RecruitR

Team Avalanche

Zane Grasso David Dubois Maxwell Hadley John Murray

Project Sponsor

Jim Bondi - RIT Career Services

Faculty Coach

Timothy Whitcher

Project Overview

RecruitR, also referred as Tinder For Career Services, is the project that was assigned to Team Avalanche. RecruitR is a mobile application that aims to create a useful prototype that can make the recruiting process easier as well as quicker for both recruiters and job seekers (specifically students). The application will have three main users: the recruiter, the student, and system admin. The goal of the recruiter will be to find acceptable student candidates with desired skills for a certain position and eventually give them an interview. The tool will assist in the skill matching and selections for the recruiter as well. For the student, the goals will be to eventually land that interview for a position that they have the skills for. The job position/recruiter can also be filtered out by students.

The high level scope of RecruitR is to provide a prototype to the project sponsor. As stated above the final goal of both parties(recruiter and student) is to set them up for an interview for a specific job posting. Before and interview is rewarded, there are three other phases that must be completed prior. We define these phases as Matching Phase(student only), Problem Phase, and Presentation Phase. The last phase, the interview, we define as just Interview Phase. The following section describes each phase and how the application flows between them on both the recruiter and student side. Each phase represents a different screen within the app, not including login and registration pages.

Matching Phase

The matching phase is only completed by the student. When a student registers an account, he/or she must enter their skills before the rest of the app can be used. These skills are used to be matched with Job Postings (see Job Posting Creation). The way skills are matched with jobs is referenced later in the document. Once skills are entered, students can view their homepage, which is a list of job postings that they have been matched with. In this phase, students can view a job postings' information and decide if they are interested in a certain position or if they find no interest in the position. If they are not interested in the posting, the matched posting will be

removed from their homepage. If the student decides they are interested in the job posting, they are moved on to the next phase within that job posting, the problem phase. The recruiter that created the job posting can now see that the student is interested in the position within their dashboard. The only information that is shown about the job posting in the matching phase, is the position's title, the name of the company, the location of the company, and the company's website.

Problem Phase

After a student selects that they are interested in a particular job posting, they are then moved on to the Problem Phase. The overview of this phase is that it a chance for the recruiter to provide students with a problem, or problem statement, and give them a chance to answer/respond. This phase is not meant to provide a problem like you might see at a technical interview where you would have to solve it to the best of your abilities, but more to introduce student with material and issues that the position might deal with. On the other side, this gives recruiters the opportunity to review students' responses to these problems and see what sort of experiences they have and how they can use their experiences to apply it in the positions environment.

On this screen, students can view the problem statement. Once the statement is read, they then write a response in the text field that is given to them. When the response is finished and ready to submit, the student submits the response. If the student feels they are not interested in the position anymore after reading the problem, they have the ability to opt out of the specific job posting. They will no longer be active on this posting and will not be able to view it. The Problem phase is not the only phase in which the student can opt out. A student can opt out of a job posting at any point of the process. Recruiters can also decide they are no longer interested in a student at any point. When a recruiter is no longer interested in a student, that student is removed from the job posting.

Once a student submits a response, the response is immediately sent to the recruiter for review. The recruiter then decides if they are interested in the student continuing in the recruitment process. If the recruiter decides they are still interested, the student moves on to the next phase, Presentation Phase. If the recruiter is not interested after reading their response, the student is removed from the job posting.

Presentation Phase

If a student has made it past the Problem Phase they are brought to the Presentation Phase. This phase is like the last, except instead of reviewing a problem and a response, a recruiter and student are reviewing a presentation. The student views a presentation of the company, that the Company submits on company registration, therefore one company has one presentation, no matter the job posting. A presentation can be in many different formats, including: video, slide show, or a combination of the two, but must be posted on youtube and submitted as embedded youtube link or the share link.

After the student has viewed the company's presentation, they then have to decide if they are still interested in the job posting. If they aren't, they are navigated out of the posting. If they are, then they must create a presentation to send to the recruiter. This presentation is meant to sell the student to the recruiter and help identify why they are right for this position. Again, this presentation can be in many different formats. For the scope of this project we decided that all presentations would be an embedded youtube video, so all students/companies would have to do is insert the link of the video. There is a time limit in which the student has to respond, with a presentation, or the job posting is automatically removed. (See Job Posting Creation). Once a presentation is created and sent off to the recruiter, they must wait until the recruiter decides if they are still interested in them. If the recruiter is still interested in the student after viewing their presentation, the student is moved to the next and final phase, Interview Phase. If the recruiter is no longer interested, the job posting is removed from the student.

Interview Phase

Students must have "passed" the last two phases in order to get to the Presentation Phase. This is the final phase of the recruitment process and the goal of both the student and the recruiter; to get to this point. This phase is where both sides get to see more information about each other and one last glance before they contact each other to schedule an interview. We also call this the reward phase because both parties are rewarded in a certain way, in which we will explain below.

Student's Reward
When a student makes it to this phase they are now able to see remaining information about the job position that you would normally see on a job posting. Information like: position description, more detailed location, and recruiter contact information.

Recruiter Reward
Once a recruiter accepts a student from the Presentation Phase to the Interview Phase, they can now see more information about the student. Information like their resume and contact information.

Job Posting Creation
A job posting is obviously only created by the recruiter. This is where a recruiter inserts all the necessary components that are needed for the student to view throughout each phase (except presentation; Company) . Each posting will need the following information:

- Position title
- Location of position
- Description of the position
- Required skills
- Recommended skills
- Response timeout

- Problem statement

Required skills are weighed more heavily than recommended skills when looking for student matches. This will be described more in depth in the design section (See matching algorithm). Response timeout is the amount of time a student has to respond in that phase. This timer will be from days to weeks. If a student doesn't respond by the specified timeout, they are automatically kicked from the posting. This timeout is also include in the Interview Phase.

Job Posting Dashboard

Unlike a student, the recruiter's home page is the list of job posting that they created. They also have a dashboard, where they can also view how many students are in each phase of their job posting. Each student's information is kept confidential, until the interview. Other dashboard items include editing a job posting and profile management.

User Registration

Users that don't have an account must register a new one. For students, they must provide the registration screen with their school email in order to sign up. If the user does not have a school email, they can not register as a student. Recruiters must provide the screen with their company email. If the company email is not in our systems database, then the company has to register (See company Registration,.

New Company Registration

We require that new companies have to be verified before recruiters can register and start making job postings. The reason for this is to prevent system abusers using the application to troll other users. For example, creating fake job postings. Whenever a new company registers, a application admin has to approve the company before anything else can be completed under that company name.

In conclusion, the overall application's goal is provide the sponsor with prototype of and IOS/Android app that streamlines the recruiting process and makes it more efficient. The end result that both main users are striving for is getting an interview scheduled for a job posting.

Basic Requirements

System Features

Student login and profile management

This feature allows a student to create and maintain a profile. Students can enter their contact information, skills, and preferences for what sort of jobs they want such as preferring large companies or preferring companies in coastal cities, and may upload their resume. They can also update this information as it changes over time.

Functional Requirements:

- Students can create new profiles
- Profiles shall be identified by the student's email address
- Student profiles shall always contain the student's name, email address, school, anticipated graduation date, and skills; optionally phone number
- Students must upload their resume
- Students may change all profile attributes aside from email at any time after initial profile creation.

Recruiter login and company profile management

This feature allows recruiters to create and maintain profiles for their company. Company profiles have the company name, location(s), size, a brief description, and a presentation which exhibits the company's culture. Recruiters may edit this information at any time.

Functional Requirements

- Recruiters shall be able to create profiles
- Recruiter profiles shall contain the recruiter's name and company email address
- Recruiter profiles shall be associated with one company profile
- Company profiles shall have company name, location(s), size, description, link to a youtube presentation about the company, and the domain of all emails used by recruiters of that company

Administrator login and system management

This feature allows administrators to create other administrators, approve or deny new recruiters, approve or deny new recruiters, and view statistics and reports about RecruitR. This feature was not implemented over the course of the project due to time constraints.

Functional Requirements

- Administrators shall be able to create a new administrator which includes a email, password, and username
- Administrators shall be able to approve or remove newly registered companies.
- Administrators shall be able to view database information relevant to site operation through the administrator website.

Recruiter job posting

This feature gives the recruiter the opportunity to post a job position that they would like to find

students to potentially interview for. These postings contain all the information that a student would be able to see if they end up navigating through all the phases. The posting shall also include information on how long a student may spend at each stage before the student is considered to have been rejected

Functional Requirements

- Recruiters can create new posting
- Recruiters must enter the name of the position, location of position, description of position, and the skills that are required and/or recommended
- Recruiters must enter the amount of time that a student may spend on each phase before the student is automatically rejected
- Recruiters must enter a problem statement

Student job matching

This feature gives the student the ability to look through job postings that match their skills and opt into the application process with the company

Functional Requirements:

- Student is matched with job postings automatically based on the intersection of their skills and the required/recommended skills of the position
- Student can begin the application process with a matched job posting
- Student can view all matched job postings
- Student can sort matched job postings by most recent, most matched skills, location, and company size

Problem phase

This feature allows recruiters to tell student the sorts of problems that students will work on at their company, and allows students to respond with project they have worked on that solve these problems, or to talk about how they might solve these problems.

Functional Requirements

- Job postings shall have a list of problems or a problem statement that the student will either be expected to solve or respond to while on the job
- Students shall be shown the problems while they're applying for a job
- Students shall be prompted to respond with how they would solve these problems, or with how they have solved these problems in the past
- Students shall create and submit a response
- Students shall submit a response
- Recruiters shall see the student's response
- Recruiters shall either accept or reject a student's response

Presentation Phase

This feature allows students to view a presentation about the company they're applying for, create a presentation about themself and their skills, and allows recruiters to view the student's

presentation

Functional Requirements

- The recruiter's presentation shall be shown to all students who have applied for a job at the recruiter's company and who have passed the problem phase
- Students may accept or reject the recruiter's presentation
- If a student rejects a recruiter's presentation, the student shall no longer be considered to be applying for the job
- The system shall record how many students reject an recruiter at the presentation phase
- If the student accepts a recruiter's presentation, the student shall create a presentation about themself and their skills
- The recruiter shall either reject or accept the student's presentation

Interview Phase

This phase will be awarded to students who have been accepted through all phases by the recruiter. This phase includes the postings' job description, a calendar where the student can select a interview date and time slot.

Functional Requirements

- Student must be able to view all of the postings' information such as position title, company, description, location
- A student must be able to view the contact information of the recruiter
- Students and recruiters have the ability to remove the interview phase match once the interview is complete

Recruiter Job Posting Dashboard

This feature will allow the recruiter to view all applicants for a job posting, as well as what phase of the application process the applicant is at. The recruiter will be able to view notifications about the status of the job posting, such as actions needed and interview schedules.

Functional Requirements:

- Recruiter will be able to view the name and phase of all applicants currently in the application process
- Recruiter will receive notifications on this page when an applicant reaches a stopping point or interview for the job
- Recruiter will be able to view and edit the interview schedule
- Recruiter can cancel interviews
- Recruiter will be able to navigate to individual application steps for each applicant
- Recruiter will be able to sort applicants by name, phase, and most recent activity
- Recruiter will be able to view the resume of applicants in the interview phase
- Recruiter will be able to navigate to their profile and job posting management pages

Student Menu

This feature allows the student to view their scheduled interviews, job postings currently active,

navigation to profile and skills management.

Functional Requirements:

- Student must be able to view all active job posting they are interested in
- Student shall be able to view all of their scheduled interviews
- Students shall be able to navigate to their profile and skill management pages
- Student can cancel scheduled interviews

Constraints

Our team dealt with numerous constraints. The first, and probably the biggest constraint was undefined requirements. Requirements gathering took a few more weeks than expected and this is largely because there were undefined requirements. Going into the project the sponsor and his team had a general idea of what they wanted to accomplish. Not having concrete idea of what the application must be lead to countless sponsor meetings where the majority of the time was conversation and brainstorming requirements. A lot of the requirements were left to us, the team, to decide. This lead to a decrease in development time with a time constraint.

Another constraint that the team encountered was Xamarin. Originally we were set to develop our application through Xamarin native apps. Meaning, we would have to develop the UI separately for iOS and android. After 2 week of the team implementing separate UI's, we decided to switch from native to Xamarin Forms, which allows us to work on both UIs concurrently. Though it took us a couple days to complete what we had natively, we still had the constraint of last minute UI tool change. In addition to this, there was only so much Xamarin Form could do. There were still a few instances, file upload and download specifically, where we had to develop natively for each platform. This was a constraint provided to us by the tools.

Other constraints included not having enough time to complete certain tasks, the need to learn new technologies which created a lot of overhead when trying to get the ball rolling on implementation. And also we had bugs arise as we got closer to the final delivery that took us some time to correct for our working prototype.

Development Process

Our development process was rapid prototyping, which was our decision and chosen because due to the less defined nature of our project we wanted to get as much sponsor input on our development as possible. The primary mandate of the rapid prototyping process was that after every development iteration, we would provide the sponsor with a working prototype on which we could get their opinions. This ensures that the project stays in line with the sponsor's image of the final product, and gives the team an idea of what isn't working if we need to change anything.

Our sponsor has given us a lot of leeway with our methodology, so we picked the process that best served our needs, while also making sure that our sponsor is confidant in our work. The primary role identified in our process is the need for someone to present our prototype, and while we all attend the weekly sponsor meeting, we decided it would be best for our group leader and sponsor coordinator to be in the driver's seat for any demonstrations required to show new functionality. We are all required to make sure that the whole team is informed as to how

completed features work, so ideally anyone could fulfil that role if the need arises.

Project Schedule: Planned and Actual

Planning Process

The majority of the project schedule planning was based on the delivery of features and on our 2 weeks iterations for our process. The plan outlined at the end of what 2 week sprint (by date) each feature could be expected to be delivered. The current activity breakdown of the project plan is as follows: Requirements Elicitation, Code/Architecture design, Development and Testing, and SE Deliverables (Poster, Presentation, etc.). Then other than the end of each iteration where a small feedback loop process is performed with the sponsor, the only major milestones are the interim delivery and the final delivery.

Execution of the Plan

At writing of this report the first milestone, interim delivery, has just passed. According to the project plan outlined on the project plan document, the only features planned for delivery that have not been delivered is the application phases and job posting filtering. This is likely due to the fact that for requirements elicitation we went longer than the plan had described. Originally we thought requirements would be more solidly defined coming, however since they weren't it required more time than expected to get them fleshed out. Another issue that caused delay was that during the beginning of implementation when all the team members thought they had their development environments setup, there was a variety of issues that kept the majority of team members from developing until about a week after the planned start. Lastly, we had a risk come to fruition that we had to mitigate; the tool we planned to use, Xamarin, was not as easy to use as originally anticipated. After about a week and a half of client side development all UI development had to be scrapped to switch platforms within Xamarin's set of tools to make things easier. This however only set us back about a week, and allowed us to gain more momentum as the new tool was easier to use.

Adaptation to Changes

Because our schedule had changed quite a bit throughout the term, our team had to quickly adapt. This wasn't that difficult because after we realized our schedule was off, we had a better understanding of our velocity and were able to estimate more accurately. As scheduling changed, so did our priorities to work on this project. Once we realized how much time was actually needed to be successful, everyone started putting more time in.

When the summer term started up we had to adapt our schedule to the more time each team member would have available for working on the project. This meant there would be more commitment to task, increase velocity, and more overhead required for planning because there was more work to plan out. Our adaptation was successful and we were able to achieve all of the above.

The last adaptation was the desire for the sponsor to provide demonstrations to 3rd parties. The product would no longer be just internal. As such we adapted by establishing a day when the code would freeze and planned that the iteration before this freeze would be focused on mostly bug fixes of existing features instead of trying to get new features done. With this, even though parts of the prototype are incomplete, the things that are completed are bug free (from what we

can tell).

System Design

Overview

The start of the design process began with each individual team member preparing a mental image of the overall architecture. These models were based on expertise from co-ops and previous projects. Then when the team came together to discuss the architecture our models were not very different from each other.

The overall consensus was to use a server-client architecture blended with a MVC architecture. In this case the server side would have controllers and the model of the MVC while the client side would also have controllers but instead with views. The duplicate controllers can be thought as the tracks on both sides of a train tunnel. The controllers pass information back and forth between each other and then handle what happens to the data after the communication on their respective sides.

In terms of data communication a simple REST API fits very well into our architecture. Since views can invoke controllers and controllers don't sporadically invoke themselves communication is only performed when the user interacts with the client side application. So as an examples if a user visits their matches page, the view will invoke the client side controller which invokes the server side controller via the REST API and then the server controller retrieves data from the model which is then sent back as a response to the API call and used by the client-side controller to show on the requested view.

There is only one other part of the server architecture to cover; services. Services are essentially server exclusive controllers invoked by the other controllers that are apart of the REST API. They perform manipulations on data within the server. What makes them different is the manipulation of the data is not on models associated with the services instead it is will the models associated with the controllers in the REST API.

With the integration of these services and the REST API the server architecture was finalized for the start of implementation as seen in figure 1.

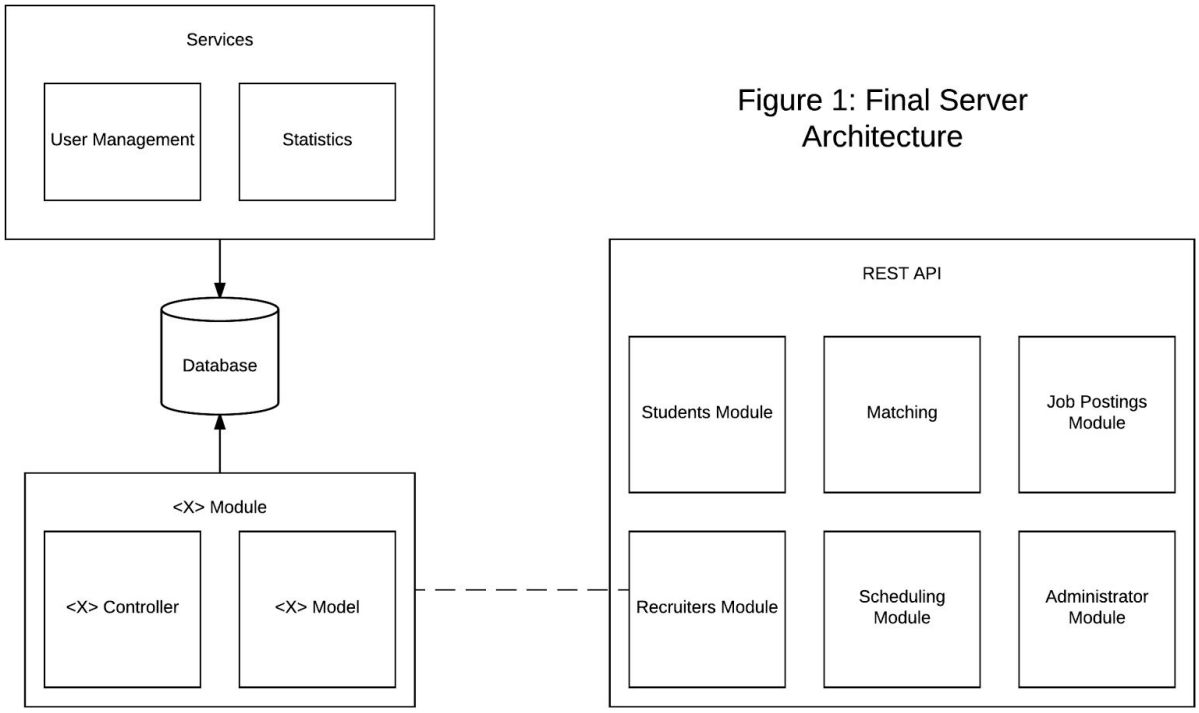

Figure 1: Final Server Architecture

As for the client side architecture the only additional specification is that the controllers are responsible for communication and that in our diagram of the client architecture the model is only the representation of the data maintained on the server and retrieved/updated by the communications in the controller. This can be seen in Figure 2, the final client architecture for the start of implementation.

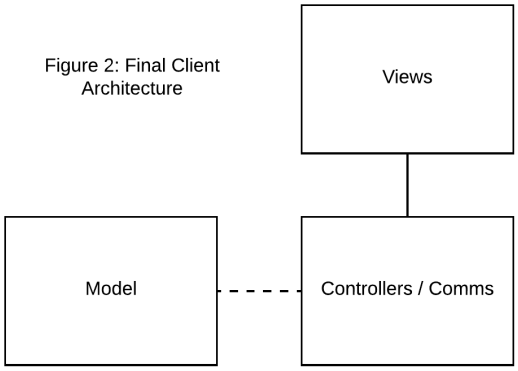

Figure 2: Final Client Architecture

Controllers

When moving "down" as it were, the controller's functionality was outlined as it's REST API endpoint calls as can be seen in figure 3.

Figure 3: Outline of controller functionality

Module	Student	Recruit	Recruiter	Job Posting	Admin	Scheduling
Models	- Student	- Skill - Match - Application	- Recruiter - Company	- Job Posting	- Admin	- Schedule - Time Slot
Controller Endpoints	- Register Student - Edit Student - Get Student	- Start Application - Submit Problem Response - Approve Problem Response - Deny Problem Response - Submit Student Presentation - Approve student presentation - Deny student presentation - Get student reward - Get employer reward - Get Application	- Register Recruiter - Register Company - Edit Recruiter - Get Recruiter - Edit Company - Get Company - Approve Company - Deny Company	- Create Job Posting - Delete Job Posting - Get Job Posting - Get Job Postings for Recruiter	- Create Admin - Generate Statistics	- Create Schedule - Get Schedule -Schedule Interview -Cancel Interview

How has it gone?

So far implementation has gone fairly well. Everyone on the team seems to grasp the structure of the architecture and how to translate from diagrams to the code implementation. We haven't had any trouble following through with our designs into implementation and therefore have not required any overhauls or refactorings of our design.

Future Considerations

Our implementation adds for the easy addition of modules or services as necessary without interfering with any of the existing code. Looking forward it appears as though we shall be able to easily accomplish our goals with the current design. That being said we are aware for risks regarding the necessity of architecture redesign and have prevention techniques in place, however the probability is low.

Process and Product Metrics

Product Metrics

We employed the following product metrics:

- Number of bugs per thousand lines of code
- Test code coverage
- Cyclomatic complexity
- Percentage of tests that passed

The number of bugs per thousand lines of code told us how well we were writing code. If this number got too high, we would have known that we were horrible coders or that we had horrible team communication, and we would have had to revisit our development process. Fortunately, this metric never got too high.

Test code coverage tells us what percentage of our code has tests written for it. We had hoped to get above 80% for all new features, we didn't have a very good idea of how to run UI testing without using a paid for framework. As a result, our unit tests were focused on the more complicated server code with us manually testing the phone UI, and it was reflected by us having a large amount of bugs in the phone UI code that we just hadn't seen before.

Cyclomatic complexity tells us how complex our methods are, and thus how hard to test and how bug-prone they are. This metric is low for most of the code, rising to nine only for methods which perform validation on data objects. When this metric gets too high, it's probably time to refactor the method for lower complexity. However, we haven't been formally checking this metric throughout the project.

Percentage of tests that pass tells us if we've found any bugs in our code. Gradle, the tool we're using to build the server code, will fail to build if any tests fail, This ensures that we don't have any obvious bugs in our code.

Process Metrics

We employed the following process metrics:

- Time tracking
- Slippage chart

The time tracking was used to track how much time we spent working each week. We were given the guideline that we should spend about eight hours per week on senior project, so that's what we all aimed for.

The slippage chart was originally intended to help us adjust our schedule as needed. If the slippage chart showed that we were behind on our schedule, we would adjust our planning to not put so many tasks in an iteration so that we'd have time to catch up. On the other hand, if the slippage chart showed that we were ahead on our tasks, we would be able to add more tasks to the next iteration and get ahead on our work.

The slippage chart ended up not being necessary, since its goals were accomplished by simply looking at the sprint board for our current iteration. We could quickly check the sprint board and see how close we were to completing tasks. In addition, we talked amongst ourselves and were easily able to tell when tasks would take longer than expected, meaning that the slippage chart, if it had existed, would have been superfluous.

Product State at Time of Final Presentation

The following major requirements have been implemented in the current prototype:

- Students can register new profiles
- Recruiters can register a new profile
- Recruiters can register a new company profile if they are the first recruiter from a company to register
- Recruiters can create new posting
- Student is matched with job postings automatically based on the intersection of their skills and the required/recommended skills of the position
- Student can begin the application process with a matched job posting
- Student can view all matched job postings
- Student can choose to ignore a job posting, hiding that posting from their list of matches
- The matching algorithm is defined as follows: (percent of matched important skills * important skill weight) + (percent of matched nice-to-have skills * nice-to-have skill weight)
    - The important skill weight and the nice-to-have skill weight are determined by the recruiter users when creating job postings
    - Recruiters select a minimum match threshold a match must achieve to become a match in the system]
- Students can view the problem prompt
- Students can respond to problem prompt's
- Students can decline to continue with the application process
- Recruiters can see the student's problem response.
- Recruiters can tag a response with one of the pre-approved choices
- Recruiters can accept or deny a student's problem response.
- Recruiters can select a presentation for the students to see.
- Students can view a recruiter's presentation
- Students can submit their presentation to a recruiter
- Students can decline to continue with the process
- Recruiters can view a student's presentation
- Recruiters can accept or deny a student's presentation
- Students can view a recruiter's contact information
- Recruiter's can view a student's contact information
- Recruiter's can view a student's resume
- Recruiters can navigate the app by job posting and then by phase
- Students can navigate the app by phase

The following features have started implementation but need more work:

- Match expiration checking

Project Reflection

What went right?

Top-Down Design Process

We employed a top-down process to design our architecture. This process worked very well. We were able to quickly design an architecture that served us well.

Estimation Process

The process we used to generate estimates was a good one. We utilized planning poker, and were able to easily arrive at estimates. We were also able to quickly re-estimate after our first iteration, once we had a better idea of how long tasks actually took.

Team Rapport

It may be that we got lucky with our team composition, but we all worked very well together. It's not just that we were friendly and could get our work done, but we could rely on each other. We could confront each other about our opinions with regards to the product, process, or whatever and debate it without it getting heated.

Team Communication

Leading off from the last point our team communication was very good. We were in constant contact with each other almost every other day and even every day from some periods of time. We always reached out to each other to consult on ideas, implementation, or whatever else we needed. On top of that we always responded to each other quickly, making sure that nobody waited too long to get something done because they were waiting to hear back from a teammate; this is probably the biggest positive of our effective communication.

What went wrong?

Extended Requirements Gathering

We spent far longer than expected in the requirements process. This was partly due to how the requirements were not very well defined at the beginning of the process. We had to spend a few weeks at the beginning of the semester simply to nail down the main ideas of our product. After we hammered out the main ideas of our product, we still had to figure out specifics. That took another few weeks to complete. This had the combined of effect of delaying code design until after spring break, with code writing following after that.

Separate iOS and Android Apps

When we first started development, we settled on Xamarin since that would allow us to develop both the iOS and Android apps from the same codebase. Since we didn't know what we were doing, we used Xamarin's Android and iOS GUI libraries, which meant we had to develop the UI code for iOS and Android separately. Not only did this negate one of the major advantages of Xamarin, but we quickly discovered that Xamarin's iOS UI library was incredibly difficult to use, so much so that we ended up spending two weeks creating a single UI screen when the equivalent screen took a single day to make with the Android UI library. We switched to the

Xamarin Forms UI library, which not only let us write the same code for iOS and Android but was also much easier to use than Xamarin's iOS UI library.

Updating of Scheduling Information

Our team was not proactive at updating tasks with the time they actually took to complete. This means that we're unable to accurately evaluate our estimates and thus unable to improve. This did improve greatly toward the second have of the summer term, but by then it was too late.

Using New (to us) Technologies

This point is related to the problem with developing an iOS app. We didn't have a lot of experience developing with Xamarin or iOS in general, which caused many hurdles during the first couple iterations of development.

Committing to too Much

We over-committed for the first iteration. This isn't too bad on its own, except that we over-committed so badly that what we thought would take one iteration ended up taking two.

A big part of that was due to the issues with iOS development mentioned above. We also had issues with team members getting their environments set up. We should have planned for setting up development environments and learning the necessary tools, which would have gives us much more accurate sets of tasks to be completed in each iteration. This could have been a sprint 0. If it has been executed that way our over commitment wouldn't have had such a harsh effect.