |
| (Workouts)Implement Workout Conduction | 2 Student Work Weeks | Zero Feature Release |
| Test Workout Feature | 1 Student Work Week | Workout Tasks |
| Ship Workout Feature | 1 student Work Week | Test Workout Feature |
| Specification/Use Case Testing for Beta | 1 Student Work Week | Ship Profile Feature  Ship Workout Feature |
| Finalize Beta | 1 Student Work Week | Profiles  Workouts |
| (Messaging) Implement Messaging in Backend | Backend team 1 week | Profiles/Users |
| (Messaging) Implement Messaging in Frontend | Frontend team 1 week | Profiles/Users |
| Test Messaging Feature | 1 Student work week | Messaging tasks |
| Ship Messaging Feature | 1 student work week | Test Messaging Feature |
| Specification/Use Case Testing for Feature Release | 1 Student Work Week | Finalize Beta  Ship Messaging Feature |
| Ship Feature Release | 1 Student work week | Profiles  Workouts  Messaging |
| Ship Release Candidate | 1 Student work week | Ship Feature Release |

**Why did we schedule our workflow this way:** We thought it was pivotal to implement the key functionality of the workouts first, hence we also needed the notion of user profiles. This way we will be able to have a working prototype early in the development process so that we can get more user’s hands on the product in a more timely matter.

***Team Structure***

Two Sub-Teams: Mobile + Backend

* There are two main components to our project; thus it’s logical to split ourselves into two teams. The first team will work on the user-facing mobile application that provides an entry point into our use cases (schedule and record workout, message trainer, etc). The second team will work on the backend that will support the mobile application (ex. store the workout schedules and other user information, record metrics, etc).
* Each team has a team lead, whose responsibility is to coordinate with the PM and their teammates to ensure development is going smoothly; it is their responsibility to ensure developers are happy and not blocked, as well as attempt to load balance tasks evenly and keep communication open with their team members.

Everyone Develops and Test

* We’ve decided that if we were to designate someone to a single role, no one would have the full experience of building a software from the ground up. This forces developers to be responsible for their own code and thus think more about tricky cases. Throughout the next 5 weeks, after each Sprint team members excluding the leads, will have the opportunity to work on a different team. If a team member prefers to stay on the same team throughout the project, that is also fine.

#### The Team

* Project Manager (PM) - Ian Turner
* Mobile Lead - Joseph Kesting
* Backend Lead - Andrew Burnell
* Mobile Team - Joseph Kesting, Melissa Chan, Justin Lee, Ruokun An
* Backend Team - Andrew Burnell, Ian Turner, Maria Treacy, Joshua Chang

Team Member Roles (See Project Schedule for more Information)

External Milestones

* Everyone:
  + 5/1 - Zero Feature Release
  + 5/15 - Beta Release
  + 5/22 - Feature Release
  + 5/29 - Release Candidate

Internal Milestones

* Everyone:
  + 4/25 - Hello World (Frontend + Backend)
  + 4/28 - Zero Feature Release
  + 5/4 - Sprint 1 Ends
  + 5/11 - Sprint 2 Ends
  + 5/18 - Sprint 3 Ends
  + 5/25 - Sprint 4 Ends
* Mobile Team:
  + 5/4 - Profiles (Sign up, Profile Page): User Interface
  + 5/11 - Workouts(Create Templates, Create Workout, Execute Workout): User Interface
  + 5/18 - Messaging: User Interface
* Backend Team:
  + 5/4 - Profiles (Sign up, Profile Page): Notion of Users
  + 5/11 - Workout (Create Templates, Create Workout, Execute Workout): Store Data
  + 5/18 - Messaging (MessageStore): Store Data

For each internal sprint, we aim to have each feature done. We aim to have our

internal milestones due to the possibility that each feature will take longer than expected. By attempting to finish external milestones a few days before the due date, we leave some time for cases where implementation may take longer than expected.

Forms of Communication

The FitTraction team has set up a weekly meeting every Monday starting at

2:30pm. The weekly meeting have been set to last 1 - 3 hours as this is the available time that all of the FitTraction team can meet. If necessary, each backend and mobile team will set up their own meeting times the weekly Monday meetings are not enough. The FitTraction team will also be meeting with our customer(Meg) every Tuesday from 11:30 - 11:50am in order to keep our customer updated on our current status.

During the weekly meeting, the FitTraction team will focus on recap of the previous agenda and determining the current agenda. After the meeting, it would then be recapped in our weekly status report. All of FitTraction’s weekly status reports and documents are located in the FitTraction Google Drive.

Our other forms of communication includes:

* FitTraction Facebook Group Page - Used as our primarily source of online communication.
* FitTraction Google Group Mailing List - Used as a form to contact our customer (CSE 403 Staff) if needed.
* FitTraction GitHub - Contains all codes and wikis needed.

Disagreements

If disagreements arise that cannot be solved by having an open-minded conversation about the issue at hand, other people can be involved. There will not be a strict policy for this scenario, since the correct approach will vary largely based on the context of the problem. Good resources for resolving disagreements if they really cannot be solved by discussion are the PM and Leads; if a decision has to be made, they are ultimately responsible for making it.

***Test Plan***

Build Server

* + We will build all of our code on a Jenkins server before deploying. All artifacts that end up running in production will be produced by Jenkins.
  + We will schedule a build immediately after every commit to master.
  + The following build steps are what we forsee (subject to tweaks but not large change):
    - Code checkout
    - Build (using Titanium/sbt depending on frontend/backend)
    - Test (both unit and system tests)
    - Package (into final artifacts such as fat JARs, docker images, apks, etc)
    - Deploy (if necessary)

Unit Test Strategy

* + Unit tests will test the functional logic in the code, which is where we expect to catch a majority of bugs. These tests should essentially test a single functional unit, and prove that each class works individually. These test will break down the code into its smallest pieces and ensure expected behavior, and will use mocked data and services to preserve test data and ensure that the data being used is not corrupt or unavailable during testing.
  + These tests should be run most frequently, as they should be extremely fast, and would be run every time something is modified, preferably on each save.
  + Every developer will create their unit tests when they implement a particular piece of functionality
  + For the frontend, since we are developing with Appcelerator Titanium, we will use Tio2 which is a Titanium utility for better automated unit and functional testing of Titanium APIs and Titanium Apps. This will be run from the command line, supports testing for iOS and Android, and is based on the Mocha JavaScript test framework. <https://github.com/appcelerator/tio2> <http://mochajs.org/>
  + For the backend, will use Scalatest for unit and integration testing. These tests are run using sbt and provide coverage reports in standard formats.
  + Unit tests will be required as part of every build.

System Test Strategy

* + Using the same tools described above, we will run through scenarios as functional integration tests. These will test the application interaction and UI modifications, and as such will be much slower. These tests will wire related classes together as they would in production and exercise some basic execution paths. When a functional test fails, with no unit tests failing, debugging will be necessary and then a unit test must be added retroactively to understand the bug and prevent it from happening again.
  + These tests will be developed by determining the likely scenarios of use, focusing on the more functional aspects and not every minor edge case, as these tests are much more expensive in time and labor.
  + System and integration tests will be run less frequently than the unit tests, but will be run as larger classes are being developed, and when we believe some functionality has been completed (eg. testing that a message can be sent from one user to another).
  + Integration tests will be run before any code changes are committed, and as part of every build.

Usability Test Strategy (Frontend Only)

* + These tests will focus on ensuring GUI elements are designed clearly, and that user interaction is made as simple and intuitive as possible. This will test that a new user can easily run through the various functionalities of the application.
  + These tests will include heuristic evaluations by the team and other peers, paper-prototypes to understand early on how potential users expect to use the application, and putting it in the hands of users and both observing their use and asking for feedback. This will mean putting the workout creators and trackers into the hands of gym-goers as soon as these features are completed to obtain feedback as quickly as possible.
  + Usability tests will be done between 3-5 users per round (Jakob Nielsen describes that five users is enough) first when designing initial GUI elements and usage patterns, and again as major features are completed.

Bug Tracking Mechanism

* + We will track bugs using Github Issues
  + Bugs will be submitted immediately upon discovery, along with details of evidence of the bug (stack traces, test failures, etc) and repro steps (if applicable)
  + Bugs will be evaluated as sprint tasks and prioritized alongside features by the PM

We believe that having a comprehensive unit test suite and integration tests to confirm major functionality completeness will be adequate for testing our application and meeting work estimates.

***Documentation Plan***

In general, most of the features of FitTraction will be intuitive for the user, but we will include help texts through the UI where it is needed. We will also incorporate a Youtube demo video as a visual option for the user to understand how to use FitTraction.

We will also provide a short admin guide to gym owners that describes the features of the application and how to best use it. There will be separate sections detailing the functionality of gym owner accounts and Trainer accounts.

For developers, we will provide documentation using conventional language documentation techniques for Scala and JavaScript, as well as API documentation for our REST API via Swagger.

***Coding Style Guidelines***

Pre-Existing Coding Style Guidelines

* Scala: <http://docs.scala-lang.org/style/>
* Javascript: <https://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml>
* XML: <http://google-styleguide.googlecode.com/svn/trunk/xmlstyle.html>
* CSS: <https://google-styleguide.googlecode.com/svn/trunk/htmlcssguide.xml>
* In addition to adhering to the above style guidelines we will also be using the Alloy approach when developing the front end in Titanium.

Enforcing The Guidelines

We will enforce these style guidelines by pairing up within teams and exchanging code at the end of each sprint. Each partner will review the other person’s code and give feedback to ensure that they are following the guidelines.

While we plan on having coding partners to formalize to practice of code reviews. As developers we believe in ownership over our entire system and we will practice the review of our peers code across our team.