Software Engineering Software Requirements Specification (SRS) Document

JobGoblin

2/22/2023

Version 2

By: Eli Stafford, Jonathan Pelaez, Maximilian Honeycutt

ES, JP, MH

Table of Contents

1. Introduction

1.1. Purpose

JobGoblin is the ultimate pseudo social media app for anyone involved in the job market. We aim to consolidate the tediousness of applications, finding jobs, or recruiting new employees. With JobGoblin companies can advertise themselves, have recruiters review and schedule interviews in less than 5 clicks, and instantly view all applicants. Job seekers can view a large list of jobs suited for them, easily apply in a few clicks, get quick responses, and save important information to one public page. It's easy to market yourself, or your business. And you're just a few clicks away from a new job! Get responses, get interviews, get jobs. JobGoblin.

1.2. Document Conventions

Client views will display different interfaces depending on the user. Whether a company who manages their recruiters, a recruiter who schedules interviews and reports to the company, and the applicant who can see listings and apply.

1.3. Definitions, Acronyms, and Abbreviations

[Include any specialized terminology dictated by the application area or the product area.
For example:]

Java	A programming language originally developed by James Gosling at Sun Microsystems. We will be using this language to build the Restaurant Manager.
MySQL	Open-source relational database management system.
.HTML	Hypertext Markup Language. This is the code that will be used to structure and design the web application and its content.
SpringBoot	An open-source Java-based framework used to create a micro Service. This will be used to create and run our application.
MVC	Model-View-Controller. This is the architectural pattern that will be used to implement our system.
Spring Web	Will be used to build our web application by using Spring MVC. This is one of the dependencies of our system.
Thymeleaf	A modern server-side Java template engine for our web environment. This is one of the dependencies of our system.
NetBeans	An integrated development environment (IDE) for Java. This is where our system will be created.
API	Application Programming Interface. This will be used to implement a function within the software where the current date and time is displayed on the homepage.
GSON	Gson library for converting data to JSON files. This will be used to help save java objects to our database without messy conversions.

Google API	Google API is an API created by google to feed data in and out of google applications such as Forms, Docs, Excel and more. This will be used to pull data from custom user created forms at ease.

1.4. Intended Audience

Broad audience: Any companies or individuals looking to hire or apply for a job
In-depth look: Companies looking for fast and easy recruitment. Anyone looking to consolidate their virtual resume to one app.

Companies: View sections 2, 5 and 6. This will explain how the software was designed and its intended way to be used, how to interface through the software as well as a broad description of the entire application.

Applicant investors: View sections 2, 6 and 7. Which will again give a broad and in-depth overview of the functionality of the software, as well as show a use example.

Current Stakeholders, developers and project managers: Eli, Jon, Max

1.5. Project Scope

Our goal is to create an interface that recruiters, applicants and company managers alike can interact with other users, access data quickly, and get quick responses to applicants. Companies can easily list job requirements, positions to be filled and the amount. This allows quick and easy applications from applicants, even quicker with our "Quick Submit" feature.

-Companies can give applicants a quick form to gauge whether the applicant is acceptable before interviewing.
Thus, making the hiring process much quicker, as you can deny applicants much earlier on. This also makes qualified applicants apply to hire time much shorter, meaning all hands on deck instantly.
-Applicants get quicker responses, which means less wait time, more opportunity seeking.
-Applicants can can make more educated decisions on applications, making their apply to success ratio higher

1.6. Technology Challenges

Learning how to pull data from google forms -> JSON -> Java readable object. Using the GSON library will help from JSON -> Java readable objects, but pulling the data from the google form itself is foreign territory.

1.7. References

No references

2. General Description

2.1. Product Perspective

JobGoblin origins lie in the desire to make the job searching experience less tedious. Many job searching websites offer intermediary services for job seekers and companies but none offer a service that handles all interactions. Handshake for example provides overviews of companies and their recruiter and allows you to contact the recruiter. Our plan is to expand on this by providing job seekers the ability to apply for jobs directly from our site and also create meaningful connections through more open interactions with prospective companies.

2.2. Product Features

The product features include the ability to create a job seeker account or company profile to show off talents and opportunities. Job seekers will be able to search for jobs by a search engine using types of identifiers like company names, positions, salary and others to locate the job that fits their interests. Companies will be able to create or delete recruiters that are able to interact with the job seekers directly via chat or email and also be able to post job openings for internships, part-time and full-time positions. This product will also feature administrators which will be able to delete companies or job seekers that are attempting to start illicit businesses or harass other users of the product.

2.3. User Class and Characteristics

Our product does not expect the user to be the most technically literate person, apart from using a web browser and representing themselves professionally on the internet. Our product removes all of the different websites that one would otherwise use to apply to multiple jobs by becoming a hub for job related interactions.

2.4. Operating Environment

The application is designed to operate on the web across many different devices. Except the switch lol.

2.5. Constraints

[Any limiting factors that would pose challenge to the development of the software. These include both design as well as implementation constraints.]
Efficiency/data storage on scale
Most likely inefficient search/show algorithms

2.6. Assumptions and Dependencies

[A list of all assumptions that you have made regarding the software product and the environment along with any external dependencies which may affect the project]

The software will be dependent on Spring Web and Thymeleaf in order to create and execute the MVC architecture that will be developed within NetBeans. The application will also use the Google API that will allow us to connect and interact with google services like docs, slides, etc.

3. Functional Requirements

3.1. Primary

Recruiters:

-Can view applicant submitted form
-Can view all applications for recruiters assigned position (by company)
-Can view applicants application history
-Can schedule interviews with the applicant
-Can view assigned positions
-Can request approval for applicants

Applicant:

- Can view a list of all jobs
  - Lists positions available, description, location
- Can query list of jobs for more specified positions
- Can apply to a job (Fill out and submit form)
- Can view company pages??
- Can view their pages
- Can upload rudimentary personal information
- Optional PDF upload for full resume??

Company:

- Can assign recruiters to available positions
- Can view/approve pending applicants (Offer letter)
- Can view approved applicants
- Can open new positions
  - Positions available (number), location, requirements, description
- Can view recruiter interviews
- Can view position applications

3.2. Secondary

- Password protection for information only accessible to employees, managers, and each individual table.
- Authorization scheme so that companies can only alter and see their positions available and no other companies' positions.

4. Technical Requirements

4.1. Operating System and Compatibility

The application will be compatible with any operating system that is able to view and to interact with traditional web pages.

4.2. Interface Requirements

4.2.1. User Interfaces

See #6.

4.2.2. Hardware Interfaces

The web application will run on any hardware device that has access to the internet, the ability to display webpages, and the ability to interact with web pages. This includes, but is not limited to, smartphones, tablets, desktop computers, and laptops. Unfortunately it can't run on the switch and support won't be coming any time soon either lol.

4.2.3. Communications Interfaces

Server must be able to connect to the internet as well as the local database on phpMyAdmin. The communication protocol, HTTP, must be able to connect to the Google API and return data to and from the web application to google services like docs, slides, etc.
User must be able to connect to the internet via a web browser.

4.2.4. Software Interfaces

We will use React and Spring Boot ThymeLeaf to help build the frontend, as well as JPA for the backend database functionality. We will also use Spring Boot with Java to connect the frontend to the backend.

No standalone software interface for clients.

5. Non-Functional Requirements

5.1. Performance Requirements

Use of the program will require local memory less than or equal to the memory required to run a webapp on a modern web browser.

5.2. Safety Requirements

- Personal, vulgar, and hateful information should not be shared, as this is a publicly accessible professional application. These terms are included in the terms agreed to on account creation, and violation merits account deletion.
- Ratings of individual persons will not be included, only on companies/jobs.
- No interaction will occur directly with client machines because this is a webapp.

5.3. Security Requirements

Personal/private data entered will only be shared with companies on application, and will not be available publicly for other users. Personal data will not be accessible in any way on user machines, save for companies.

5.4. Software Quality Attributes

5.4.1. Availability

Hosted 24/7 on jobgoblin servers. Maintenance will be forecasted a week in advance when possible. Emergency maintenance may occur, and all users will be notified.

5.4.2. Correctness

Data shown on the website is managed by each individual user. Jobgoblin does not control or moderate any false/misleading information posted by users. Companies are vetted before being given access to jobgoblin, and are expected to manage their recruiters. Jobseekers can be banned without notice of infraction.

5.4.3. Maintainability

Very little complex data structures/algorithms/dependencies which would merit maintenance. Barebones application.

5.4.4. Reusability

Servers are run by jobgoblin, so users will not have access to jobgoblin after permanent server shutdown.

5.4.5. Portability

Runs on any web browser.

5.5. Process Requirements

5.5.1. Development Process Used

Agile process model used. Beginning with data structures/architecture, then building out U.I.

5.5.2. Time Constraints

This course.

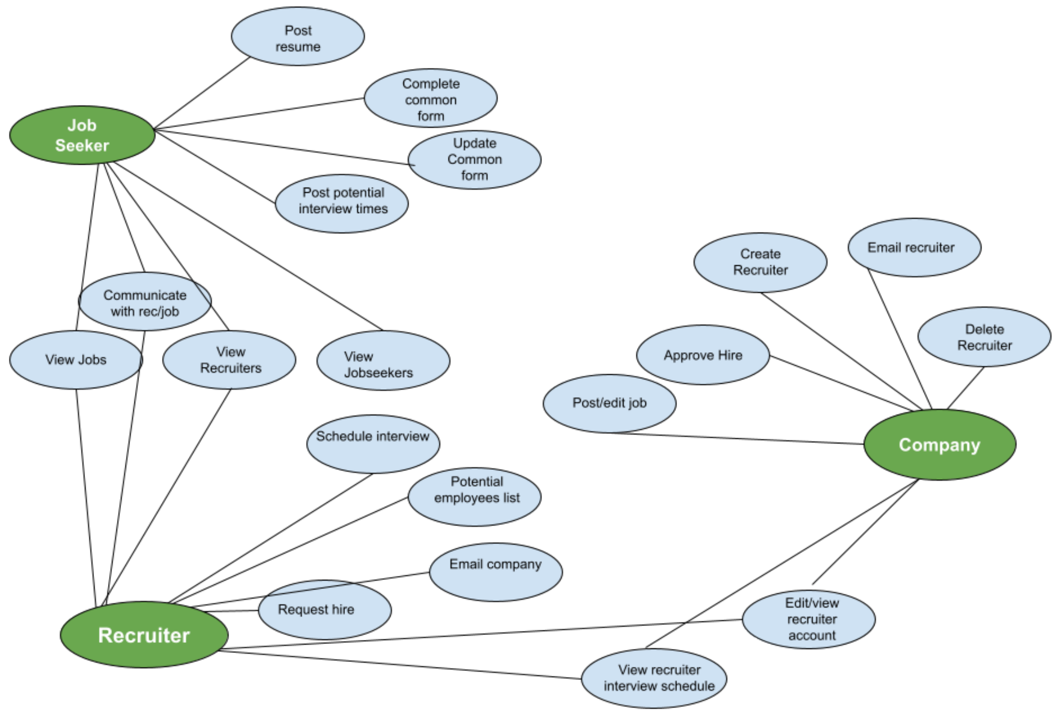
5.5.3. Cost and Delivery Date

$0, delivery on end of course.

5.6. Other Requirements

TBD

5.7. Use-Case Model Diagram



5.8. Use-Case Model Descriptions

5.8.1. Actor: Jobseeker (Max)

- **View Jobs**: Search/view/interact with job postings
- **View Recruiters:** Search/view/interact with recruiters
- **Communicate w/ Rec/Job**: Access email/other contact for jobs/recruiters

- **Interview Times:** Edit/add interview times for interested recruiters to view.
- **Update/Fill Common Form:** Fill/edit common application sent with all job applications.
- **Post Resume**: Add resume to account.
- **Edit account:** Edit personal/public information.

5.8.2. Actor: Recruiter (Eli)

- **View Jobs**: Search/view/interact with job postings
- **View Jobseekers:** Search/view/interact with recruiters
- **Post/Edit Job:** Create/edit job postings.
- **Schedule Interview:** Send interview request to jobseekers at specified time.
- **Potential Employees:** View employees that have 1) been favorited or 2) have an interview/have been interviewed.
- **Email Company:** Company information/communication.
- **Request Hire:** Send a hire request to the company with an attached employee.
- **Edit account:** Edit personal/public information.
- **View Recruiter Schedule:** View scheduled interviews/push to google calendar.

5.8.3. Actor: Company (Jon)

- **Create Recruiter**: Create/delete recruiters accounts.
- **Recruiter List:** View all active recruiters with company, as well as emails.
- **Edit account:** Edit company/public information.
- **Approve Hire (offer):** Approve hire request, sending info to employee.
- **Edit recruiter account:** Edit personal/public information of a recruiter.

5.9. Use-Case Model Scenarios

5.9.1. Actor: Jobseeker (Max)

- **View Jobs/Recruiters**:
  - **Initial Assumption**: Logged in
  - **Normal**: Show jobs
  - **What Can Go Wrong:** Jobs must reach the client’s machine.
  - **Other Activities**: Interact with jobs, applying/viewing email contact.
  - **System State on Completion**: View job/recruiter
- **View/Edit Account**:
  - **Initial Assumption**: Logged in
  - **Normal**: Edit account info
  - **What Can Go Wrong**: Invalid account info entered (email, birthday, etc)
  - **Other Activities**: Save account, view calendar, view applied jobs, view offers
  - **System State on Completion**: Updated user account

5.9.2. Actor: Recruiter (Eli)

- **Use-Case Name**: View Employees
  - **Initial Assumption**: Logged in
  - **Normal**: View employees
  - **What Can Go Wrong**: Employers must reach client machine
  - **Other Activities**: Interact with employees
  - **System State on Completion**: Change screen
- **Use-Case Name**: View Employee

- **Initial Assumption**: Chose employee from view employees
- **Normal**: View employee
- **What Can Go Wrong**:
- **Other Activities**: Get email, schedule interview
- **System State on Completion**: Possible information sent to jobseeker

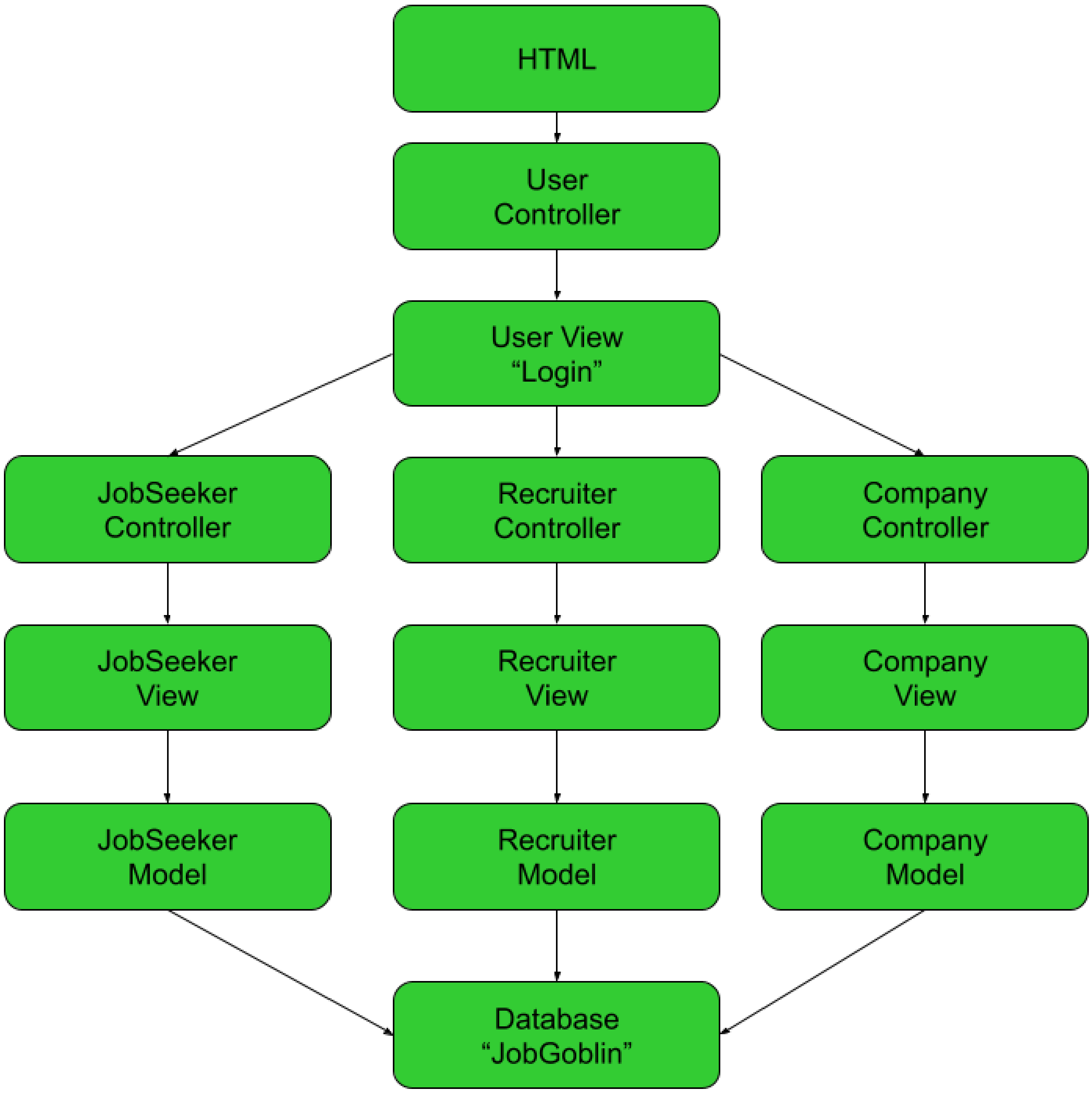
- **Edit Account**:
  - **Initial Assumption**: Logged in
  - **Normal**: Edit account info
  - **What Can Go Wrong**: Invalid account info entered (email, birthday, etc)
  - **Other Activities**: Save account, view assigned jobs
  - **System State on Completion**: Updated user account

5.9.3. Actor: Company (Jon)

- **Create/Edit Recruiters**:
  - **Initial Assumption**: Click recruiters
  - **Normal**: View recruiters
  - **What Can Go Wrong**:
  - **Other Activities**: View calendars, edit recruiter, add new recruiter, assign position
  - **System State on Completion**: Update recruiters
- **Edit Account**:
  - **Initial Assumption**: Logged in
  - **Normal**: Edit account info
  - **What Can Go Wrong**: Invalid account info entered (email, desc, etc)
  - **Other Activities**: Save account
  - **System State on Completion**: Updated user account
- **Create/Edit Position:**
  - **Initial Assumption**: Click “jobs”
  - **Normal**: View jobs
  - **What Can Go Wrong**: Invalid account info entered (email, birthday, etc)
  - **Other Activities**: Create job, edit job
  - **System State on Completion**: Updated job list

6. Design Documents

6.1. Software Architecture-Max



6.2. High-Level Database Schema-Eli

Terminology notes:

Disc = discriminator, i.e job is identified via companyid, and discriminator. Two jobs with different companyids can have the same discriminator, but two jobs with the same companyid cannot

General Domains:

id's : long

Name : 60 char

Descriptions : 'text' size, a little large, ~65,000 chars, may have to split into multiple tinytexts (255 chars)

Most other domains are between 60-100 chars

Underline: primary key(s) (identifying key)
Italics: foreign key (in other table)
Tables
Jobseeker(id, name, birthdate, address, desc)
- List of all jobseekers
- Ethnicity, disability, veteran are protected info sent to company on hire

Resume(*jobs_id*, resume_file)
- List of jobseeker resumes, stored separately for space consideration
- Store a single pdf resume file, with max size constraint to be determined

Jobseeker_job(*jobse_id, job_disc, companyid*, favorite, progress)
- All jobs connected to a jobseeker, which progress indicating, not-started, applied, hired, and favorite indicating this job is favorited by the jobseeker

Job(disc, *companyid*, *quizdisc*, name, desc, rating, ratingNum, dateposted, salarylow, salaryhigh)
- List of all jobs

job_recruiter(*jobid, recruiterid*)
- List of jobs connected to certain recruiter. Multiple recruiters can have multiple jobs, and vice versa

Quiz(*companyid*, quiz_disc, quiz file)
- List of company quizzes, i.e job specific applications

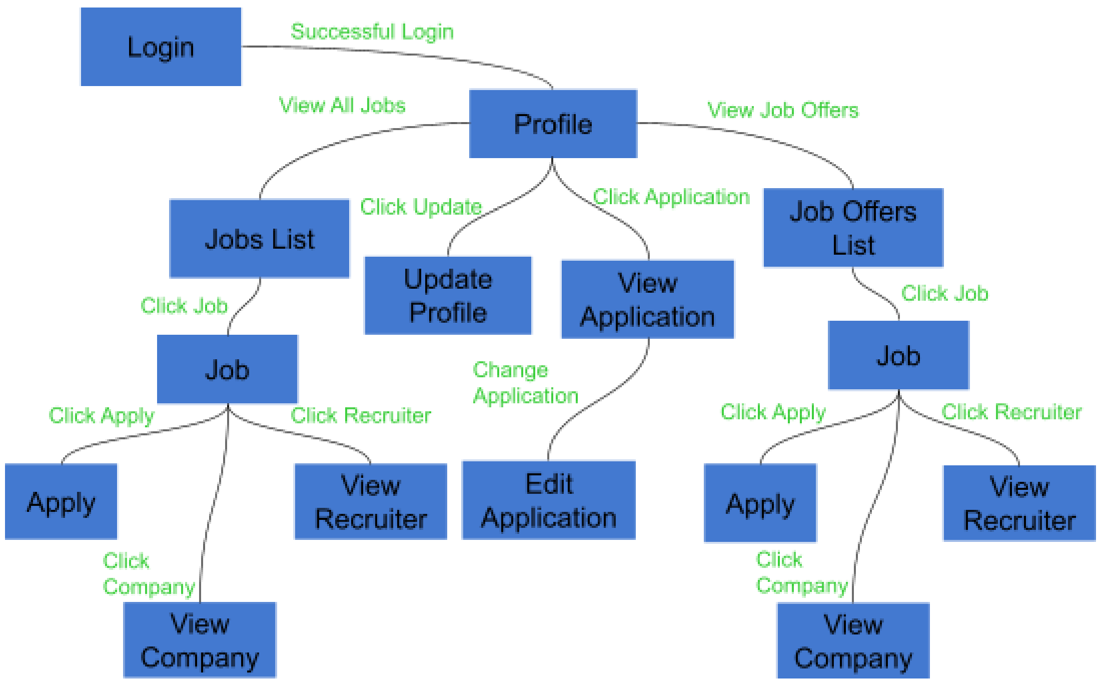
Company(*id*, name, desc, founddate, startdate, website, rating, numratings,)
- List of companies

Recruiter(*id*, rating, ratingnum, desc, email, start date)
- List of recruiters
- Companyid denotes an assigned company. Currently allow one company for each recruiter

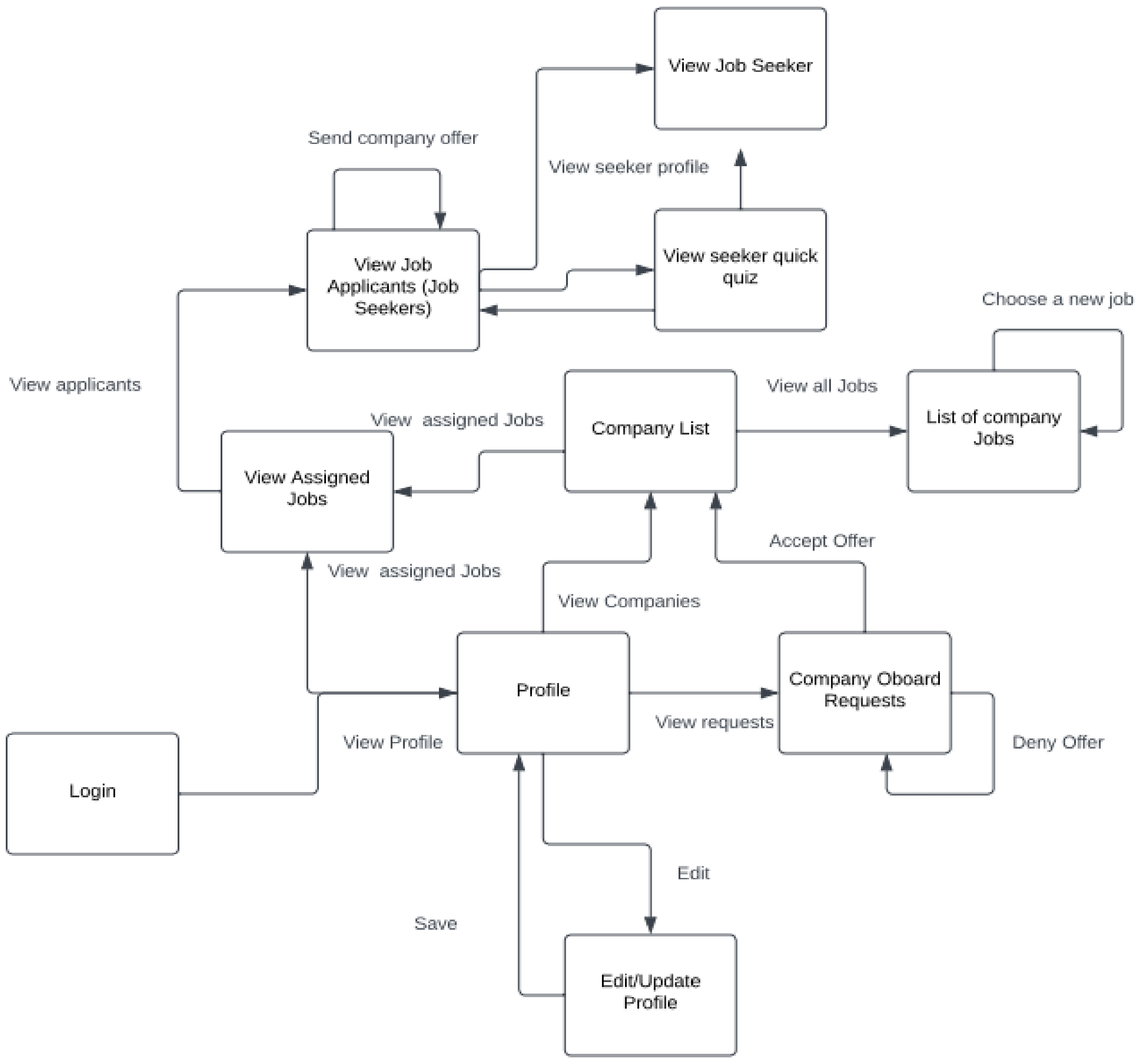
Company_Recruiter(*recruiter_id, company_id*)
- Denotes recruiter working for a specific company

6.3. Software Design

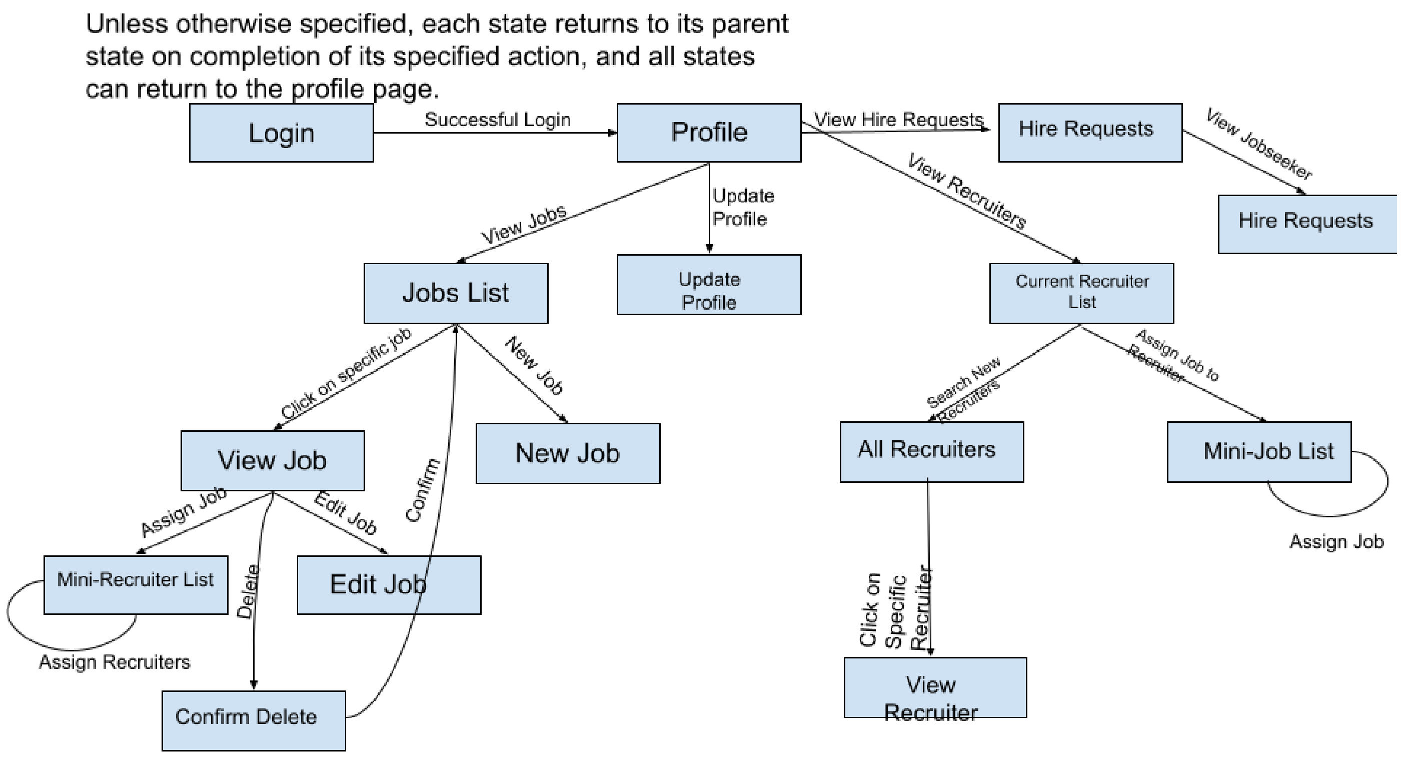
6.3.1. State Machine Diagram: Jobseeker (Max)



6.3.2. State Machine Diagram: Recruiter (jp)

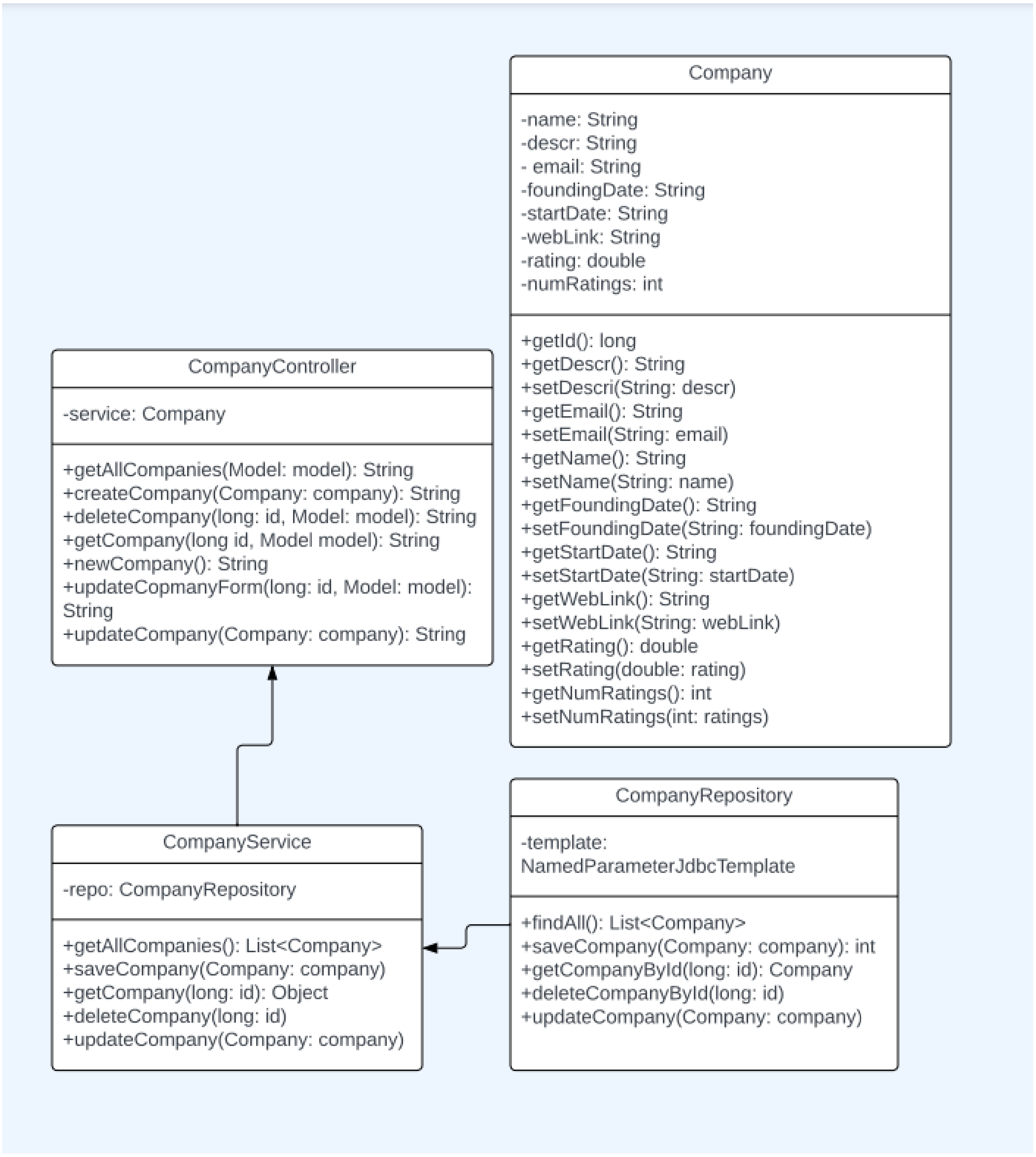


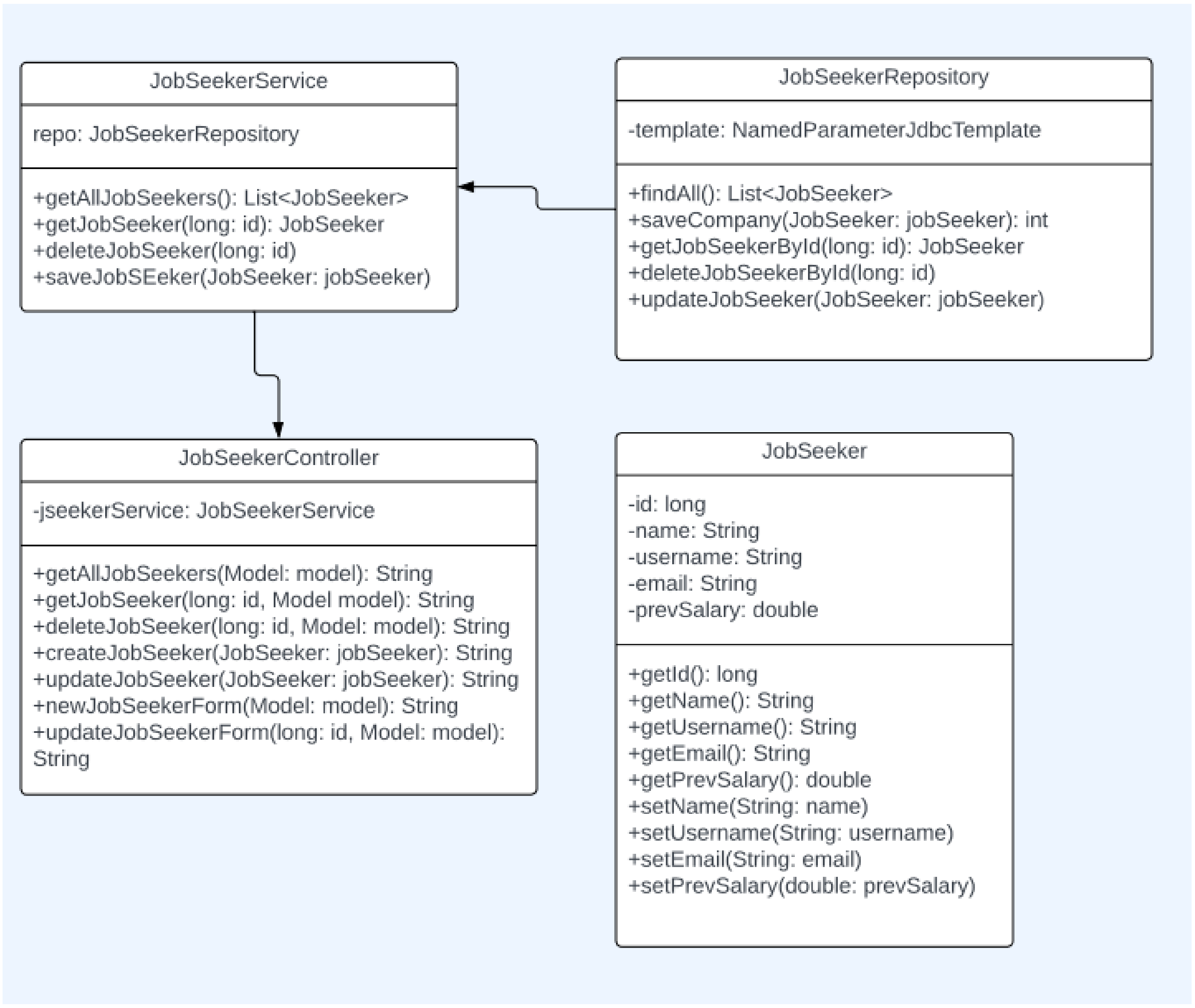
6.3.3. State Machine Diagram: Company (Eli)

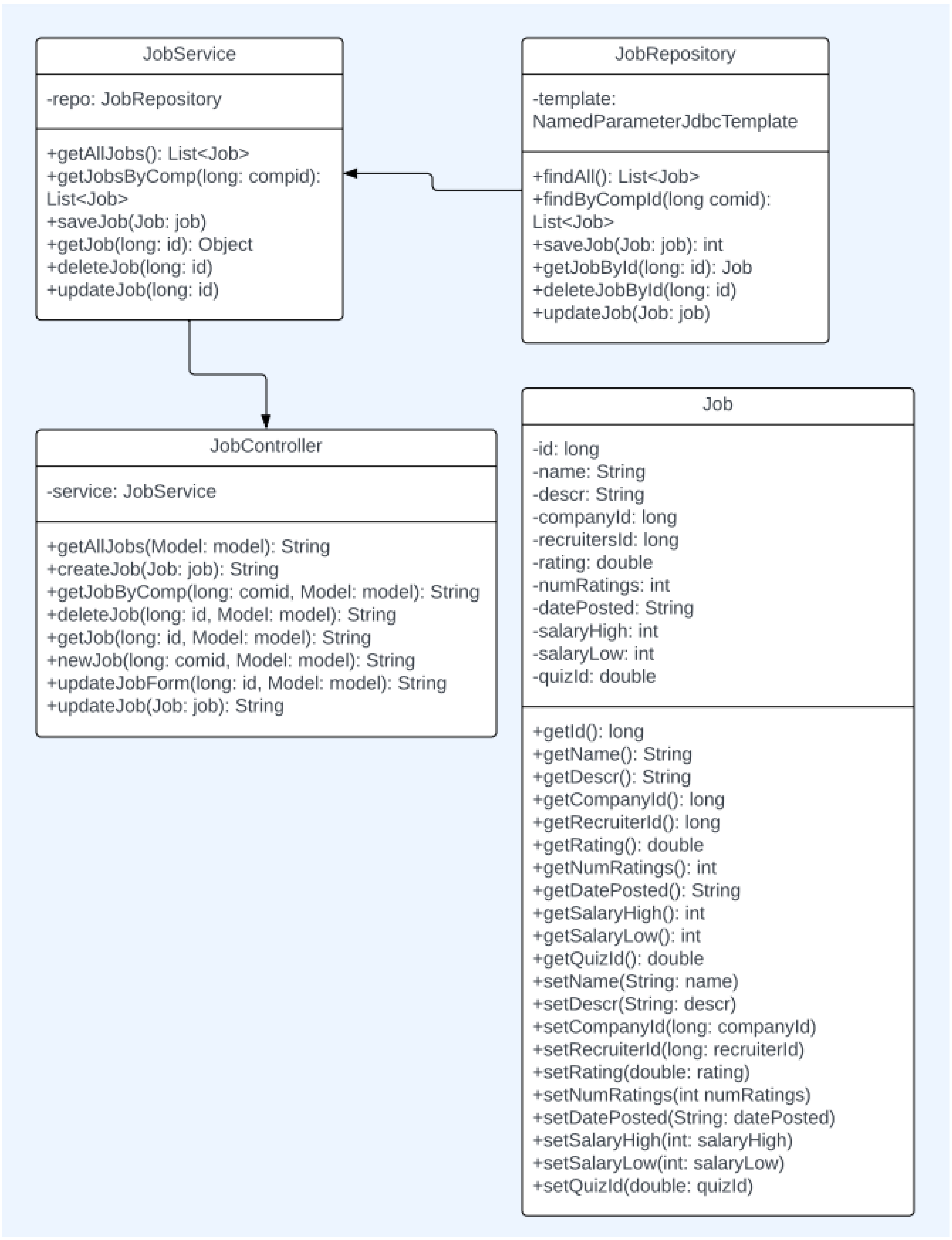

Unless otherwise specified, each state returns to its parent state on completion of its specified action, and all states can return to the profile page.

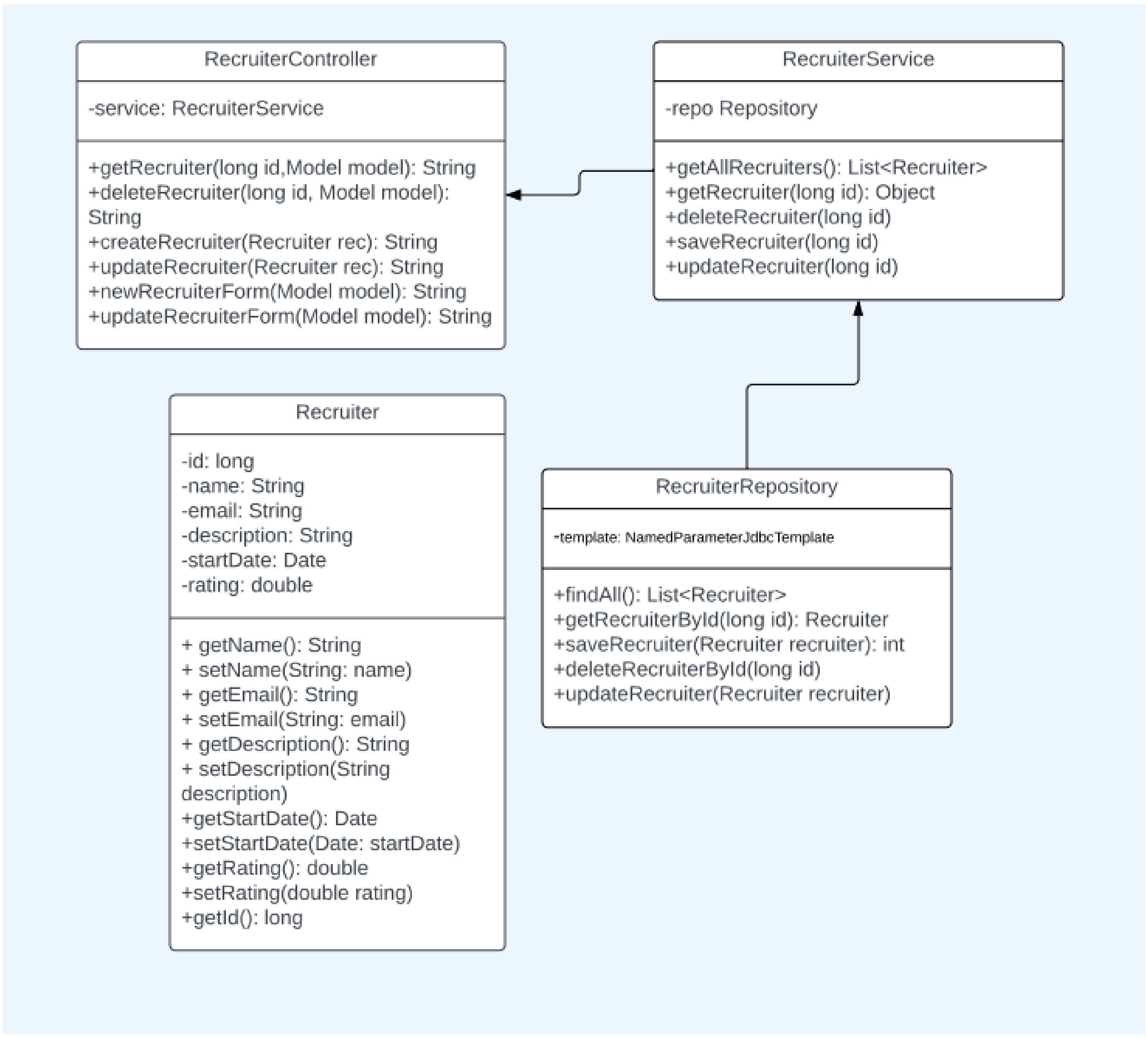
6.4. UML Class Diagram-Jon

(Had to chop up the pdf so it would fit)


Company
-name: String
-descr: String
- email: String
-foundingDate: String
-startDate: String
-webLink: String
-rating: double
-numRatings: int
+getId(): long
+getDescr(): String
+setDescri(String: descr)
+getEmail(): String
+setEmail(String: email)
+getName(): String
+setName(String: name)
+getFoundingDate(): String
+setFoundingDate(String: foundingDate)
+getStartDate(): String
+setStartDate(String: startDate)
+getWebLink(): String
+setWebLink(String: webLink)
+getRating(): double
+setRating(double: rating)
+getNumRatings(): int
+setNumRatings(int: ratings)
CompanyController
-service: Company
+getAllCompanies(Model: model): String
+createCompany(Company: company): String
+deleteCompany(long: id, Model: model): String
+getCompany(long id, Model model): String
+newCompany(): String
+updateCopmanyForm(long: id, Model: model): String
+updateCompany(Company: company): String
CompanyService
-repo: CompanyRepository
+getAllCompanies(): List<Company>
+saveCompany(Company: company)
+getCompany(long: id): Object
+deleteCompany(long: id)
+updateCompany(Company: company)
CompanyRepository
-template: NamedParameterJdbcTemplate
+findAll(): List<Company>
+saveCompany(Company: company): int
+getCompanyById(long: id): Company
+deleteCompanyById(long: id)
+updateCompany(Company: company)


JobSeekerService
repo: JobSeekerRepository
+getAllJobSeekers(): List<JobSeeker>
+getJobSeeker(long: id): JobSeeker
+deleteJobSeeker(long: id)
+saveJobSEeker(JobSeeker: jobSeeker)
JobSeekerRepository
-template: NamedParameterJdbcTemplate
+findAll(): List<JobSeeker>
+saveCompany(JobSeeker: jobSeeker): int
+getJobSeekerById(long: id): JobSeeker
+deleteJobSeekerById(long: id)
+updateJobSeeker(JobSeeker: jobSeeker)
JobSeekerController
-jseekerService: JobSeekerService
+getAllJobSeekers(Model: model): String
+getJobSeeker(long: id, Model model): String
+deleteJobSeeker(long: id, Model: model): String
+createJobSeeker(JobSeeker: jobSeeker): String
+updateJobSeeker(JobSeeker: jobSeeker): String
+newJobSeekerForm(Model: model): String
+updateJobSeekerForm(long: id, Model: model): String
JobSeeker
-id: long
-name: String
-username: String
-email: String
-prevSalary: double
+getId(): long
+getName(): String
+getUsername(): String
+getEmail(): String
+getPrevSalary(): double
+setName(String: name)
+setUsername(String: username)
+setEmail(String: email)
+setPrevSalary(double: prevSalary)


JobService
-repo: JobRepository
+getAllJobs(): List<Job>
+getJobsByComp(long: compid): List<Job>
+saveJob(Job: job)
+getJob(long: id): Object
+deleteJob(long: id)
+updateJob(long: id)
JobRepository
-template: NamedParameterJdbcTemplate
+findAll(): List<Job>
+findByCompId(long comid): List<Job>
+saveJob(Job: job): int
+getJobById(long: id): Job
+deleteJobById(long: id)
+updateJob(Job: job)
JobController
-service: JobService
+getAllJobs(Model: model): String
+createJob(Job: job): String
+getJobByComp(long: comid, Model: model): String
+deleteJob(long: id, Model: model): String
+getJob(long: id, Model: model): String
+newJob(long: comid, Model: model): String
+updateJobForm(long: id, Model: model): String
+updateJob(Job: job): String
Job
-id: long
-name: String
-descr: String
-companyId: long
-recruitersId: long
-rating: double
-numRatings: int
-datePosted: String
-salaryHigh: int
-salaryLow: int
-quizId: double
+getId(): long
+getName(): String
+getDescr(): String
+getCompanyId(): long
+getRecruiterId(): long
+getRating(): double
+getNumRatings(): int
+getDatePosted(): String
+getSalaryHigh(): int
+getSalaryLow(): int
+getQuizId(): double
+setName(String: name)
+setDescr(String: descr)
+setCompanyId(long: companyId)
+setRecruiterId(long: recruiterId)
+setRating(double: rating)
+setNumRatings(int numRatings)
+setDatePosted(String: datePosted)
+setSalaryHigh(int: salaryHigh)
+setSalaryLow(int: salaryLow)
+setQuizId(double: quizId)



RecruiterController
-service: RecruiterService
+getRecruiter(long id,Model model): String
+deleteRecruiter(long id, Model model): String
+createRecruiter(Recruiter rec): String
+updateRecruiter(Recruiter rec): String
+newRecruiterForm(Model model): String
+updateRecruiterForm(Model model): String
RecruiterService
-repo Repository
+getAllRecruiters(): List<Recruiter>
+getRecruiter(long id): Object
+deleteRecruiter(long id)
+saveRecruiter(long id)
+updateRecruiter(long id)
Recruiter
-id: long
-name: String
-email: String
-description: String
-startDate: Date
-rating: double
+ getName(): String
+ setName(String: name)
+ getEmail(): String
+ setEmail(String: email)
+ getDescription(): String
+ setDescription(String description)
+getStartDate(): Date
+setStartDate(Date: startDate)
+getRating(): double
+setRating(double rating)
+getId(): long
RecruiterRepository
-template: NamedParameterJdbcTemplate
+findAll(): List<Recruiter>
+getRecruiterById(long id): Recruiter
+saveRecruiter(Recruiter recruiter): int
+deleteRecruiterById(long id)
+updateRecruiter(Recruiter recruiter)

7. Scenario

7.1. Brief Written Scenario with Screenshots

Companies