Chapter 1 - Product Description

Introducing HokieCFA, the ultimate solution to the long queues and waiting times at in-person career fairs. With an average of 30 employers attending each fair and a person spending an average of 15 minutes in each queue, attending a career fair can be a time-consuming process. But with HokieCFA, all you have to do is register and wait for a notification!

Our optimized queueing and scheduling algorithm ensures that you can speak to as many employers and recruiters as possible within a stipulated time. Whether you are a student or job seeker, HokieCFA offers a seamless experience where you can assume the persona of an employer or a job seeker.

HokieCFA aims to reduce wait times per student per queue, allowing you to share your profile with employers and increase your chances of getting hired. Through our Android mobile app, you can talk to employers efficiently through a notification and queueing system.

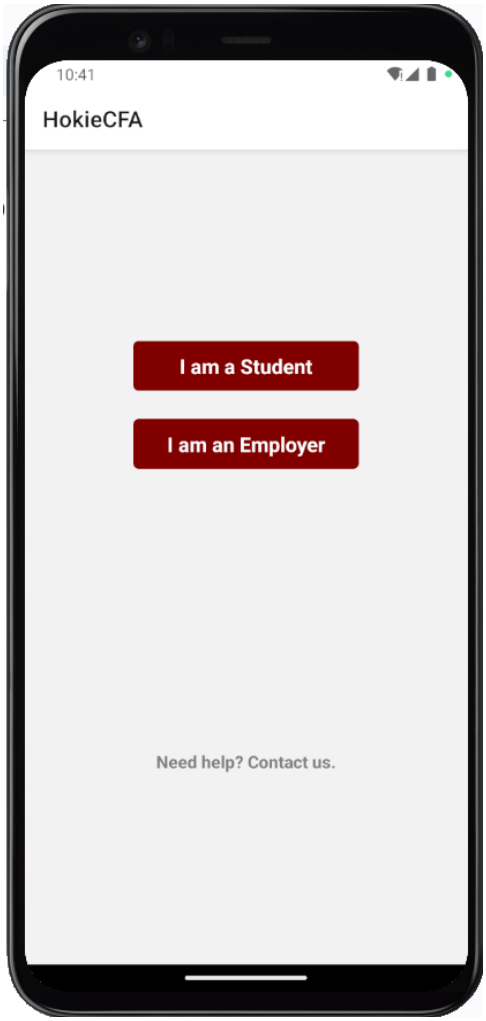
All you have to do is scan the QR code and get registered in a queue. You will receive a notification when it is your turn to speak to the employer. You can even sign up to talk to multiple employers at once, increasing your chances of finding your next job.

With a target market of 17.9 million college students in the USA and an average of 40 employers attending each career fair, HokieCFA is the perfect solution for anyone looking to attend an in-person career fair. We want to make the experience of attending a career fair easy and connect top talent with top employers in the most optimal way.

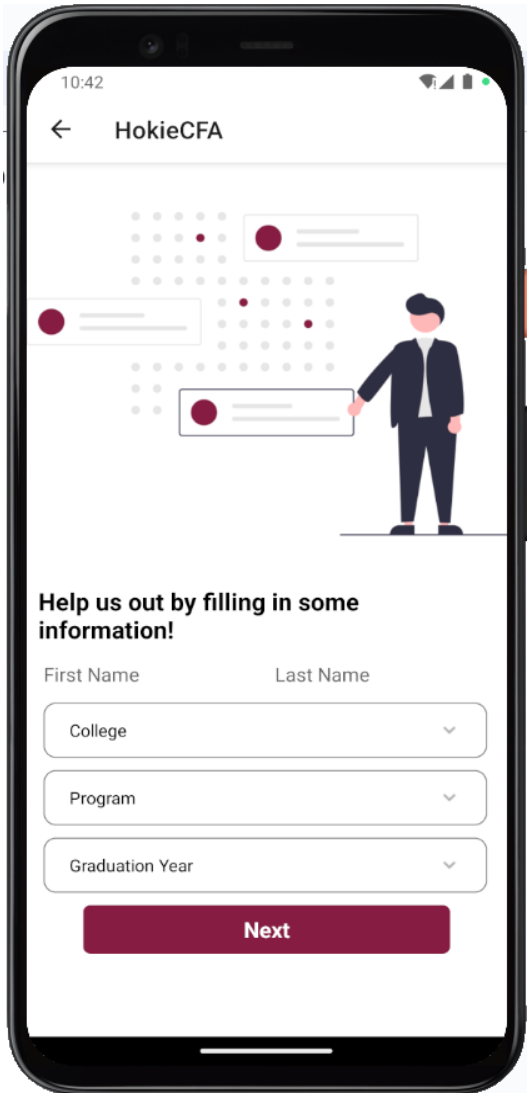
So why wait in long queues when you can register with HokieCFA and find your next job with ease? Join us today and experience the future of career fairs!

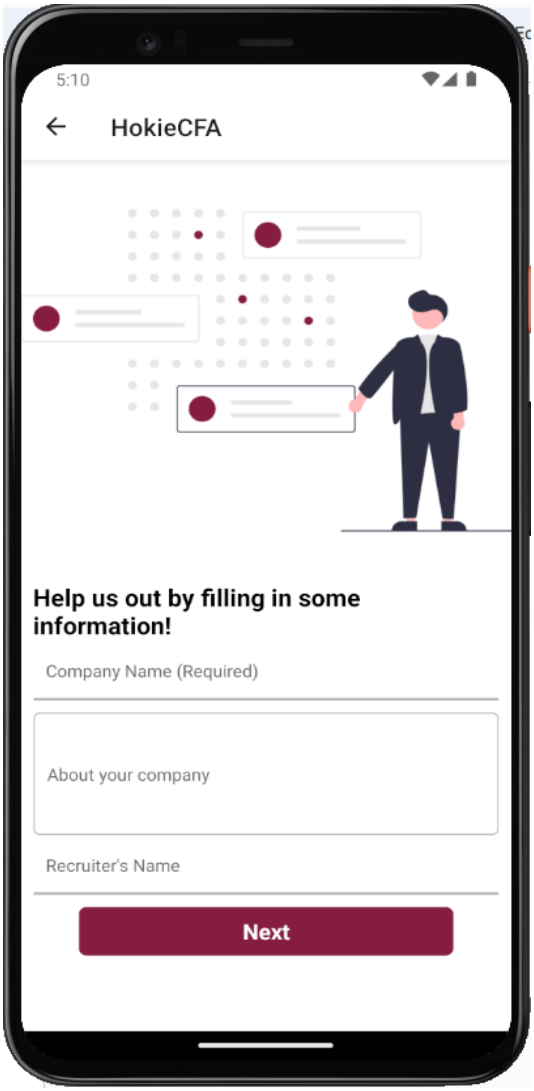
Chapter 2 - Product Functionalities

To use our app HokieCFA, the user first has to register with us. He can register with us either as a **student** or an **employer**.

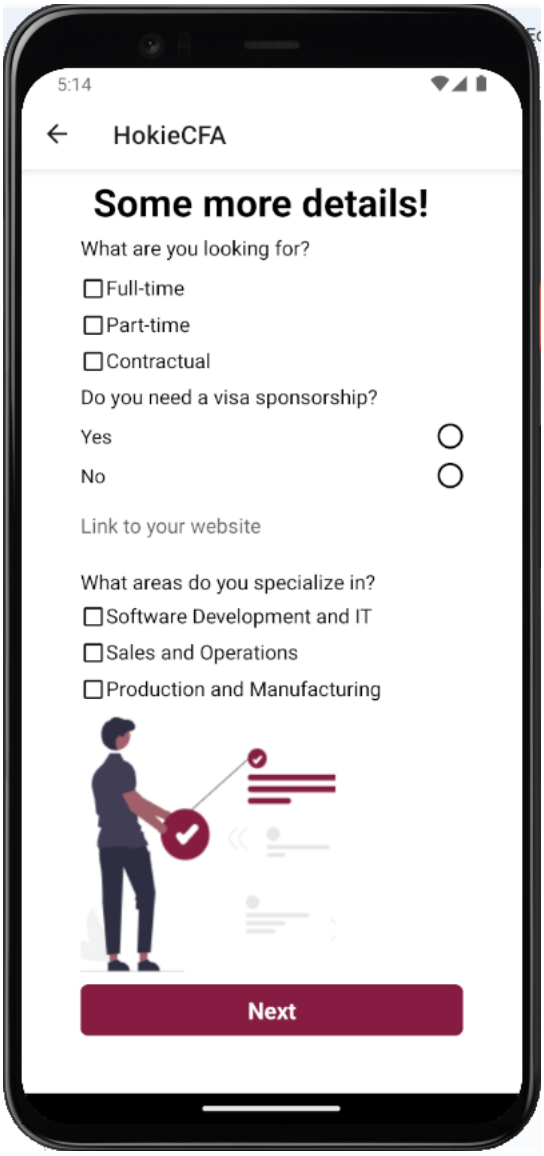


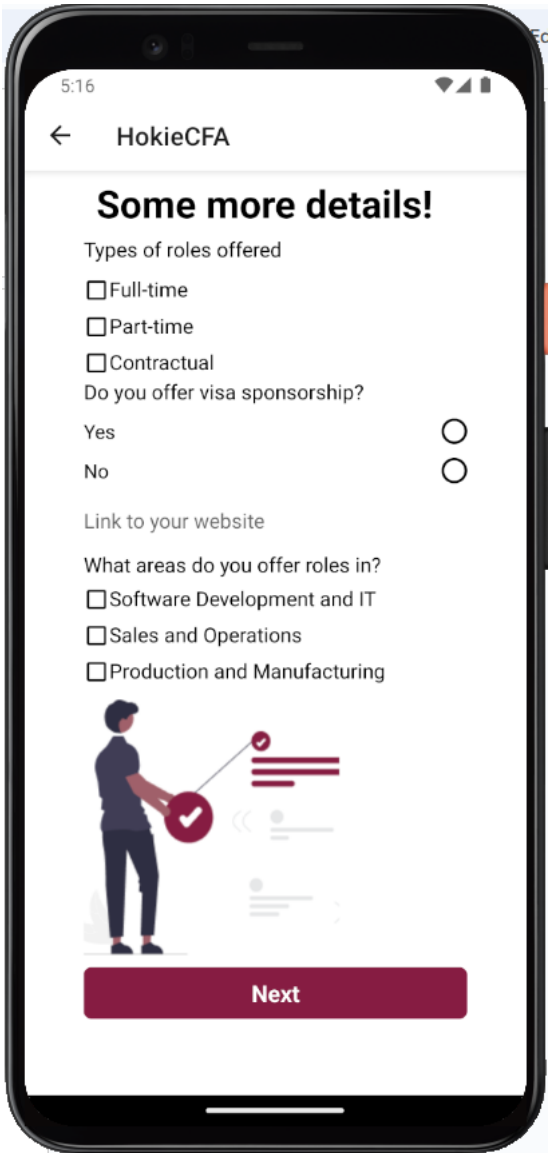
If the user registers with us as a **student,** they will have to fill out a form that details which college they are from, which program they are in, and what year theyt are graduating. Currently the app is built only for Virginia Tech students, but we could expand it to other universities as well once all the functionalities are implemented. If the user registers with us as an **employer,** they also has to fill out a form that details the recruiter's name, the company name, and a short description about their company.



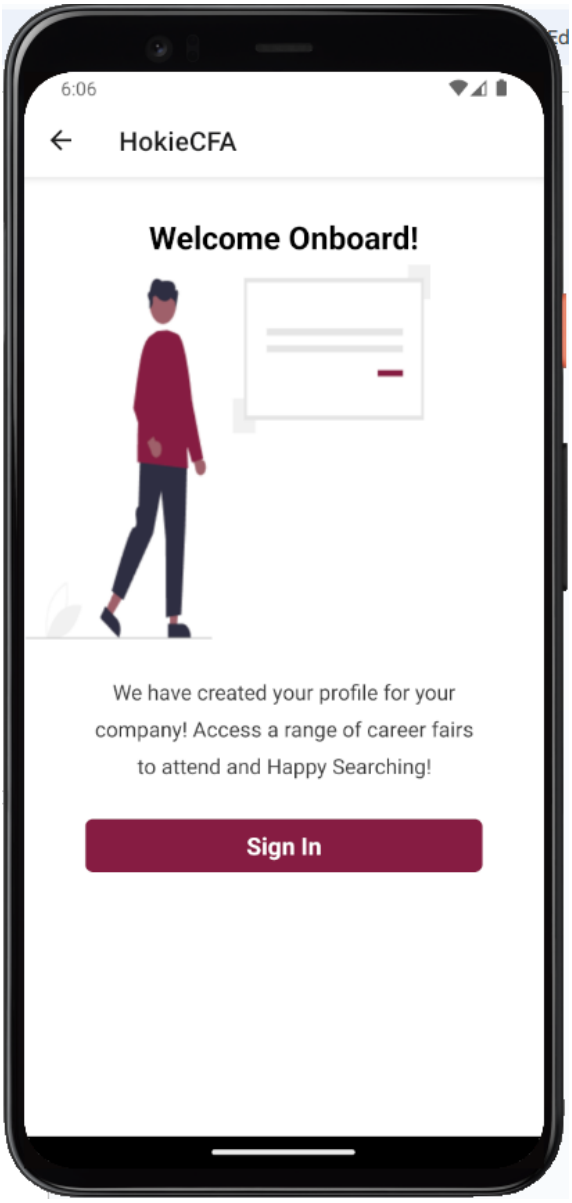


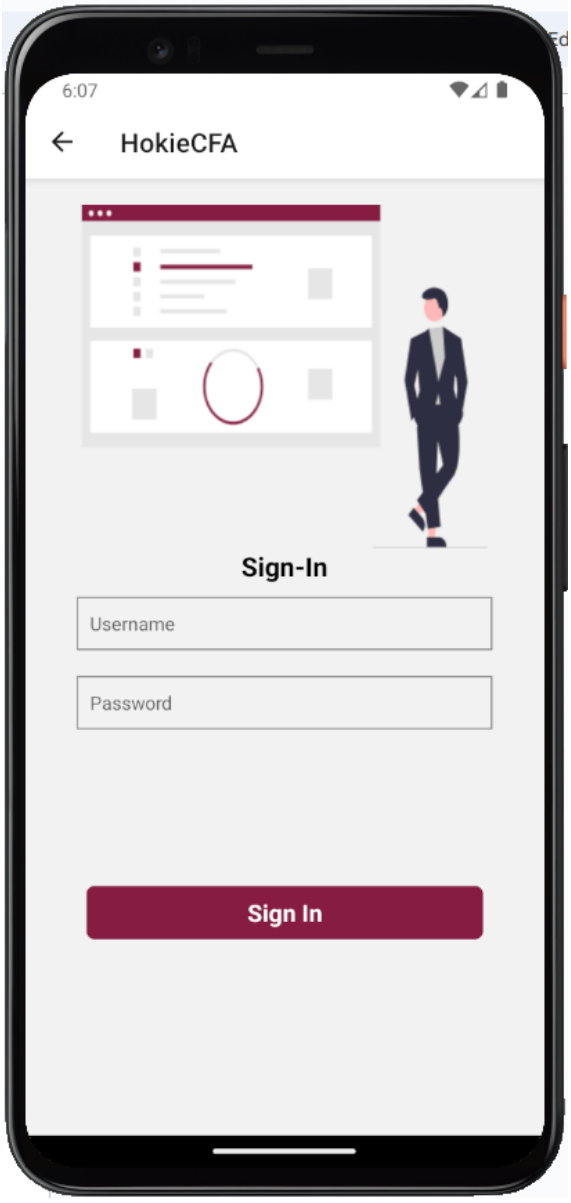
Once the user fills out the information, the student and the employer both are asked some more details. Students are asked what areas they specialize in, if they want to add a link to their personal website, if they are international students and looking for visa sponsorships, and what kind of role they are looking for. Employers are asked similar questions such as the kind of employment they offer, if they offer sponsorship to international students, the link to their careers page, and the types of roles they offer.



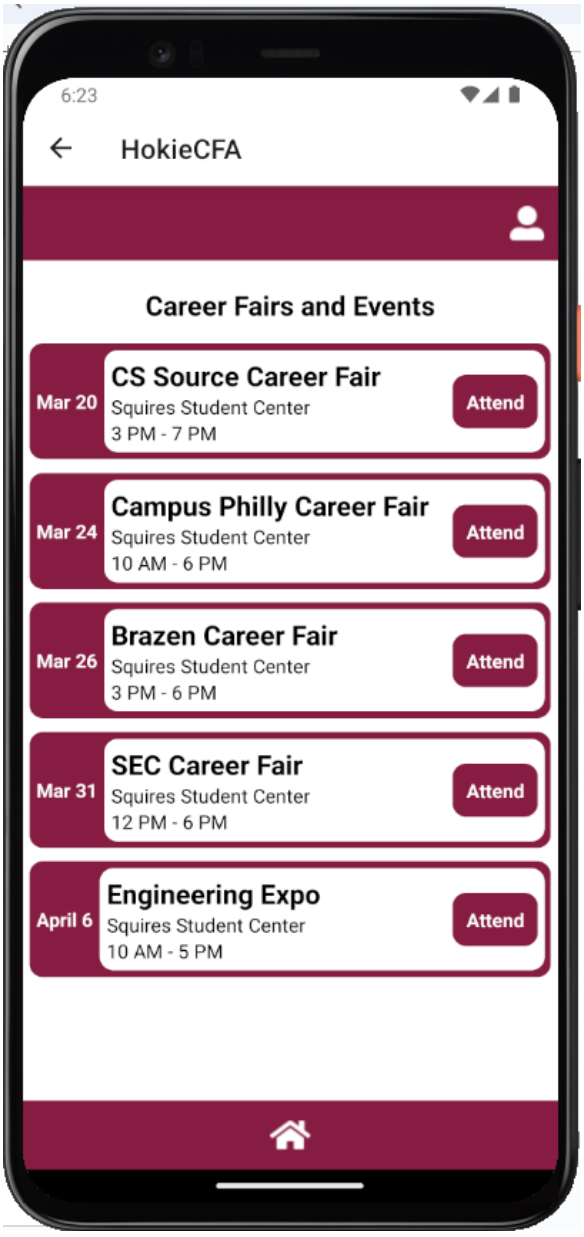


Once the student or the employer fills out the form and provides necessary information, they can click on the “Next” button and they will be registered with us. Once registered, they will then be redirected to the login/sign up page, where they can login by providing their credentials.

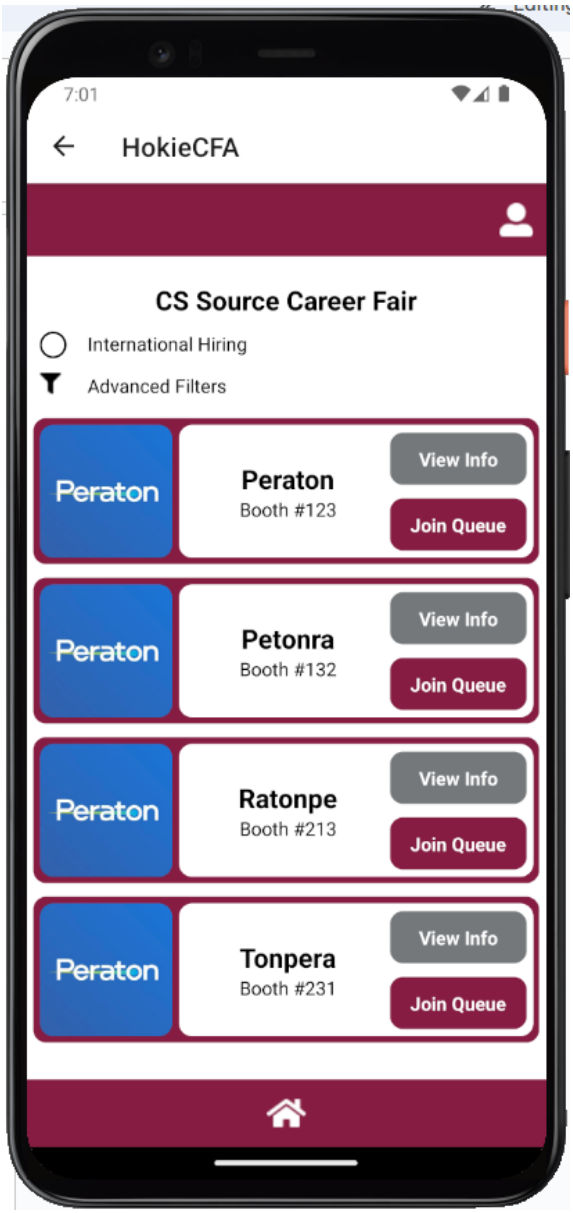


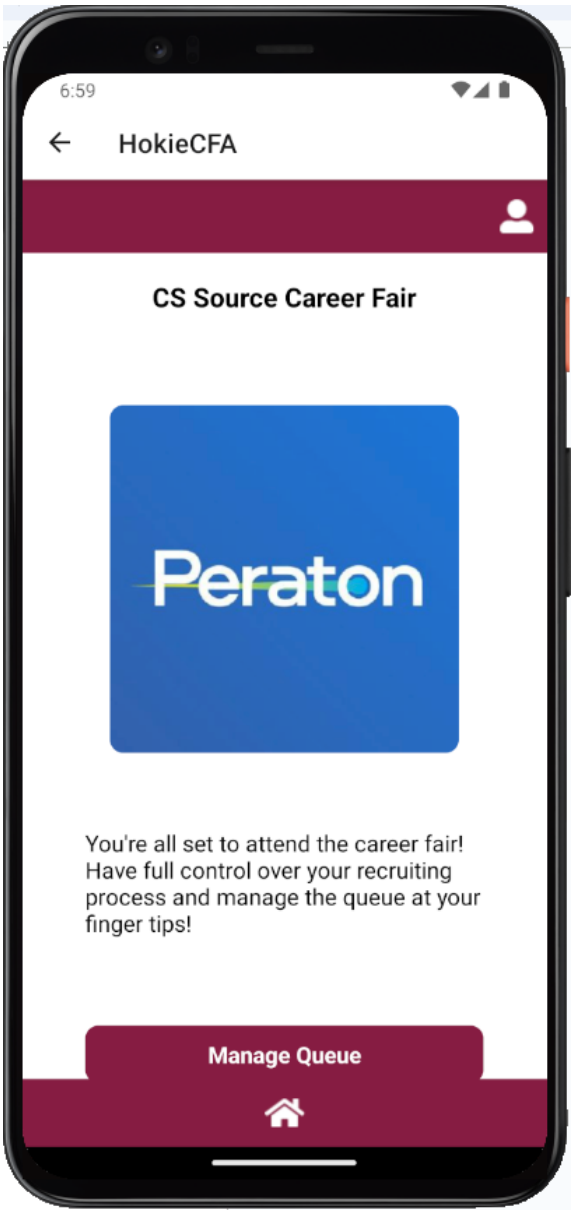


Once the user logs in, they are directed to the landing page where they can view all of the career fairs that they can attend. They can view what day the career fair is being conducted, the location, and the time for the career fair. They can attend the fair by clicking on the “Attend” button.

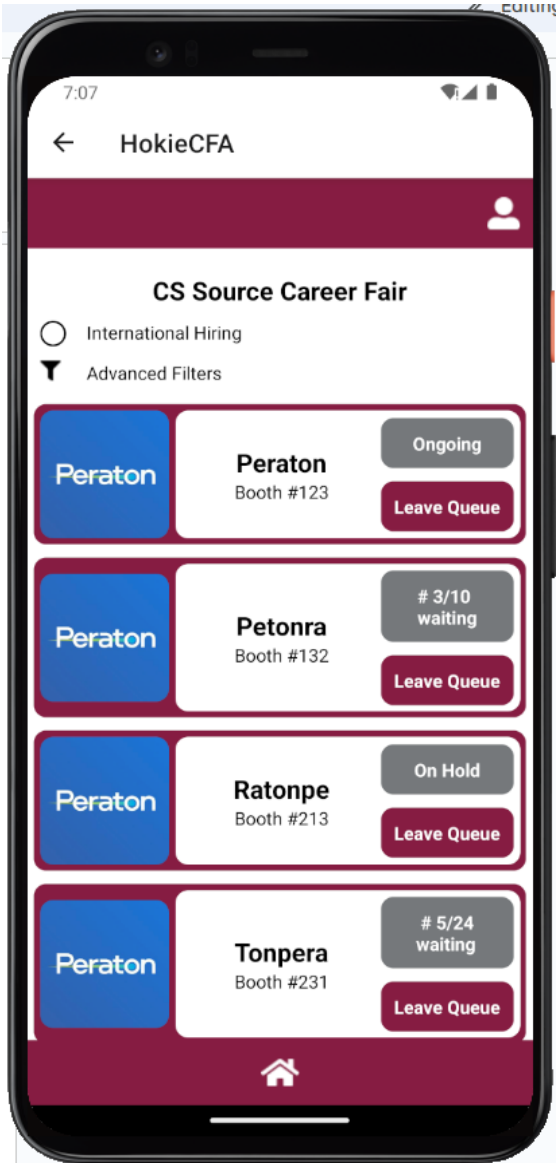


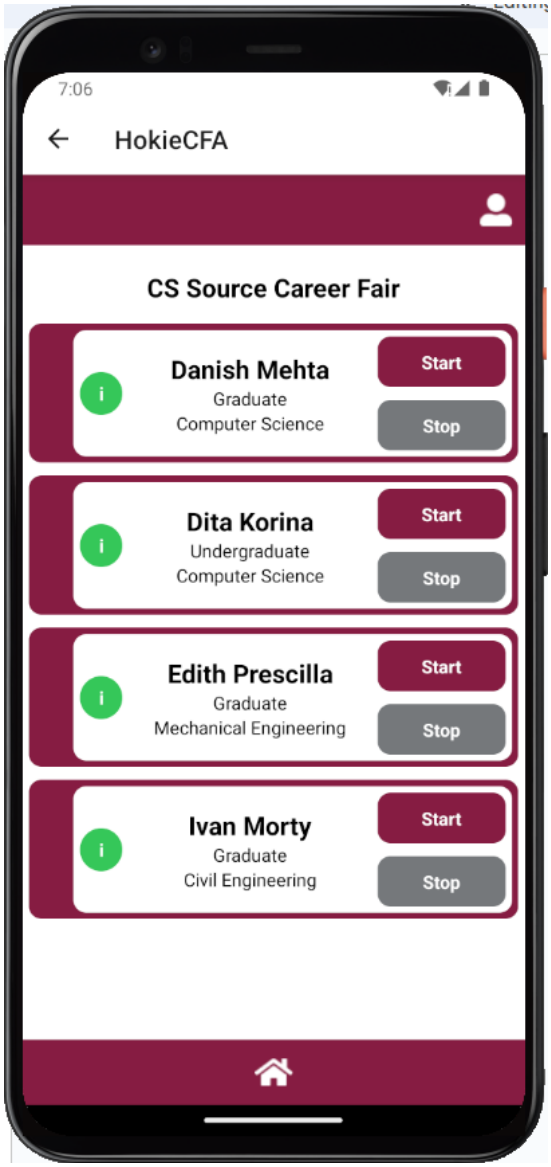
Once the students click on “Attend,” a list of companies that are attending the career fair is displayed. Students can view the company information if they click on “View Info,” and they can join the company queue by clicking on the “Join Queue” button. For the employers, once they click on the “Attend” button, they will be registered for the career fair. Once the employer is registered for the career fair, they will be able to manage the queue of students by clicking on the “Manage Queue” button.





When the student clicks on “Join Queue,” they will have to scan the QR code and they will be registered into the queue. They will then be able to see his position in the queue, whether his interview is on hold or ongoing. When the employer clicks on “Manage Queue,” they can view and manage the queue of students who have registered for interviewing with their company. They can then start the interview by clicking on the “Start” button and stop the interview by clicking on the “Stop” button after they are done interviewing the student.

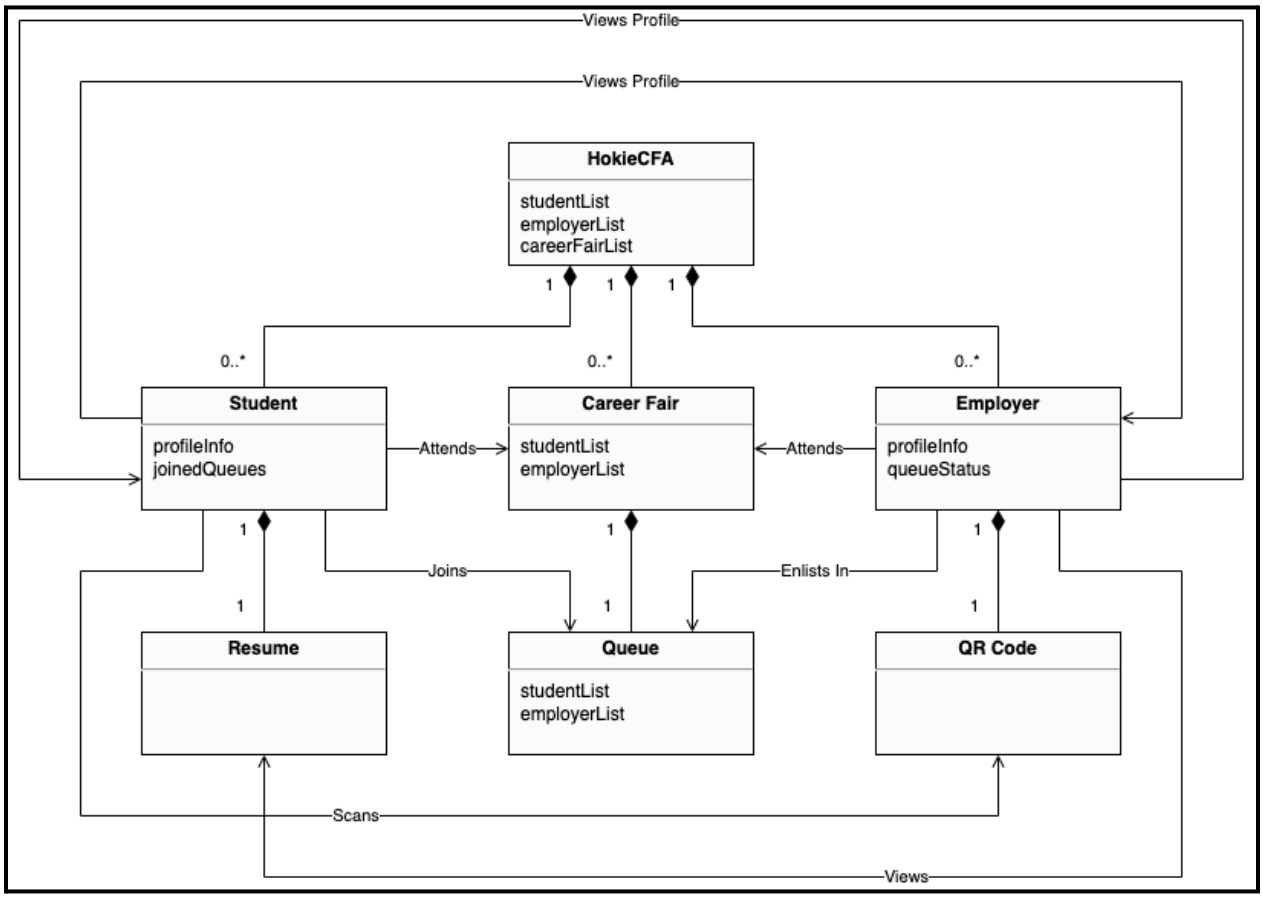




Chapter 3 - Design

In this section, we decided to include our domain model, the design class diagram and one example each of sequence diagram and collaboration diagram. Each diagram is explained in detail - complete with the various elements involved and how they interact with each other. The example of choice for both the included interaction diagrams is that of “login”.

3.1 Domain Model



The domain model diagram represents the relationships between different entities in a system for managing career fairs. Let's discuss each entity and their attributes in more detail:

- **Student**
    - This entity represents a student who is attending the career fair. It has two attributes, "profileInfo" and "joinedQueues", which represent the student's personal information and the queues they have joined, respectively. The entity is related to the "Resume" entity with a "Scans" relationship, which represents the

action of an employer scanning a student's resume. Additionally, the entity is related to the "HokieCFA" entity with a "Views Profile" relationship, which represents the action of a career fair organizer viewing a student's profile.

- **Resume**
    - This entity represents a student's resume. It is related to the "Student" entity with a "Scans" relationship, which represents the action of an employer scanning a student's resume. Additionally, it is related to the "HokieCFA" entity with a "Views Profile" relationship, which represents the action of a career fair organizer viewing a student's resume.

- **HokieCFA**
    - This entity represents the career fair organizers. It has three attributes, "studentList", "employerList", and "careerFairList", which represent the list of students, employers, and career fairs managed by the organization, respectively. The entity is related to the "Student" and "Resume" entities with a "Views Profile" relationship, which represents the action of a career fair organizer viewing a student's profile or resume.

- **Career Fair**
    - This entity represents a specific career fair organized by HokieCFA. It is related to the "Student" entity with an "Attends" relationship, which represents the action of a student attending a career fair. Additionally, it is related to the "Employer" entity with an "Attends" relationship, which represents the action of an employer attending a career fair.

- **Employer**
    - This entity represents an employer attending the career fair. It has two attributes, "profileInfo" and "queueStatus", which represent the employer's personal information and their current status in the queue, respectively. The entity is related to the "Career Fair" entity with an "Attends" relationship, which represents the action of an employer attending a career fair.

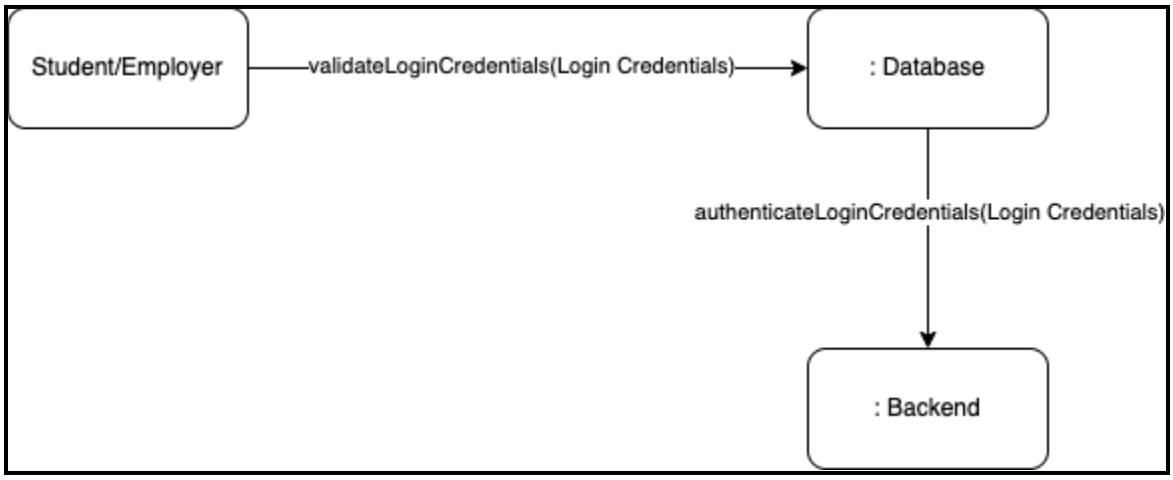
- **Queue**
    - This entity represents a queue for a specific employer. It has two attributes, "studentList" and "employerList", which represent the list of students and employers in the queue, respectively. The entity is related to the "Student" and "Employer" entities with a "Joins" relationship, which represents the action of a

student joining an employer's queue or an employer enlisting a student in their queue.

- **QR Code**
    - This entity represents the QR code associated with a specific employer's queue. It is related to the "Student" and "Employer" entities with a "Views" relationship, which represents the action of a student or an employer viewing the QR code.

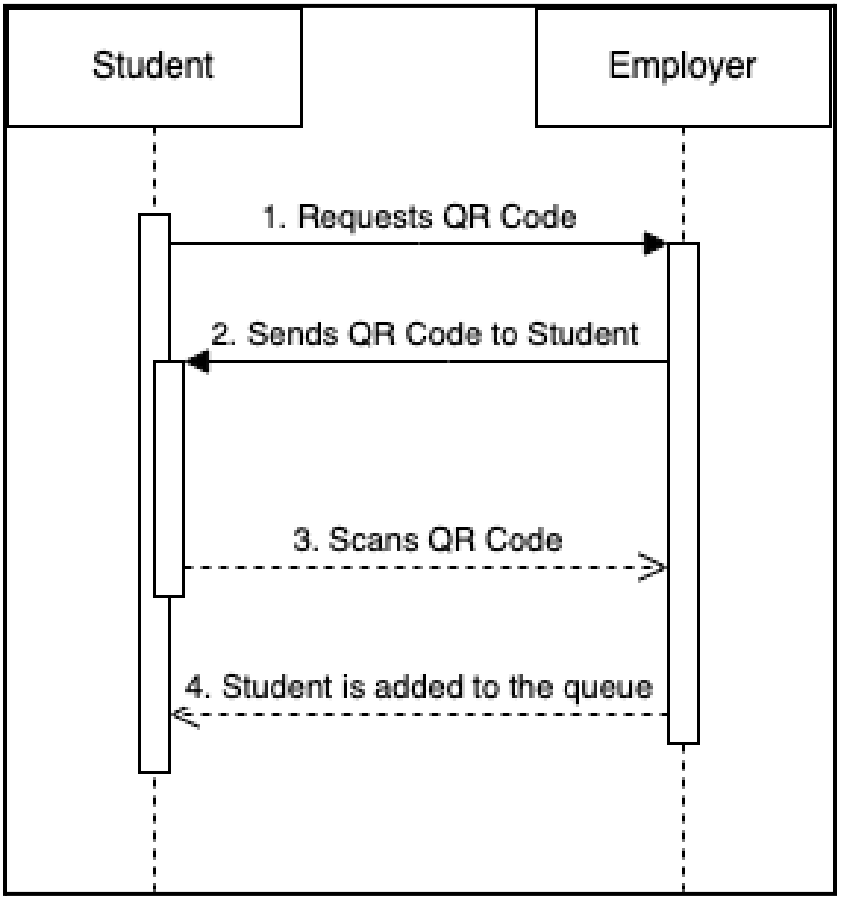
The domain model diagram represents the relationships between different entities involved in the in-person career fair process, such as students, employers, resumes, and queues. The diagram provides a visual representation of the relationships between these entities and their attributes, which can help in understanding and designing the system for managing career fairs.

3.2 Collaboration Diagram - Login



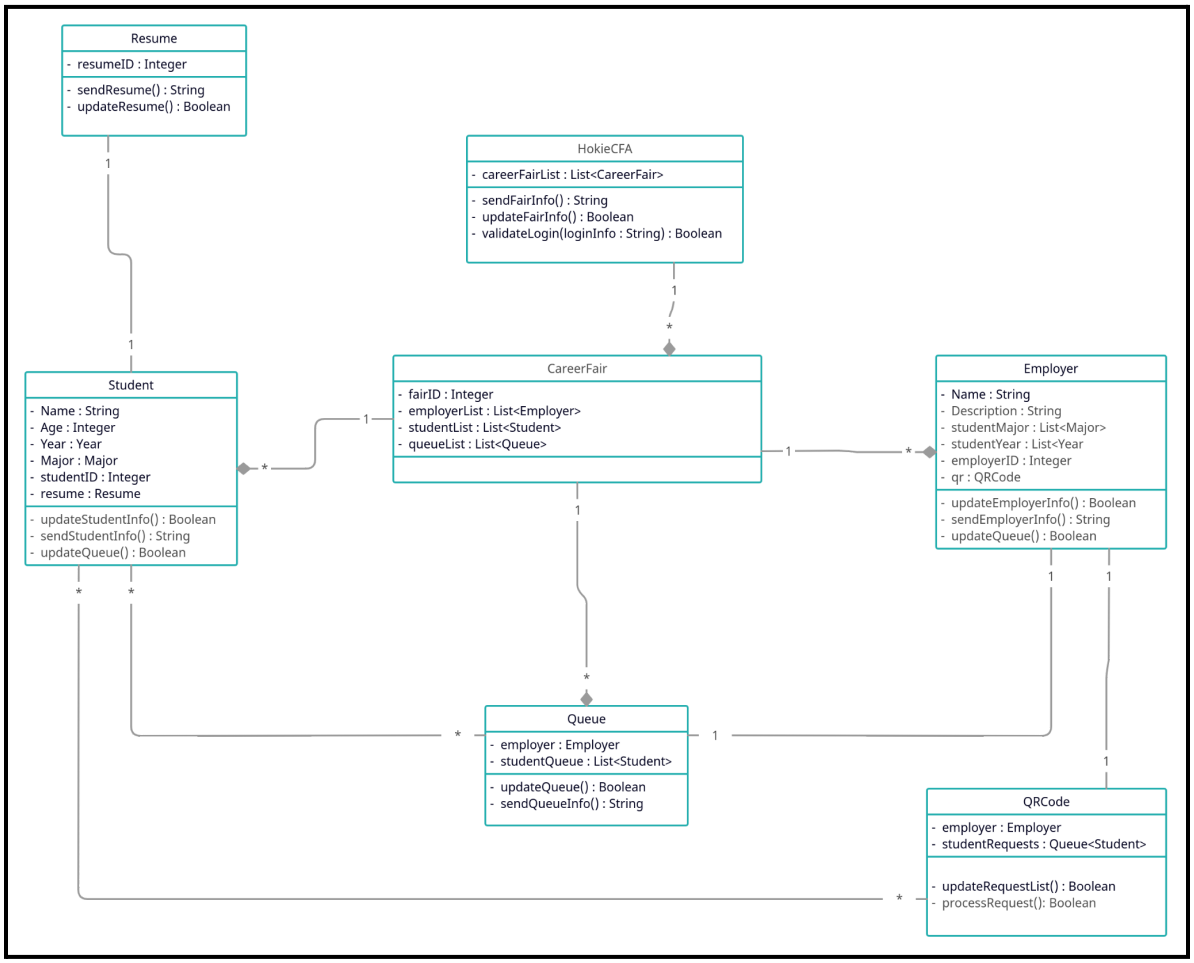
The “Student/Employer” objects are represented here as the primary actor in this use case. They will attempt to log in and will pass their login credentials to the HokieCFA database, where the back-end will authenticate the credentials. If they are correct, they will log in successfully; if they are incorrect or there are some other errors, the student/employer will be presented with an error message.

3.3 Sequence Diagram - Scan QR Code



In this sequence diagram, the primary actor is represented as "Student" and the secondary actor is represented as "Employer." The sequence of interactions starts with the student requesting the QR code (Step 1), followed by the employer sending the QR code to the student (Step 2). The student then scans the QR code (Step 3), and the main success scenario corresponds to the student being able to join the queue (Step 4). The sequence diagram shows a simple interaction between the student and the employer for this use case.

3.4 Design Class Diagram



Aforementioned is the class diagram for the HokieCFA mobile app. The design class diagram represents a system for managing career fairs that involves several classes representing different entities involved in the career fair process, such as students, employers, and career fair organizers. These classes have various attributes and methods that allow them to interact with each other and manage the flow of information during a career fair.

- **The HokieCFA Class**
    - This class represents the organizers of the Career Fair. It has an attribute "careerFairList" of type List<CareerFair>, which represents a list of all the career fairs organized by HokieCFA. The class has three methods: "sendFairInfo()", which returns a string and represents the organizers sending information about a career fair to potential attendees, "updateFairInfo()", which returns a boolean and represents the organizers updating information about a career fair, and

"validateLogin()", which takes a string as input and returns a boolean, representing the organizers validating a login attempt.

- **The CareerFair Class**
    - This class represents a specific career fair organized by HokieCFA. It has an attribute "fairID" of type Integer, which uniquely identifies each career fair. Additionally, it has three lists: "employerList" of type List<Employer>, which represents a list of all the employers attending the career fair, "studentList" of type List<Student>, which represents a list of all the students attending the career fair, and "queueList" of type List<Queue>, which represents a list of all the queues for the career fair. The class has no methods.

- **The Student Class**
    - This class represents a student attending the career fair. It has several attributes, including "age" and "year" of type Integer, which represent the student's age and academic year. Additionally, it has an attribute "major" of type Major, which represents the student's area of study. The class also has an attribute "studentID" of type Integer, which uniquely identifies each student. Finally, the class has an attribute "resume" of type Resume, which represents the student's resume. The class has three methods: "updateStudentInfo()", which returns a boolean and represents the student updating their information, "sendStudentInfo()", which returns a string and represents the student sending their information to potential employers, and "updateQueue()", which returns a boolean and represents the student updating their position in the queue for a career fair.

- **The Employer Class**
    - This class represents an employer attending the career fair. It has an attribute "employerID" of type Integer, which uniquely identifies each employer. Additionally, it has a list "industry" of type Industry, which represents the industry the employer belongs to. The class has three methods: "updateEmployerInfo()", which returns a boolean and represents the employer updating their information, "sendEmployerInfo()", which returns a string and represents the employer sending their information to potential students, and "updateQueue()", which returns a boolean and represents the employer updating their position in the queue for the career fair.

- **The Queue Class**
    - This class represents a queue for a career fair. It has several attributes, including "name" and "description" of type String, which represent the name and description of the queue. Additionally, it has two lists, "studentMajor" and "studentYear" of type List<Major> and List<Year>, respectively, which represent the major and academic year of the students in the queue. The class also has an attribute "employerID" of type Integer, which represents the employer associated with the queue. Finally, the class has an attribute "qr" of type QRCode, which represents the QR code associated with the queue. The class has two methods: "updateQueue()", which returns a boolean and represents the queue updating its position, and "sendQueueInfo()", which returns a string and represents the queue sending its information to potential employers.

- **The QRCode Class**
    - This class represents the QR code associated with a queue. It has an attribute "employer" of type Employer, which represents the employer associated with the QR code. Additionally, it has a queue of type Student, which represents the students who have requested to join the queue. The class has a single method "updateRequestList()", which returns a boolean and represents the QR code updating its request list.

- **The Resume Class**
    - This class represents the resume of a student attending the career fair. It has a single attribute, "resumeID" of type Integer, which uniquely identifies each student's resume. The class has two methods: "sendResume()", which returns a string and represents the student sending their resume to potential employers, and "updateResume()", which returns a boolean and represents the student updating their resume.

3.5 Our Algorithm

In order to keep track of the students in each of the queues and the position, we implemented a sorting algorithm, which sorted the students in the queue by priority values. Those students with the highest priority would be at the front of the queue, while those with the lowest priority would be at the back of the queue. Whenever a student or an employer attempted to gather information about the queue, we updated the priority values for each student in the queue using our priority function, and then sorted the queue list from highest to lowest priority. We optimized our sort using a B-tree index. This technique was chosen to avoid having to keep track

of and update the priority value for each individually, which would have been much more complex and potentially slow.

The function used to calculate priority value is the Gompertz curve $y = ae^{-be^{-ct}}$. This function is a class of sigmoid function, where the growth rate is slowest at both the beginning and the end of the curve. The value *t* represents time in this case, which is the number of seconds since the student joined the queue. The value *a* represents the maximum value which is set to 10, and *b* shifts the function rightward by a value 10, so 0 seconds nearly represents 0 priority. The value *c* is updated dynamically, and is based on the number queues that the student is in. As a student joins more queues, the c value falls, causing that student's priority value for that queue to increase more slowly. We chose this design to try and prevent students who may want to exploit the queueing system by showing up to the career fair much later and joining a dozen or more queues. However, this slowdown is not extreme. If a student joins 10 queues, their slowdown will be around 50% for each queue, and this can be easily adjusted to improve results in a real career fair.

3.6 Tools Adopted

- Android Studio
    - Android Studio was a crucial part of our project, because we utilized its Android emulator to run our app, in addition to using Expo Go on an iPhone.
- Flask
    - Because this was only a semester-long project, we wanted to use a software framework that would allow us to build our RESTful API backend quickly and efficiently. Flask is a micro-framework for Python which provides the basic functionality necessary for implementing a Web API, and can be easily extended for any backend programming application. In our case, we used multiple Flask extensions for implementing the backend, including MongoDB database management, password encryption, and RESTful routing services.
- JavaScript
    - For developing the frontend of our application, we had to use a variety of different javascript functions and variables. Javascript integrated with React Native, so we were easily able to pass the results of fetch calls to the backend to different components used by React Native to display our application.
- JSON
    - For our project, we passed data between the frontend and backend using the JSON data format. This is a fairly standard data format used for Web APIs, and can be easily parsed in both javascript and Flask.
- MongoDB

    - MongoDB was our database of choice for this project, because it works very well for real-time applications, which is exactly what HokieCFA aims to be. MongoDB is also very flexible, providing a large number of different query functions, and can be structured to work with many different application designs.
- Neovim
    - Neovim is a modern extension of the text editor Vim. For backend programming, Neovim is a great tool for using the command line while programming, which greatly simplifies development and testing of backend services.
- npm
    - npm is the package manager that we used to keep track of the packages that were needed for this app.
- OpenSSL
    - OpenSSL is an open-source implementation of the SSL that allows developers to quickly create self-signed certificates that work with HTTPS. While this is not secure for an enterprise application, it is very useful for testing an application's functionality and security in a local environment without having to buy a certificate from a domain provider.
- Python
    - Python is a general-purpose programming language which can be used to easily build a variety of different applications. For this project, we used Python to design and implement our backend system, and tested it using a mock database built with Pytest.
- React Native
    - React Native is a popular frontend library for developing UI for mobile applications. It works for both iOS and Android applications, which was important for working with the Android Emulator. Our team did not have much experience with mobile development, but we did have experience with React for web applications, so React Native was a good starting point for mobile development.
- Visual Studio Code
    - For the frontend development, Visual Studio Code was our IDE of choice because of its ability to run multiple terminals next to the code view and also because of the extensions that it offers that were helpful for programming a React Native app.

3.7 The Uniqueness of Our Product

- Our app incorporates a real-time notification system, which means that the user won't have to wait in just one queue, which is a big advantage over Brazen.
- Our proprietary queueing algorithm allows students to join multiple queues simultaneously, which greatly improves the experience at career fairs.

Chapter 4 - Retrospection

4.1 What went well?

- Collaboration and communication: One of the key factors that contributed to the success of the project was the team's collaboration and communication. We maintained open communication throughout the project, which allowed us to share our ideas and resolve issues quickly.
- Technology stack: We chose a good technology stack for our project. The stack consisted of VS Code, Node.js, MongoDB, Neovim, Python, React Native, Android Studio, Flask, OpenSSL, NPM, JavaScript, and JSON. These technologies worked well together and allowed us to develop a robust and efficient product.
- Testing: We implemented testing early in the development process, which helped us to identify and fix issues before they became major problems. We used unit testing, integration testing, and end-to-end testing to ensure the quality of our product.

4.2 What went wrong?

- Compatibility issues: Integrating different technologies into our tech stack led to compatibility issues, especially with our database management system, MongoDB. We spent a lot of time resolving these issues.
- User experience: We initially designed the product without taking into consideration the user's perspective, resulting in a complicated and confusing interface. This led to negative feedback during the beta testing phase, and we had to spend additional time redesigning the product to make it more user-friendly.
- Scope creep: We added too many features to the product, which resulted in scope creep. This led to delays in the project timeline and made it difficult to manage the project effectively.

4.3 What have we learned?

- Improved technical skills: Working on this project allowed us to gain more experience in using various technologies such as Node.js, MongoDB, and React Native. We also learned new skills, such as implementing efficient testing strategies, integrating different technologies into a tech stack, and creating user-friendly designs.
- Improved communication and collaboration skills: We learned the importance of efficient communication and collaboration within a team. This project provided us with

the opportunity to work on our communication and collaboration skills, which are essential for successful project management.

- Understanding user needs: We learned the importance of considering the user's perspective in product design. We conducted user research and testing to ensure that the product meets the user's needs and is user-friendly.

4.4 What should have been done to avoid the problems encountered?

- Planning the integration of different technologies: We should have taken more time to research and plan the integration of different technologies into our tech stack. This would have allowed us to identify potential compatibility issues and mitigate them beforehand.
- Managing scope effectively: We should have managed the project scope effectively by prioritizing features and adding them incrementally. This would have prevented scope creep and made it easier to manage the project timeline.

4.5 How could our product get better?

- Personalized recommendations: We could incorporate personalized recommendations to enhance the user experience. By analyzing user data and preferences, we could recommend jobs and employers that would be a good match for the user.
- Implementing a search functionality: We could provide a search functionality in-app through advanced search algorithms. This would allow users to find jobs and employers that meet their specific needs and preferences.
- Machine learning algorithms: We could integrate machine learning algorithms into the product to make it more intelligent and intuitive. By analyzing user data, we could provide personalized recommendations, job matching, and other features that improve the user experience.
- Scalability and performance: We could work on improving the product's scalability and performance, making it easier to handle a large number of users and data. This could involve optimizing the product's code, database management, and server infrastructure.
- Integrating with other platforms: We could consider integrating the product with other platforms such as LinkedIn, Indeed, and Glassdoor. This would allow users to import their profiles and job search history, making it easier to find relevant job opportunities.
- Social media integration: We could integrate social media platforms, such as LinkedIn, Twitter, and Facebook, to provide users with more networking opportunities. This could allow users to connect with employers and other job seekers, share job opportunities, and receive job alerts through social media channels.

- Improved user interface: We could continue to improve the user interface of the product by incorporating user feedback and conducting user testing. This could involve redesigning the product to make it more intuitive, user-friendly, and aesthetically pleasing.

Overall, HokieCFA has the potential to become a powerful tool for job seekers and employers alike. By incorporating new features and addressing existing issues, we can continue to improve the product and provide users with a seamless and effective job search experience. As a team, we first want to revolutionize the career fairs at Virginia Tech. We envision that HokieCFA would ultimately be used at all colleges, universities, and high schools all across the United States!