Team 7 - Design Document

Ashvin Lohiya, Akshat Goyal, Shubhang Kulkarni, Palina Rawat, Aakash Keswani, Sidhant Chadda

Index

Purpose

Currently in career fairs, companies have long lines of students wanting to talk to them. Students attending career fairs spend most of their time standing in these lines rather than networking and talking to potential employers. While scheduling systems exist, none have been used to solve this problem. Our product will aim to tackle this issue by creating virtual queues, which will reduce the time students spend standing in line through various optimization techniques.

The days of waiting in long queues to talk to recruiters at career fairs are over. Our system will be used by students to schedule themselves in company queues virtually where optimization techniques will reduce their waiting time to the minimum. Also, recruiters will use our system to keep track of students they have talked to and who they will converse with next.

Currently there is no such application or system in existence which solves the problems faced by students and recruiters in career fairs. Career fairs are a big part of every student's university experience and we aim to make the whole experience much more efficient and better. Our system will enable students to talk to more companies in less time and thus have more career opportunities and options than before.

Functional:

1. As a user, I would like to be prompted if I'm a student or a recruiter.
2. As a student, I would like to login using mypurdue credentials.
3. As a student, I would like to be able to see the list of all the companies.
4. As a student, I would like to be able to see each company's profile
5. As a student, I would like to be able to filter companies based on my class standing.
6. As a student, I would like to be able to filter companies based on my major.
7. As a student, I would like to be able to upload my own resume.
8. As a student, I would like to rank the companies by preference.
9. As a student, I would like to be able to see my current meeting schedule.
10. As a student, I would like to be able to cancel my meeting appointment.

11. As a student, I would like to be able to edit my profile.

12. As a student, I would like to be able to filter companies based on internship or full time opportunity.

13. As a student, I would like to be able to filter companies based on visa requirements.

14. As a student, I would like to be able to get an estimate of the wait times in different queues.

15. As a recruiter, I would like to be able to put my company's login credential in the system.

16. As a recruiter, I would like to login into to the scheduling system.

17. As a recruiter, I would like to be able to see the queue of students.

18. As a recruiter, I would like to be able to upload my company profile.

19. As a recruiter, I would like to be able to edit my company's profile.

20. As a recruiter, I would like to be able to dequeue students.

21. As a recruiter I would like to be able to see the resume of the student.

22. As a recruiter, I would like to be able to choose the time estimate for talking to a student.

23. As a recruiter, I would like to be able to mark certain students.

Design Outline

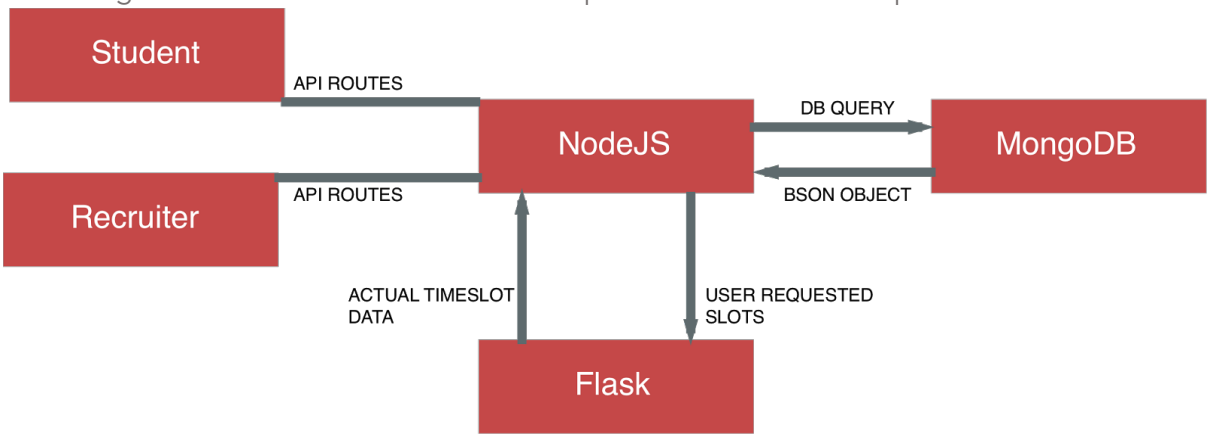
We plan on using a micro services architecture approach to build and scale our product. Our implementation of this will include having both a node js server to handle basic routing / Database features as well as a flask python service which will handle the recommendation systems. On top of this we will implement front-end tools for both the recruiter as well as the student entering the career fair.

1. Web Client
    a. Our web client will be implemented using react js a front-end library that allows for component driven development.

   b. Our web client will receive and send information to the node.js server via http requests with the JSON format.
   c. Our web client will use the JSON object and map it to existing react components to make use of react.
2. Node.js API Server
   a. Our API server will handle all incoming http requests from both the student and the recruiter. It will parse the appropriate JSON object from the client and convert it to a model which we can store to our MongoDB database.
   b. Our API server will also have to connect to our flask recommendations service in order for us to compute the appropriate time slots for each meeting.
   c. Lastly our API server must send information back from our database to our client as per request.
3. Flask Recommendation Service.
   a. Our flask recommendation service will be responsible for computing all information regarding the appropriate time slots for each individual company.
   b. Our flask service must also maintain communication to the node js server via http requests in order for information to be sent and received between the two.

High level Analysis:

Below is a high level diagram depicting overall design of our product. Breaking down product into individual modules allows for increased productivity in teams and better scaling products. We take this point a step further by utilizing a micro service architecture for our backend. This will split up the work of the backend into two core components responsible for different tasks. Our flask service will simply perform the algorithm responsible for computing the timeslots for each individual student, whereas our Node.js service will allow constant communication between the users and the service by handling routes and database connections. This type of architecture allows us to be able to boost up the performance of a single machine if there is a performance bottleneck for any individual component while allocating less amount of resources for components that do not require as much.



Design issues

Functional

Issue 1: Class organization

Option 1: Make single class for user, and sub-classes student and company/recruiter

Option 2: Make separate classes for both company/recruiters and students

Since recruiters/companies and students have very few similar attributes, and because companies and students are going to be managed in two distinct databases, we decided to choose two different classes for companies and students.

Issue 2: How students select preferences

Option 1: Choose the companies before entering the career fair

Option 2: Choose the companies after entering the career fair

This way students won't need to waste time looking through company descriptions and shortlisting them while the career fair is going on.

Issue 3: How students login with authentication

Option 1: Login with Purdue Career account authentication

Option 2: Login with LinkedIn

Option 3: Maintain our own database of users

We have decided to use Purdue credentials to login students into the system as our scheduling system will be exclusive to Purdue recruiting fairs for now. We think that using Purdue credentials make the system more robust and trustable. Option 2 was very intriguing but we decided not to go with it because not 100% of the students might have a LinkedIn account or maybe they don't want to show their profile to the recruiters or any other personal reason. Option 3 was out of the play just because maintaining our own database of users would be just extra work and less secure compared to Purdue login.

Issue 4: How do recruiters enter company information

Option 1: Open ended system where recruiters can enter information freely

Option 2: A close-ended system where recruiters fill in information in predefined fields.

We decided to go with a close-ended system where recruiters will fill in the company information in predetermined fields. The reason for doing this is to have a systematic filtering system. If we give companies predetermined fields like do they take international students or not, or an option to specify the class standing they want to talk to, it would be easier for us to provide students with the information in an organized way.

Issue 5: How recruiters handle a no-show/late student

Option 1: Recruiters flag the student and move to next student in the queue.

Option 2: The spot becomes open and anybody can take that spot.

We decided to let the recruiters flag if a student didn't show up or show late because by this way students already in the queue will be given a chance. If we make the spot open, students could go in without putting any information in the application which could then clash with future meetings.

Non-functional

Issue 1: How are we going to host our back-end services

Option 1: Use Amazon web services.

Option 2: Use Heroku.

We plan on utilizing Amazon web services instead of heroku because of the issue of scalability. With amazon can automatically vertically as well as horizontally scale our product. This is useful because if their are too many users than we expected their to be we can easily scale to handle the influx. Heroku on the other hand is easier to get up and running faster, because of its ease of use / integration. This makes it better for smaller scaled applications.

Issue 2: Which languages are appropriate to implement the backend

Option 1: Python with flask

Option 2: Javascript with Node.js.

Option 3: Use both.

We are planning on utilizing both Node.js and flask for our backend services. We are going to use Node.js to handle routing / database management. We are also going to utilize flask to handle the recommendation services. The two will communicate together to transfer data via http requests. We plan on using this micro services architecture because it better modularizes the features of our product. Using this architecture also allows for more scalable products. By eliminating bottlenecks on single machines.

Issue 3: What framework should we use for user-interfaces

Option 1: Angular js

Option 2: Jquery

Option 3: React

React uses virtual DOM. This concept enables us manipulating and changing virtual copy of the DOM that is lightweight and then only pushing the changes. React is unique in its syntactic and conceptual simplicity. Most of the developers find it simpler than angular and richer than jquery.

Issue 3: How students are scheduled

Option 1: based on absolute times

Option 2: based on relative scheduling

Students would be able to see all the company profiles before the career fair (or just after the career fair starts, according to the organizing department's policy). Students select the companies that they want to talk to, and rank them according to their personal preferences. These get stored in a priority queue within the student's timetable. When the student arrives at the career fair, he/she gets enqueued in only his top preference company initially. The estimated wait time is calculated. If this estimated wait time is above a certain threshold time, then the students wait time is pipelined, by enqueuing the student into his next preference company, and this process continues

Pros of option 2:

- Models the actual process of a career fair, and adds an element of efficiency as the students' wait time will be utilized productively
- Students arriving late will be able to speak to their top preference company (provide there is sufficient time remaining for the fair) as they will be pushed ahead of all the students who don't have the company listed as their 1st preference

Downfalls of Option 1:

- Students arriving late might not get to meet any of their top company preferences
- This approach does not model a career fair. It would rather model scheduling interviews
- Students would not be able to change their top company preferences dynamically, as if they were schedule with a company, changing the preference would dequeue them for the company queue, thus leaving gaps in the time queue.
- The absolute scheduling would have to be done in fixed intervals of time, every 15 minutes say, as otherwise, a company's slot may get filled up as soon as the career fair opened
- Pipelining the waiting times would be difficult

<u>Issue 4: What profiles need to be maintained in the database</u>

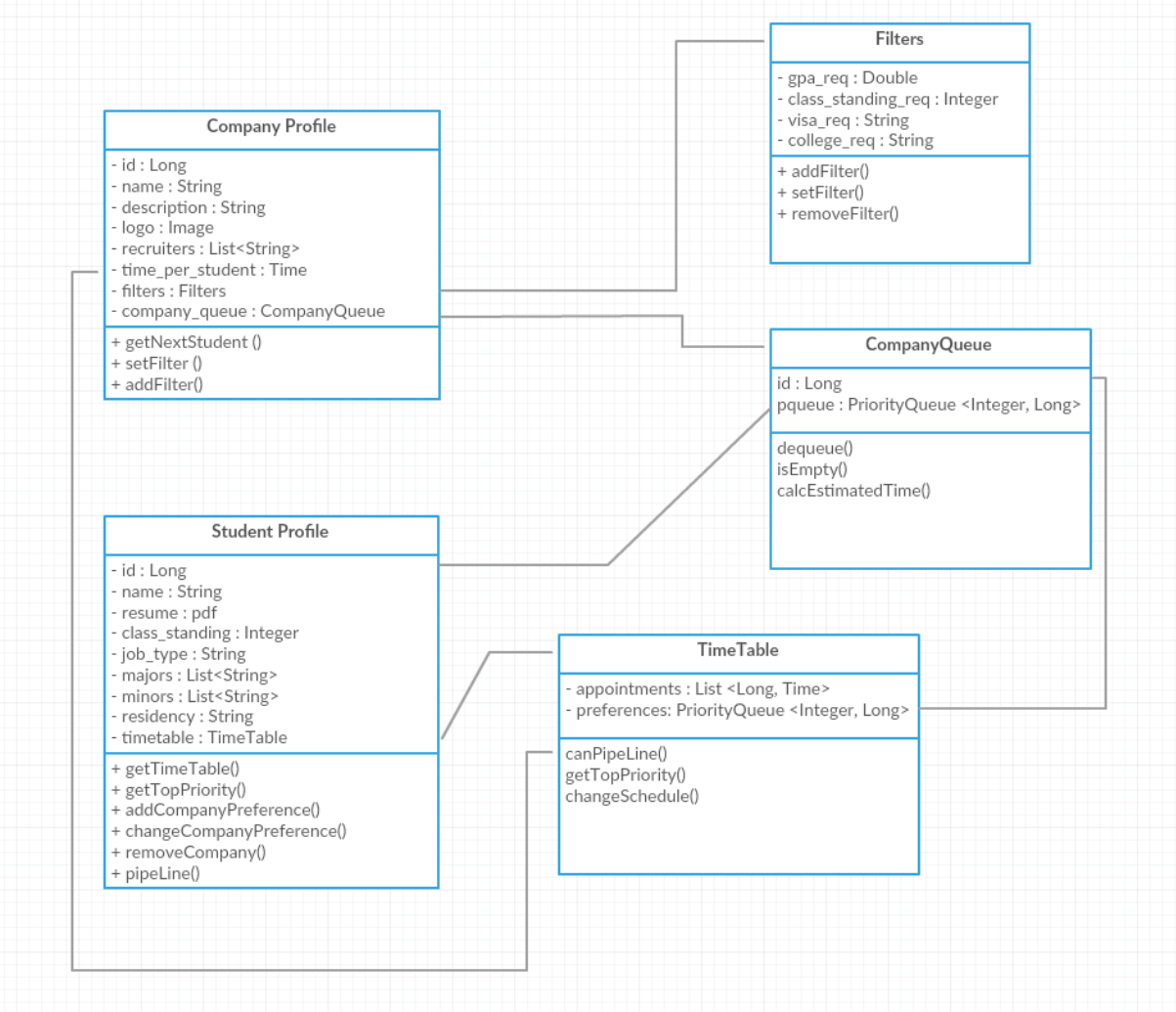
Option 1: Only Students

Option 2: Both Students and Companies

In career fairs, the recruiters need information about students and vice versa. That's why we will give profiles of companies (Which would be filled in by the company itself) and the students (Which would be filled in by students)

Design Details

Data Class Level Design



Description of Data Classes and their Interaction

Student Profile:

Whenever students register on the app, they will create a unique instance of this class on our database. Once this instance is created and stored in our database, every time the students logs back in to the application, his student and password combination will be used to access this instance pull the corresponding data out of the database.

A student profile, as the name suggests, is an objective representation of the student on the application. It will have all the necessary objective information that a company needs so that they understand more about the student.

Following is the data scheme of the student profile class:

- ID – This is the unique student identifier, which will enable student authentication and profile access. It will also be the identifier by which students can enqueue themselves in company queues.
- Name – String whose data value will be the full name of the student
- Resume – data value acts as an identifier which allows the viewer (both student and recruiter) to access the student's resume
- Class standing – Refers to the standing of the student corresponding to credits of the student. (freshman/sophomore/junior/senior/graduate student/post doc etc)
- Major[] – an array of all the majors that the student is officially pursuing
- Minor[]- an array of all minors the that the student is currently pursuing
- Residency – The residency status of the student
- Time table – Every student will have a time table which will hold all his appointments and make the actual scheduling possible

Time Table:

Every student will have a unique instance of this class to hold his schedule. The time table will have the following attributes:

- Appointments – A list of all the company preferences which have been scheduled, including his 1st priority company as well as the other companies that have been scheduled to pipeline his/her waiting time.
- Preferences – Contains list of all shortlisted companies by the student according to the priority he/she selected.

Company Profile

This class contains the details of the Company. It contains all the information that is related to the company that a student may be interested in. There will be a password for every company, that the representative selects, and can reset based on the contact information provided. This class contains all the information that is displayed to a student when they are looking up a company at the career fair.

- ID - Unique ID for every company (Primary Key).
- Name - String containing name.
- Description - Contains description of the company (Set by the company representative)
- Recruiters [] - List of recruiters/ their positions/ contact information (optional).
- Image - The logo of the company.
- Time_Per_Student - Amount of time a company plans to spend on each student.

- Queue - The queue of students to be scheduled for an appointment with the company at the career fair (Priority Queue based on Company Ranking and TimeStamp)
- Filters – A reference to the filters table that contain all company restrictions.

Filters

Each company has an instance of this class. This class is used to define all the restrictions the company has for students. This will be added when the company profile is being created where the company will be asked to enter their information in pre-defined restriction fields. Students will be asked to fill in their information and will be enqueued according to the criteria they meet.

Following is the data scheme of the filters class. Note that this will include more attributes

- GPA Requirements - Company can add a minimum GPA requirement for the students.
- Class Standings - Company can ask for students with particular class standings.
- Visa Requirement - Whether the company takes students who might require sponsorship.
- College Requirement - Company can specify a major that they are looking for.

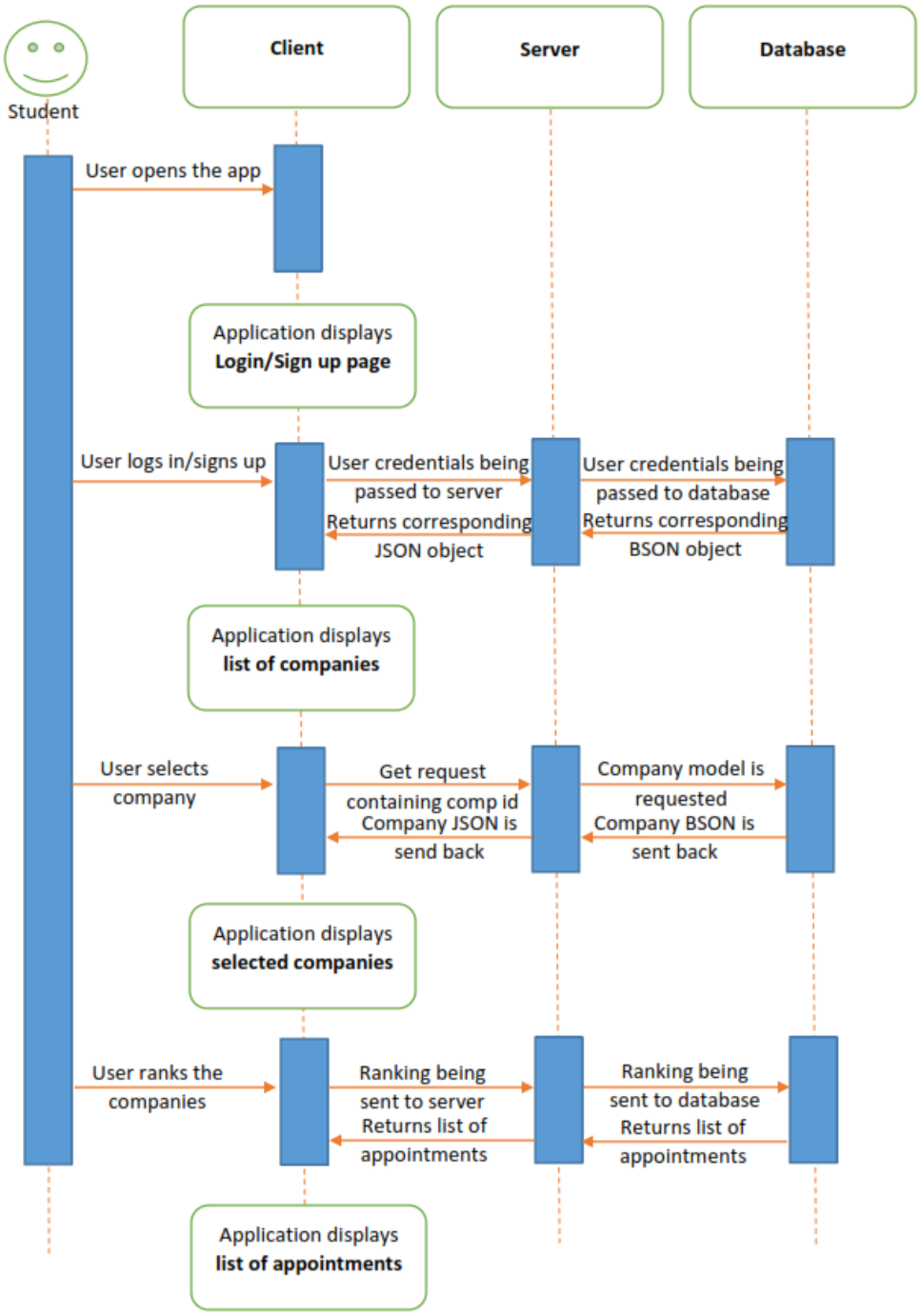
Company Queue

This table is instantiated by each company. This table contains a list of all student IDs that are currently enqueued to talk to the company.

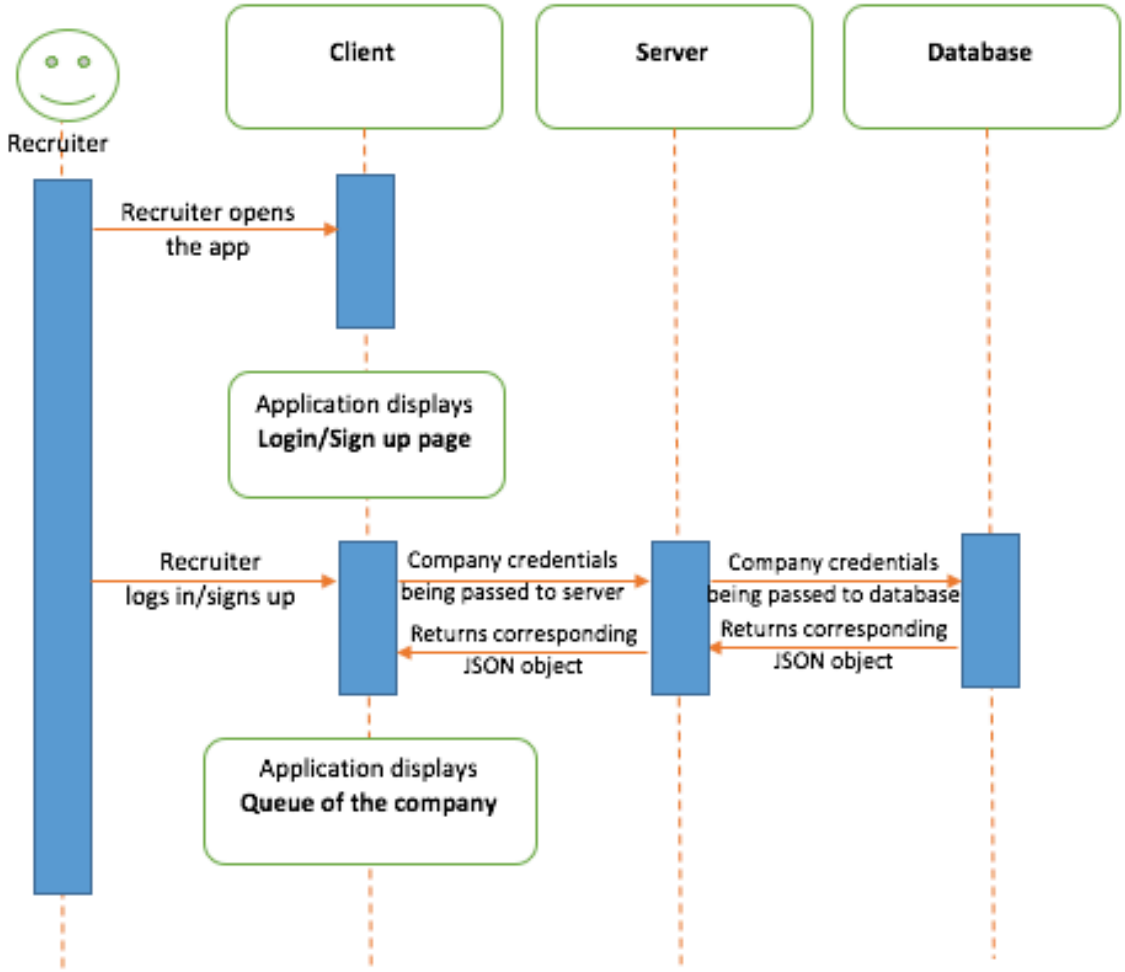
Following is the data scheme of the filters class. Note that this will include more attributes

- ID – The identification of the company queue that references this instantiation to the company
- pqueue – This is the list of students' IDs that are currently enqueued in the company list.

Sequence Diagram

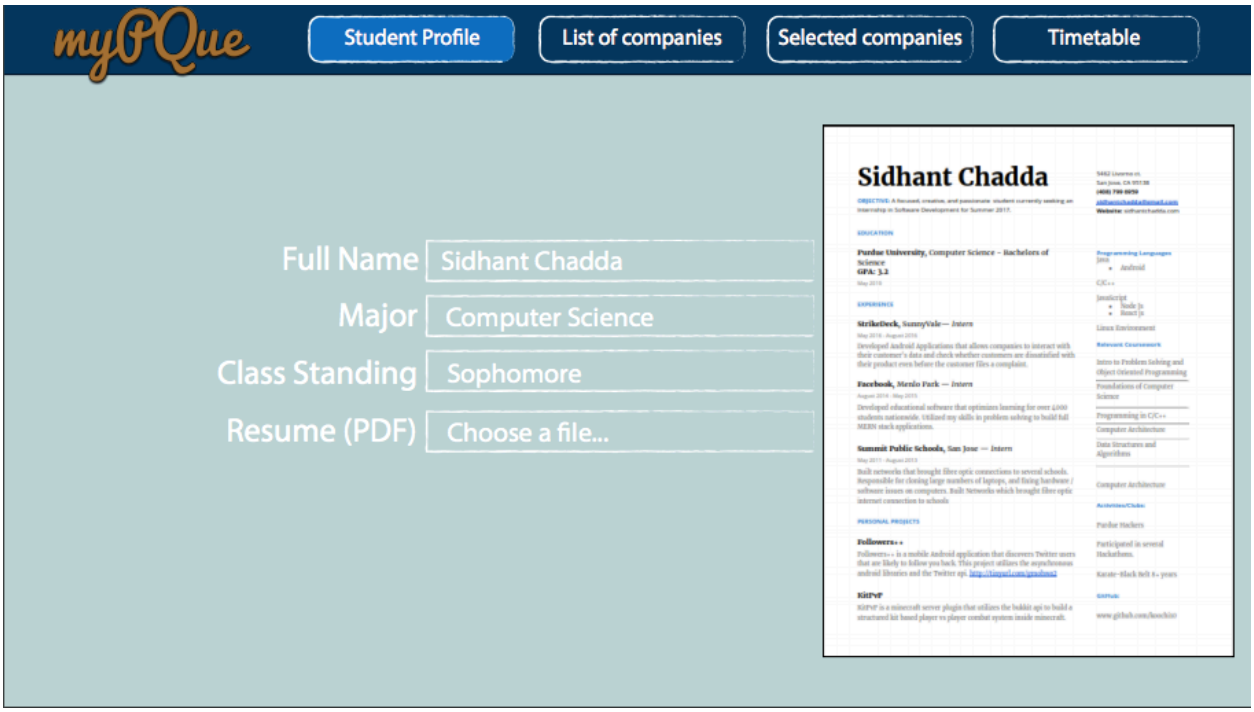


Sequence of events that the student will take during the course of career fair and before.



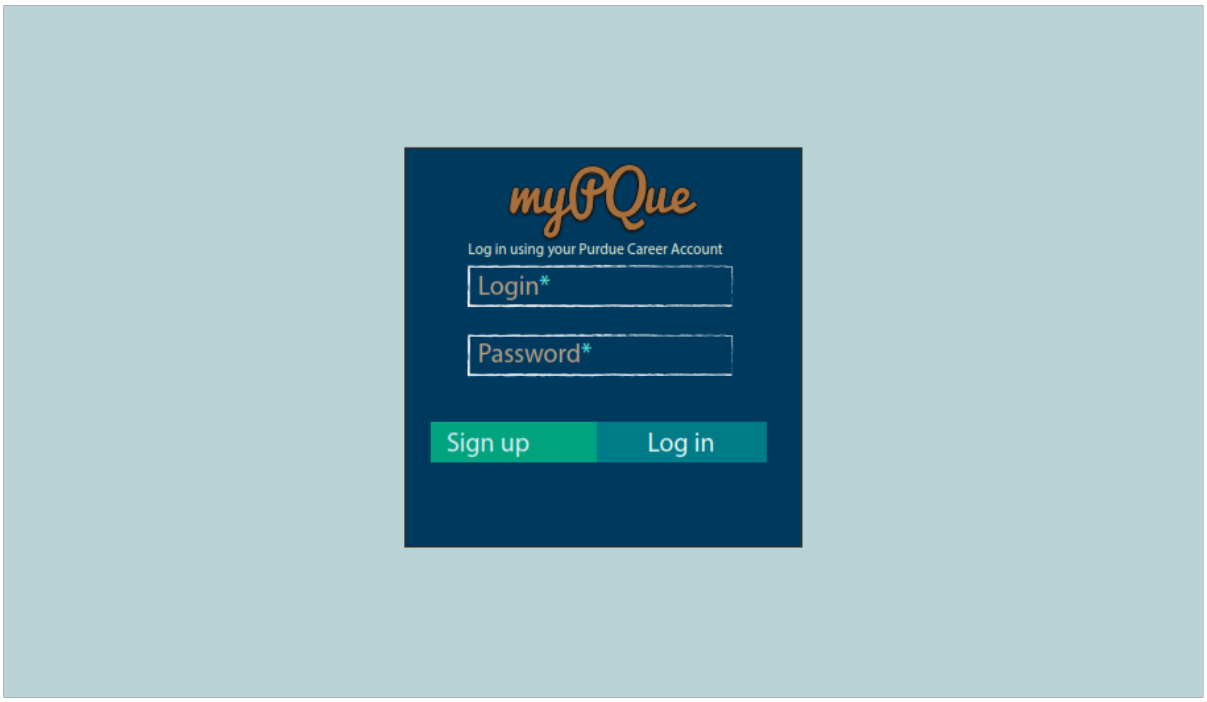
Sequence of events that the company recruiters will take during the course of career fair and before.

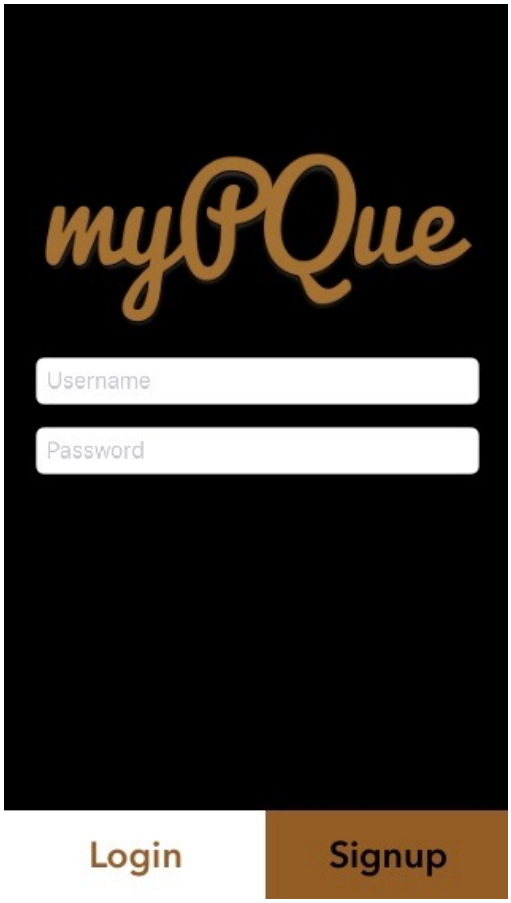
UI Mockups



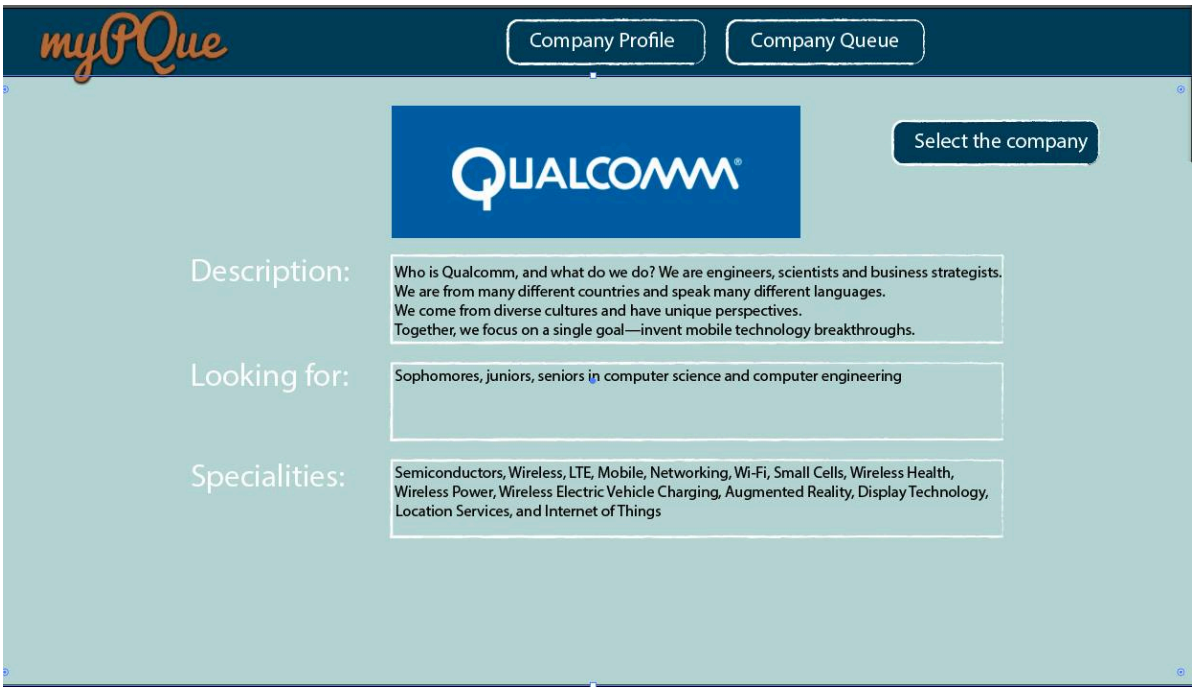
Index page for Student - the webapp is a single page application with three tabs.

Student Profile - Here students will see their details including their resume.
List of Companies - Students can find all companies here with their details.
Selected Companies - Students can find all of their shortlisted companies, their priorities and their time table in this tab.



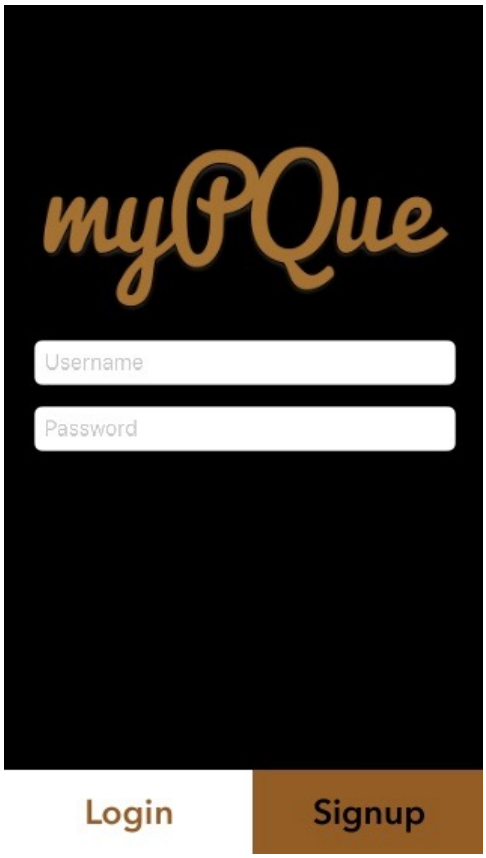


Login/Sign-up page for student, where they can log in using Purdue Career Account.



Index page for recruiter

Company Profile - Here recruiter can edit/see the details about the company.
Company queue - Recruiter can see how many students have enqueued the company queue, look at their profile.



Login/Sign up page for recruiters