CareerBound.org
Software Design Description

W13 Milestone 6: Final SDD

Class Section 01
Brother Linwood Jones
CSE 430-01, Architecture

1. Introduction - Front Matter

1.1. Date and Status

Dec 10, 2022
Milestone 6: Final SDD
Expected time until completion of design: The design is now complete

1.2. Scope

The system will connect employers to students; it does not guarantee employment to a student or the filling of an employment position to the employer. This SDD covers the design views necessary to create the system and the terms and visuals necessary to understand it.

1.3. Authorship

Project Sponsor: Linwood Jones
Project Manager: Abel Wenning
Team Leaders: Alexander Dohms, Ethan Nelson, Grant Holley, Samuel Casellas
Individual Contributors:

Aaron Burnside	Braxton Meyer	Michael Jackson
Alex Jacobs	Collette Stapley	Michael LeFevre
Andrew Morris	Colter Christensen	Nicholas Balabanov
Anita Woodford	Elijah Harrison	Preston Millward
Austin Earl	Jacob Elzinga	Shaun Crook
Benjamin Beales	Jacob Parker	Torin Bolander
Brad Howard	Jonathan Gunderson	Zack Pedersen
Bradley Payne	Leonardo Galani	
Brandon Wareing	Michael Fisher	

1.4. References

[1] Idaho, B. Y. U.-. (n.d.). *BYU-Idaho Login*. CAS – Central Authentication Service. Retrieved October 15, 2022, from https://content.byui.edu/ Career Bound Software Requirements Specification Document, https://content.byui.edu/file/968cdd33-09eb-4430-99a4-131d25e138d4/1/Teach/SRS/430.SRS.CareerBound.pdf

[2] https://cloud.google.com

[3] https://grafana.com

[4] https://kubernetes.io

[5] "What does work experience mean?" https://www.definitions.net/definition/work+experience

[6] "Message" Wikipedia, https://en.wikipedia.org/wiki/Message.

[7] "Strategy, Project Selection, and Portfolio Management – Technical ..." https://wisc.pb.unizin.org/technicalpm/chapter/project-selection/.

[8] https://brainly.com/question/17268113

[9] "Student" https://www.encyclopedia.com/social-sciences-and-law/education/education-terms-and-concepts/student.

1.5. Revision History

The initial component diagrams for each of the divisions of the CareerBound.org program have been expanded to include the first dozen specific views.

Name	Date	Reason	Version
Design Overview	10/1/2022	Original Document	1.0
First Dozen Views	10/15/2022	Added sections 2.1.1 - 2.1.3 2.2.1 - 2.2.3 2.3.1 - 2.3.2 2.4.1 - 2.4.6 2.5	2.0
Component Diagrams	10/29/2022	Added sections 2.2.4 – 2.2.13 2.3.3 – 2.3.32 Additions to §2.5	3.0
Interfaces and Data Structures	11/12/2022	Added sections 2.1.4 – 2.1.6.2 2.2.14 – 2.2.20 2.3.33 Additions to §2.5	4.0
Algorithms	12/3/2022	Added sections 2.2.21 – 2.2.64 2.3.34 - 2.3.76 Additions to §2.5	5.0

Final	12/10/2022	The entire document has been reviewed in detail, and most sections have received minor revisions to grammar, references, and some viewpoints/diagrams.	6.0

1.6. Stakeholders

This program is designed for Linwood Jones, the project sponsor. For suggestions on changing the design, contact Abel Wenning. Considerations to take into account include the Software Requirements Specification document listed in the references and the overall goal of the Career Bound system.

As for the application users, the stakeholders include any registered students and company representatives interested in employer-to-student connections for employment purposes.

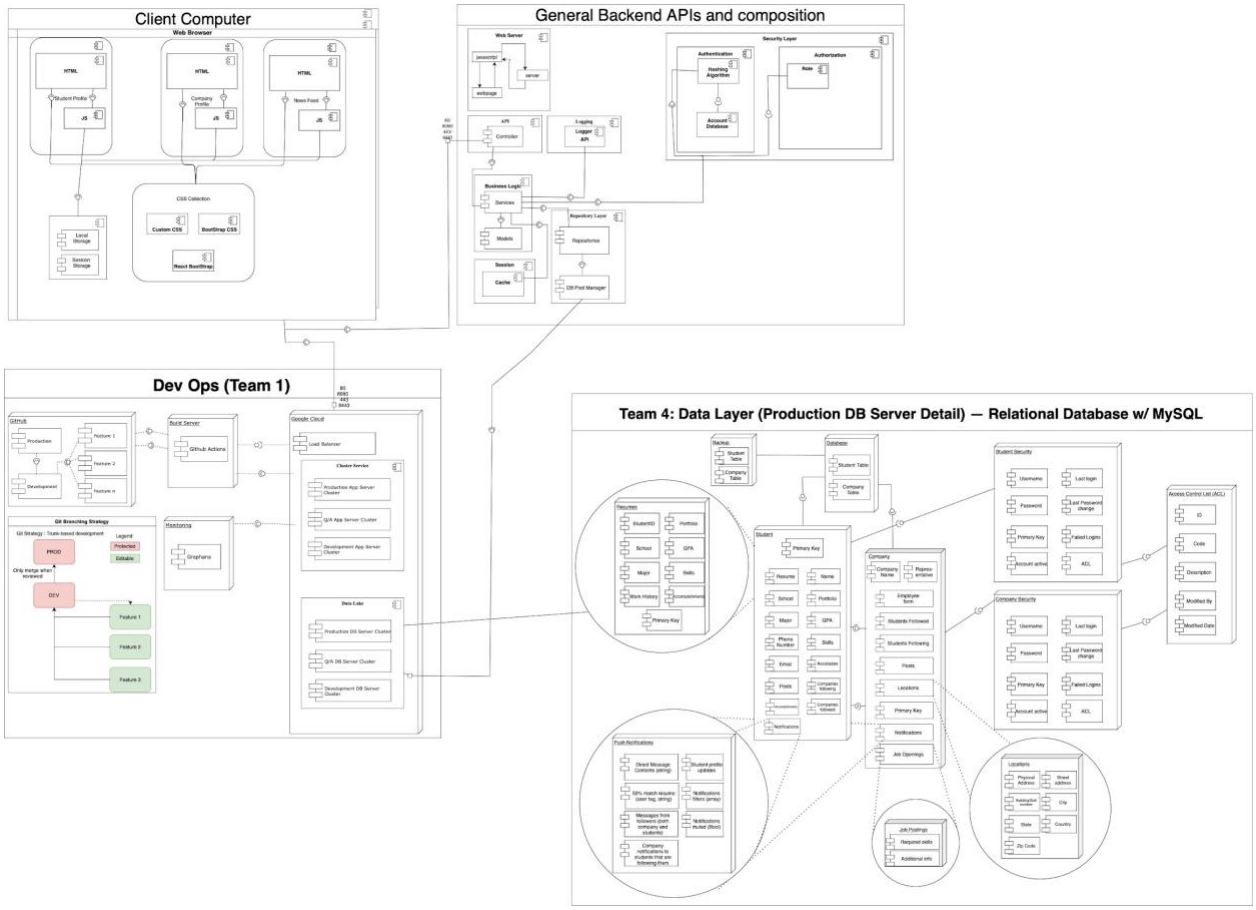
1.7. Stakeholders

This program is designed for Linwood Jones, the project sponsor. For suggestions on changing the design, contact Abel Wenning. Considerations to take into account include the Software Requirements Specification document listed in the references and the overall goal of the Career Bound system.

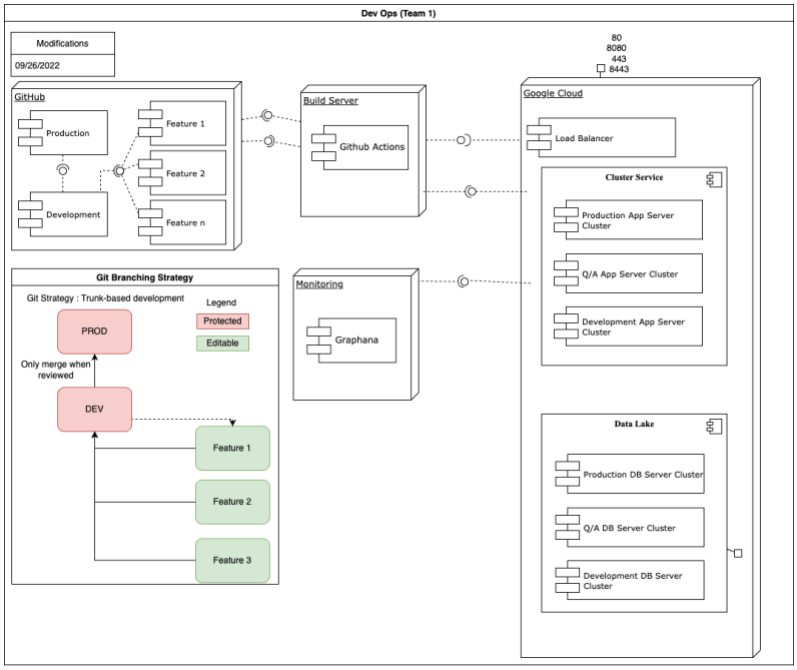
As for the application users, the stakeholders include any registered students and company representatives interested in employer-to-student connections for employment purposes.

2. Design Views

Architectural application overview:

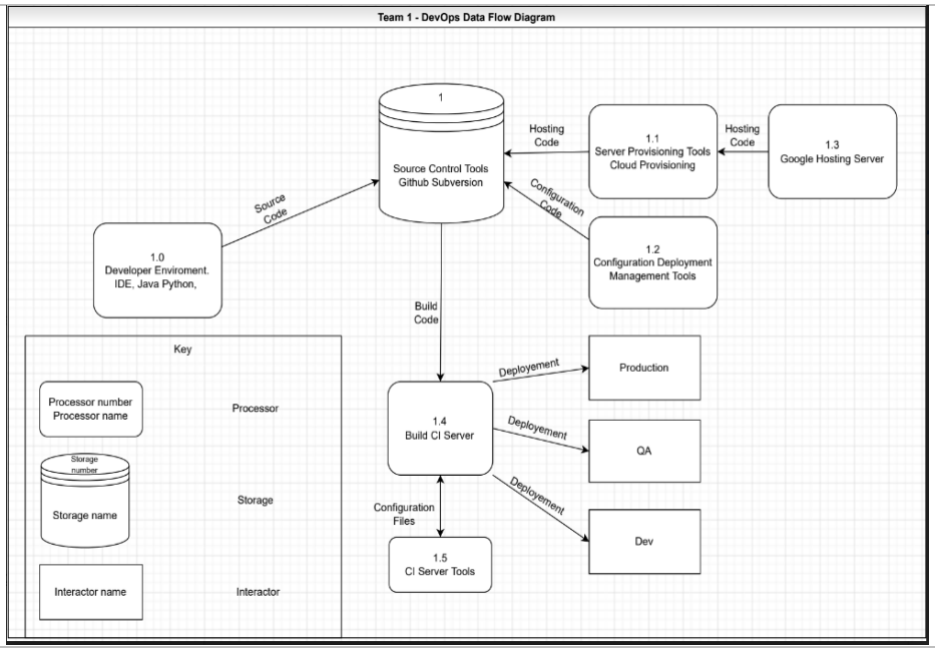


2.1. DevOps



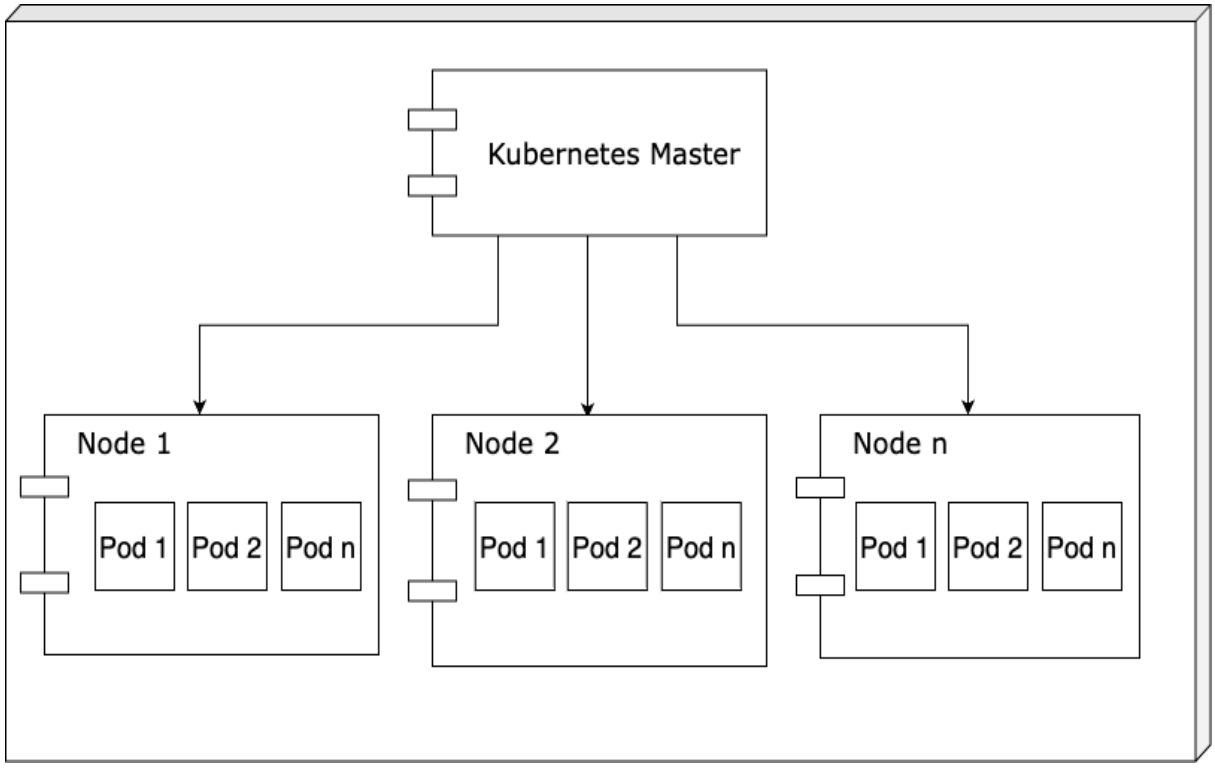
Name	**View**
Purpose	Describe overall DevOps architecture [1]
Description	The diagram describes the cross-functional delivery and requirements of DevOps. In addition, the diagram describes the automated integrations between the software and the deployment of the software code with operations.
Requirements	1.1, 4.1, 4.2, 4.3
Elements	**GitHub:** Web-based software used for version control repository
	Git: Version control and source code management
	Build Server: The build server consists of web servers, application servers, and database servers
	Google Cloud: Cloud computing is a model for enabling on-demand networking, and IT resources Google Cloud will hold the data [2] **Server Clusters:** 2.1.2
	Monitoring: Grafana will be used to monitor and analyze the software components and data [3]
Referenced By	2
Viewpoint	Component Diagram

2.1.1. Data Flow Diagram



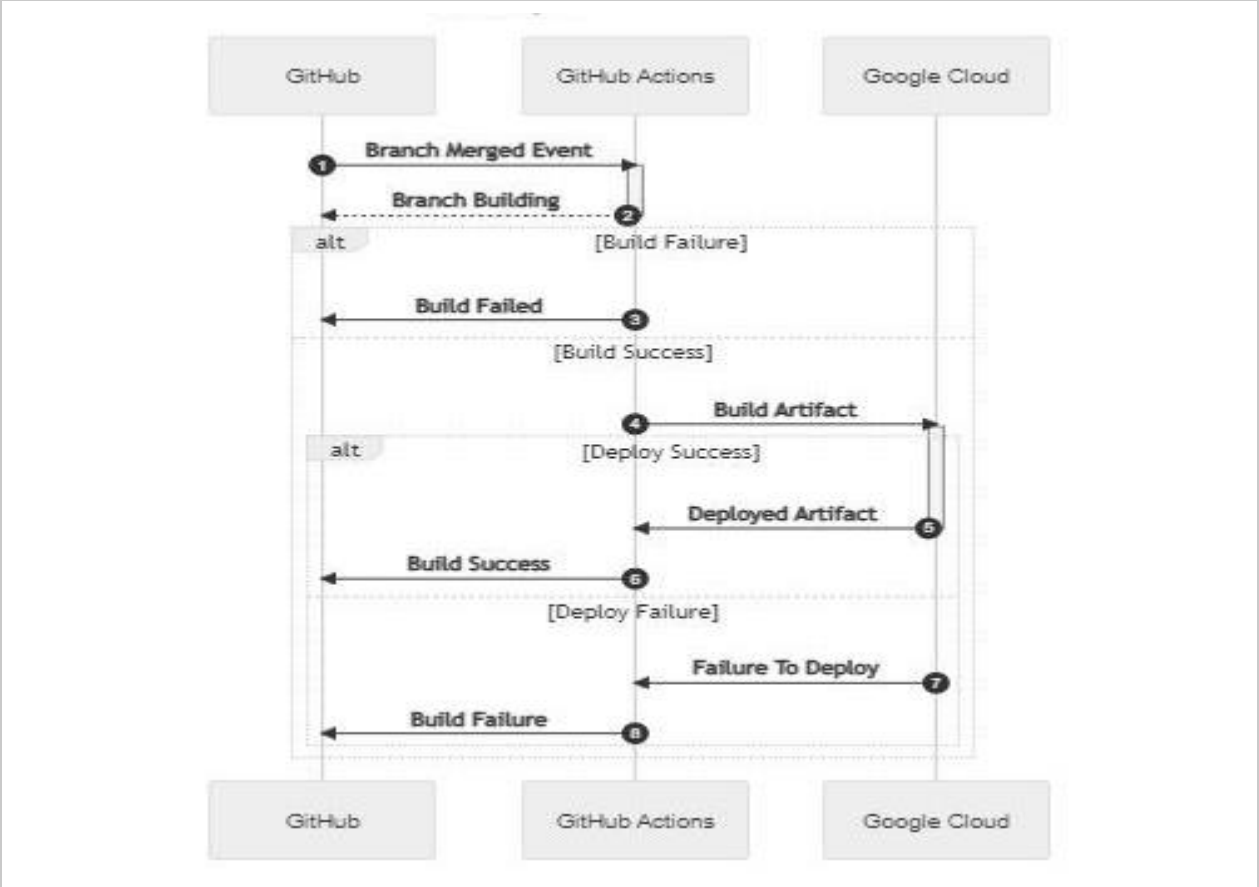
Name	**DevOps Data Flow Diagram**
Purpose	Describes the DevOps DFD
Description	Basic DFD where data starts from a developer IDE and flows through the source control, cloud servers, and hosting to deployment
Requirements	1.1, 4.1, 4.2, 4.3
Elements	**Developer Environment**: Used to produce the original code
	Source Control: Data storage where code is stored
	Cloud Hosting Server: Code is taken from the hosting server to provide data services for the application
	Server Provisioning Tools: Tools to help the code and the server communicate
	Configuration Deployment Management Tools: Tools used to support the deployment of the server
	Build CI (Continuous Integration) Server: Server that builds and deploys the product
	CI Server Tools: Meant to communicate the will of the developers to the build server
	Production: Deployment of the final system for the user to interact with the web application.
	QA: Deployment of the web application to check quality
	Dev: Deployment of the web application that the development team will use
Referenced By	2.1
Viewpoint	Data Flow Diagram

2.1.2. Component Diagram



Name	**Sever (Kubernetes) Cluster [4]**
Purpose	Describe the architecture of the server clusters within the Google Cloud
Description	The diagram describes how servers are clustered, providing the application the ability to run reliably by providing redundancy.
Requirements	1.1, 4.1, 4.2, 4.3
Elements	**Kubernetes Master:** Node that controls and manages other nodes
	Nodes: Contains needed services to run pods
	Pods: Group of containers sharing storage and network resources and a specification on how to run the containers
Referenced By	2.1
Viewpoint	Component Diagram

2.1.3. Sequence Diagram



Name	**DevOps Sequence Diagram**
Purpose	Describes the Sequence of Events in DevOps
Description	A simple sequence diagram starts as code upload at the GitHub server
Requirements	1.1, 4.1, 4.2, 4.3
Elements	**GitHub:** Web-based software used for version control repository
	GitHub Actions: Continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline
	Google Cloud: Used for networking and storing data
Referenced By	2.1
Viewpoint	Sequence Diagram

2.1.4. Server-Server Interface

TCP/IP are a set of rules and procedures that can be used to communicate with a computer network. TCP defines how user requests can create communication channels in a network and manages how data and requests are broken down into smaller parts, which are then sent through the internet and restored to how they were before when they reached the user who sent for the data. IP is the address of a network on the internet and the route needed to reach it.

In the case of the company, a user who is trying to access the company's website would have to first send a message or request to access data from our network, which will then check if they are allowed to access the website or parts of the website. If they are, then the network will assemble the data and information for the website into parts through TCP and then send it to the user's IP address, where it will be put back together by their TCP for them to view and use.

2.1.5. Client-Server Interface

The client-server interface will use HTTPS for communication, Bcrypt for password hashing, and session cookies to maintain sessions. (See section 2.1.6).

2.1.6. Security Interfaces and Encryption Protocols

2.1.6.1 HTTPS

With our systems, HTTPS will allow users to authenticate the origin and identity of our site based on certificate authorities that come pre-installed in their software. This authentication ensures that we are who we say we are and that our site is trustworthy, thereby protecting users against any man-in-the-middle attacks. The HTTPS protocols will then allow users to pass data back and forth with the site that has been bidirectionally encrypted using Transport Layer Security (TLS). This data will include the HTTP protocols, such as the request's URL, query parameters, headers, and cookies, but not the website's address or port. These protocols will allow for the privacy of the user's personal information and passwords and ensure that they aren’t being eavesdropped on.

2.1.6.2 Hashing Password

The interface to the hashing passwords is as follows:

The system uses a library called Bcrypt. Bcrypt allows you to take a plain text password as input and how complex the hashing is as passes. Fifteen passes should be good enough for a brute-force-resistant hash. After passing this to Bcrypt, it will return a string of the complete hash. This can then be stored.

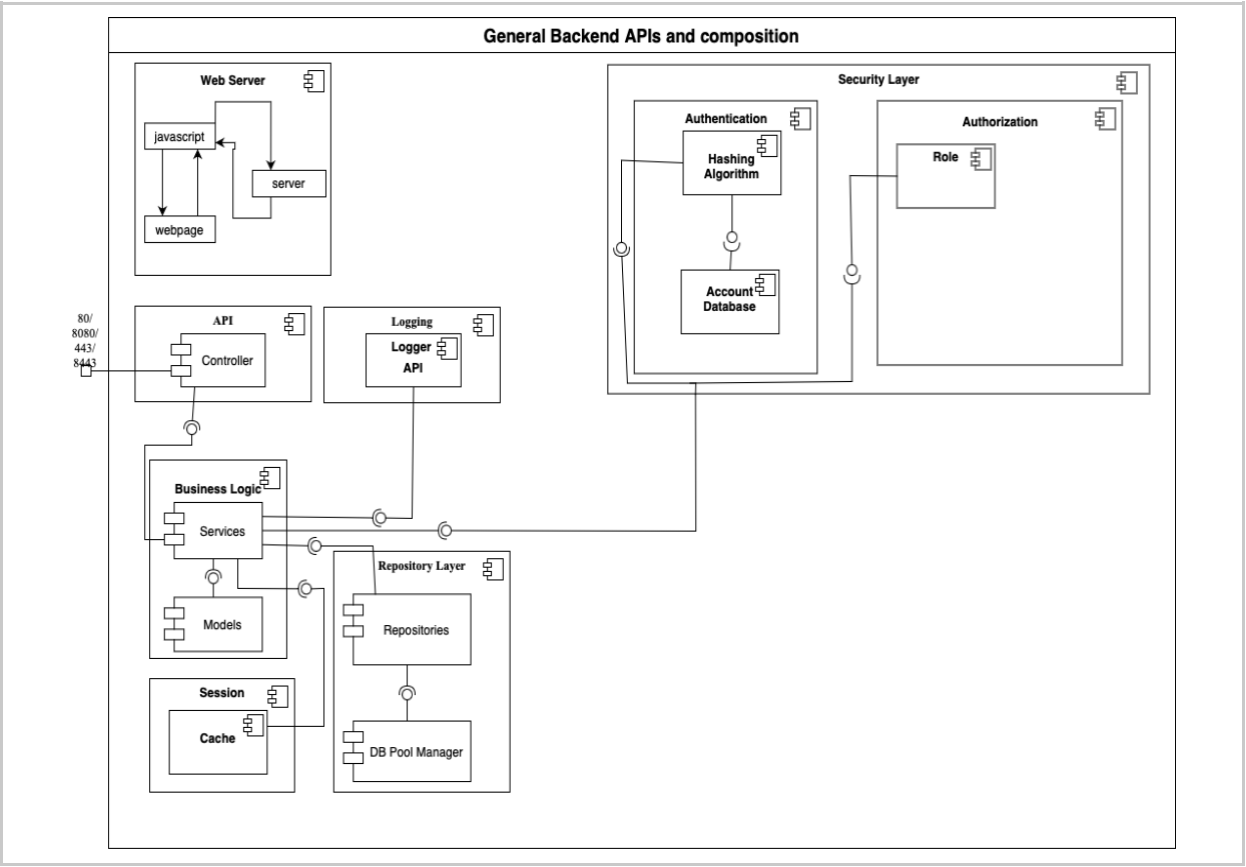
To compare a plain text password to a hashed password, Bcrypt exposes an interface that allows you to test a plain text password to an already hashed and stored hash. It returns true or false depending on whether the hash and the plain password text are effectively the same.

2.1.6.3 Session Cookie

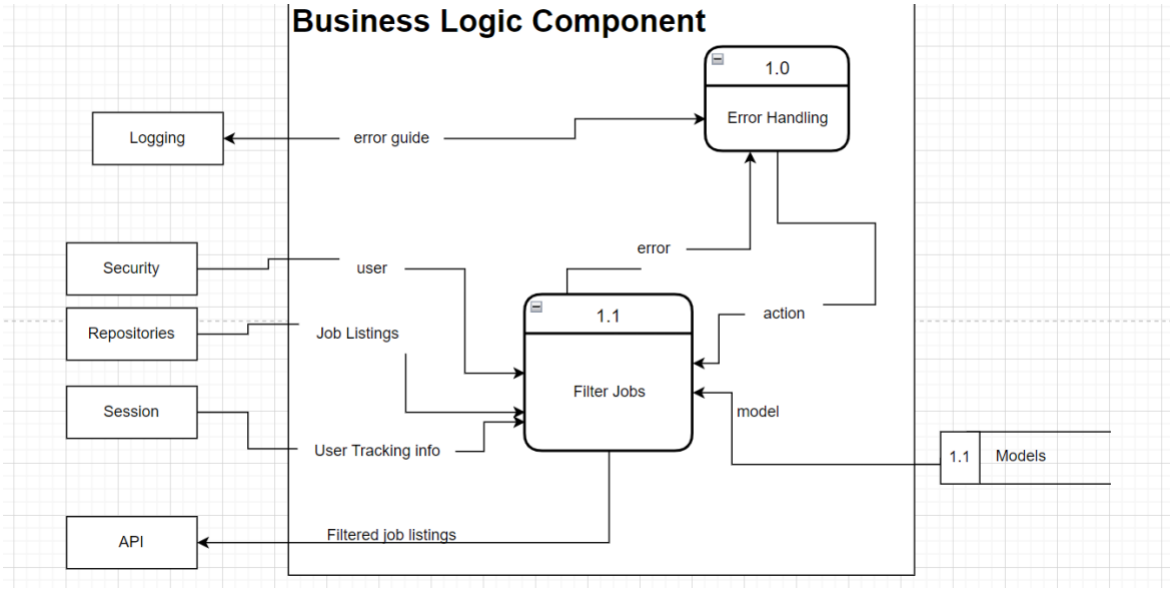
When a user logs in, the server will send the client a session cookie. The client can send this cookie with every request to the server to prove the client's identity in order to receive personalized and session-specific content. The following attributes will be used in each session cookie: HTTPOnly and SameSite: Strict.

2.2. Logic Tier – Backend

Backend Logic Tier overview:

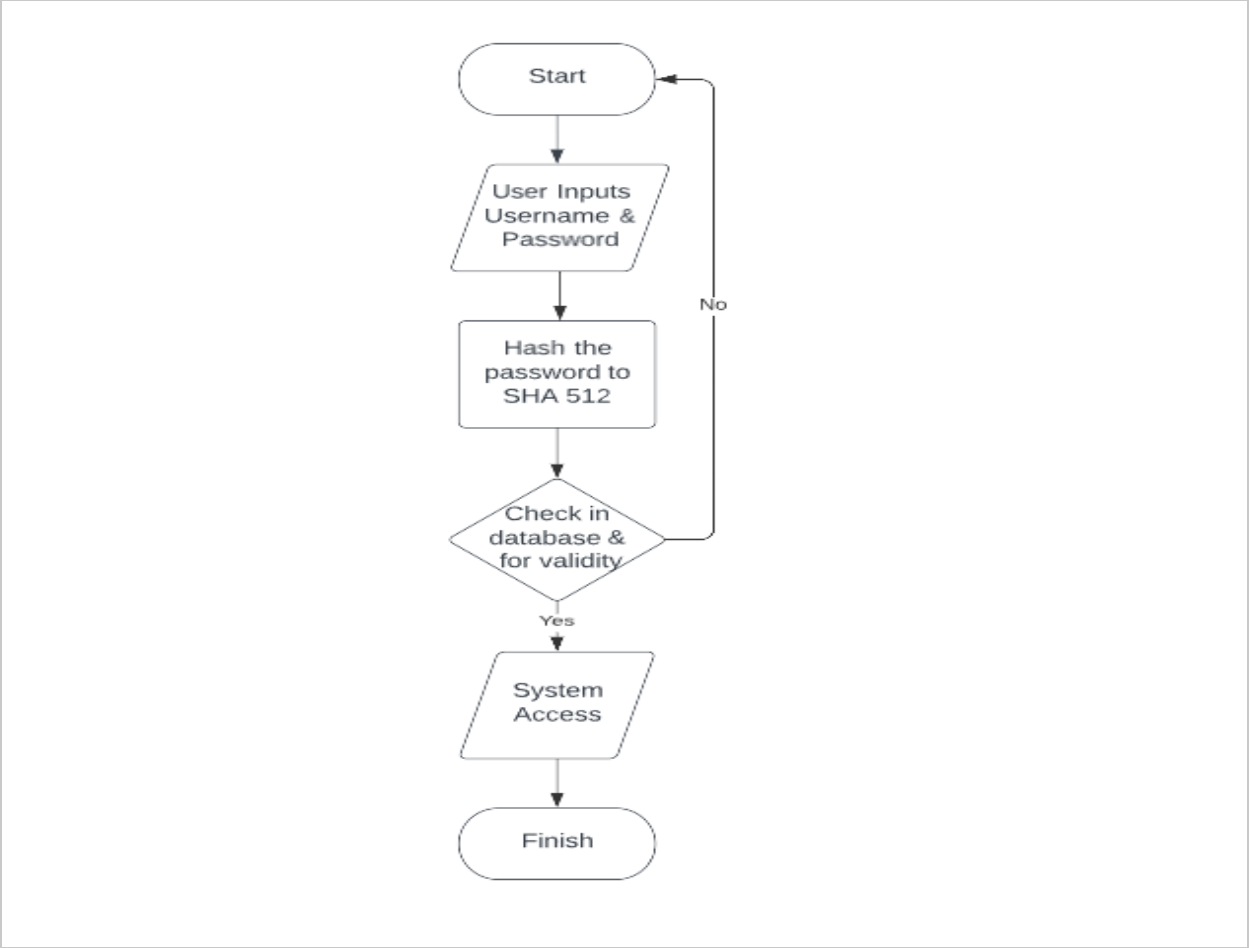


2.2.1. Data Flow Diagram



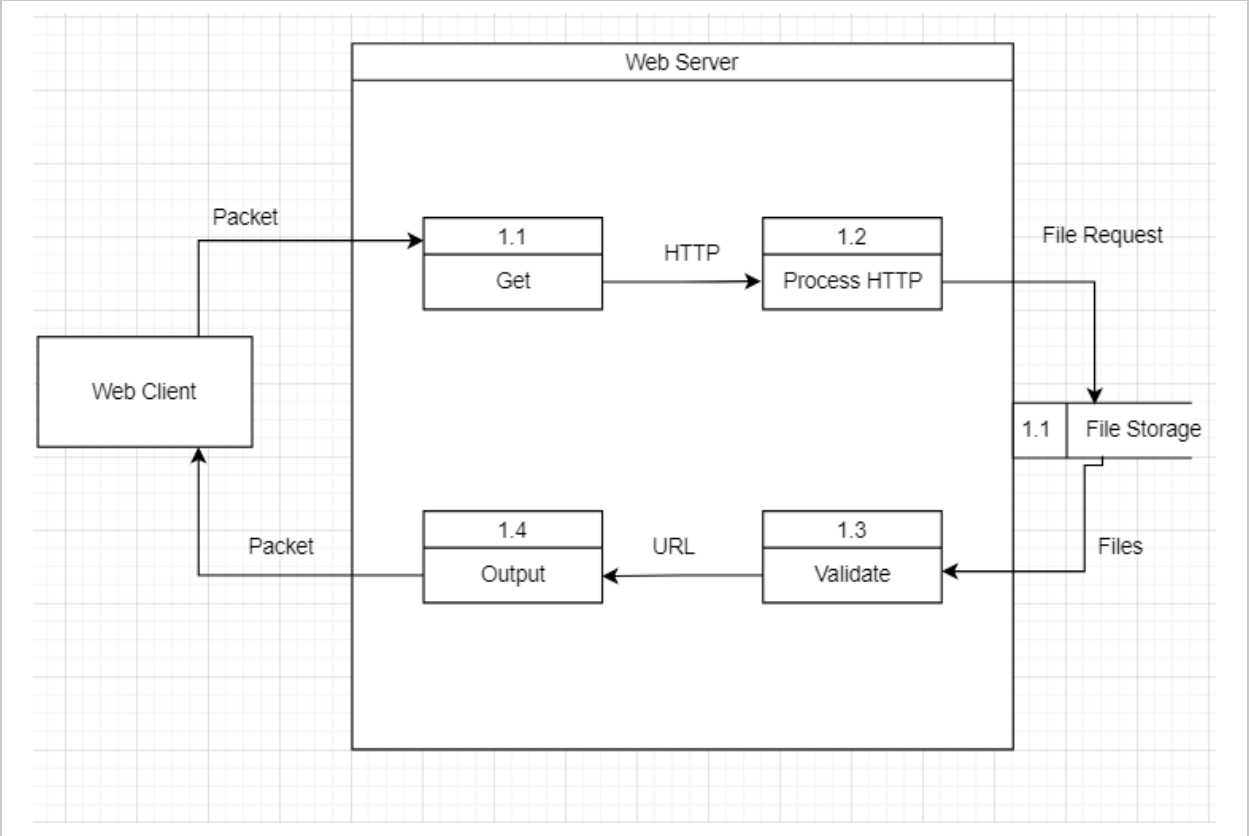
Name	**Company Logic View**
Purpose	Flow of data between database and user interface
Description	A series of computer algorithms containing guidelines for creating, storing, and processing data.
Requirements	2.5.9, 3.1, 3.3, 3.4
Elements	**Logging:** Contains information for error handling.
	Security: Brings validated user information.
	Repositories: Contains records of job listings.
	Session: Stored information about the user and other cached info.
	API: Connection from the backend to the frontend website where processed information is directed.
	Models: Internally stored information on different ways to process information.
Referenced By	2.2, 2.2.15, 2.2.30, 2.2.59, 2.3.40
Viewpoint	Data Flow Diagram (DFD)

2.2.2. Authentication Flowchart



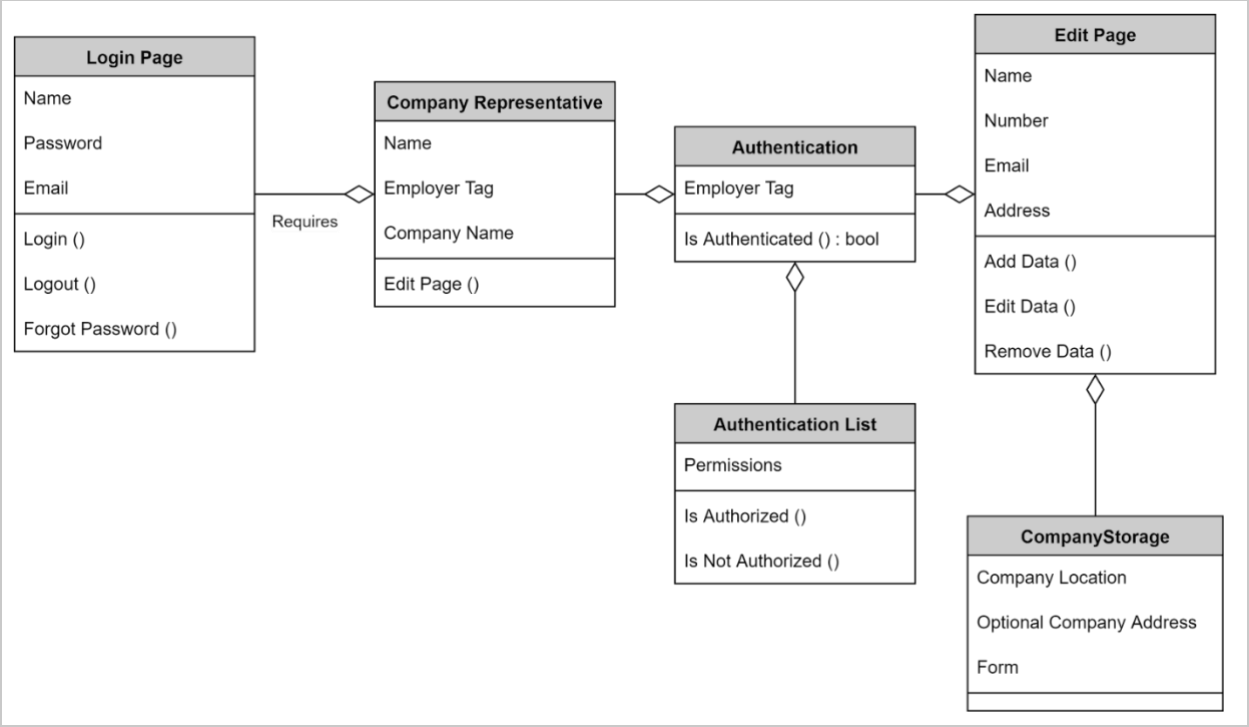
Name	**Security Layer View**
Purpose	Secure user passwords.
Description	Flowchart gives a visual description of user passwords being hashed and stored/checked in the database.
Requirements	2.1.3.3, 2.1.3.10, 3.1.2
Elements	**Hashing Algorithm:** Used to authenticate users when first accessing the system.
	Username: user specific, used to access account- should never change
	Password: user specific, needed in combination with the User's username to access account. Changes occasionally with necessity or choice, etc.
	Database: Used to validate the Username and Password-contains user information
Referenced By	2.2
Viewpoint	Flowchart

2.2.3. Data Flow Diagram



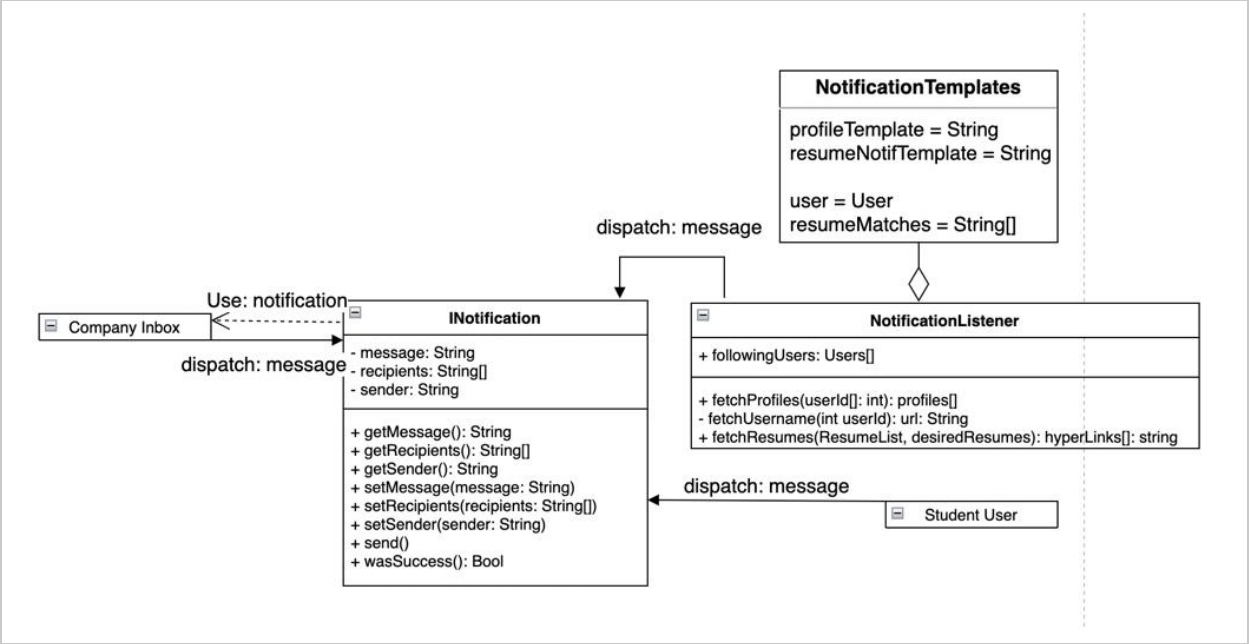
Name	**Web Server View**
Purpose	Describes overall web server view
Description	Illustrates interactions between the web server and other components of the system
Requirements	4.1, 4.2, 4.3
Elements	**File Storage:** Stores HTML, CSS, and JavaScript files
	Web Client or Browser: Allows the user to request files for web pages, makes a request to a web server and renders given files
	Web Server: Receives and interprets HTTP requests, then returns appropriate files or error codes
Referenced By	2.2
Viewpoint	Data Flow Diagram

2.2.4. Companies Profile Page: UML Class Diagram



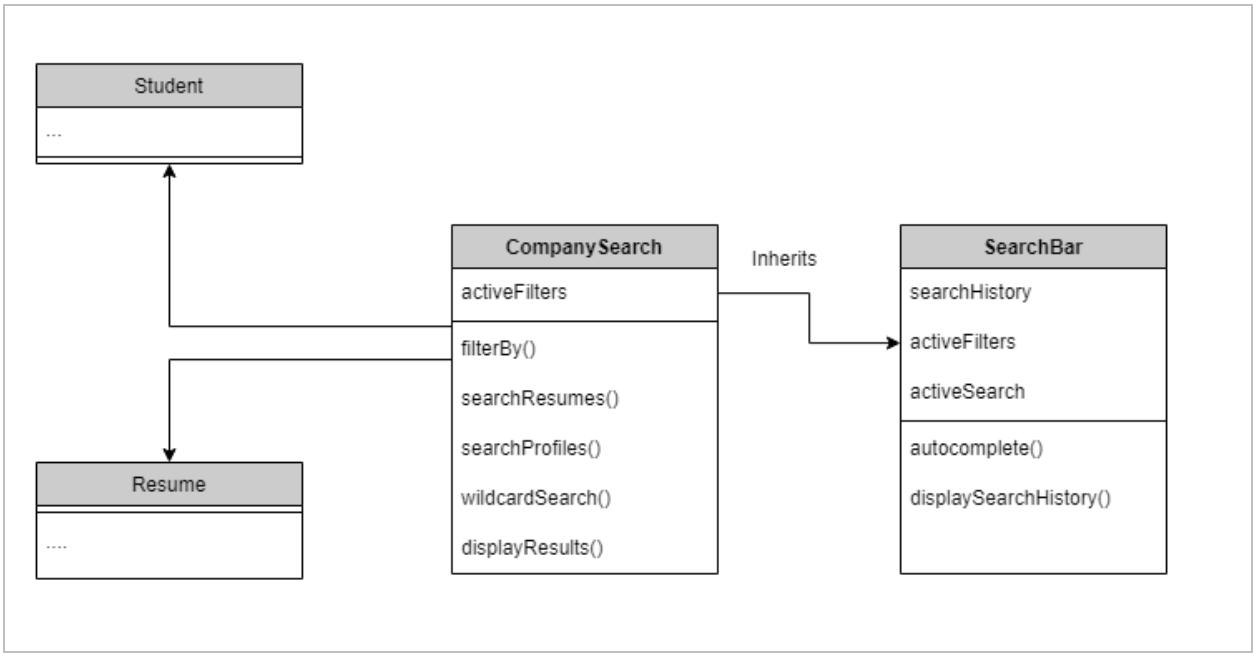
Name	**Companies Profile Page (Backend): UML Class Diagram**
Purpose	Describes the required components for the user to interact with the backend via the company profile page
Description	The above components will be used in tandem with page elements to interface the user to log in, edit company profile, and otherwise interact with backend in an encapsulated, loosely coupled way.
Requirements	2.1.1 - 2.1.3.10
Elements	**Login Page:** Initial login authentication page where the user enters credentials for confirmation
	Company Representative: Holds information about a company employee.
	Authentication: Specific permission given to companies that own this page.
	Authentication List: List of default authentications or granted permissions assigned to the company, as well as helper methods specifying if the company has necessary authentication
	Company Storage: Holds all information about the company retrieved from the backend or cache.
	Edit Page: Interface to edit company information.
Referenced By	2.1
Viewpoint	UML Class Diagram

2.2.5. Companies Notifications: UML Class Diagram



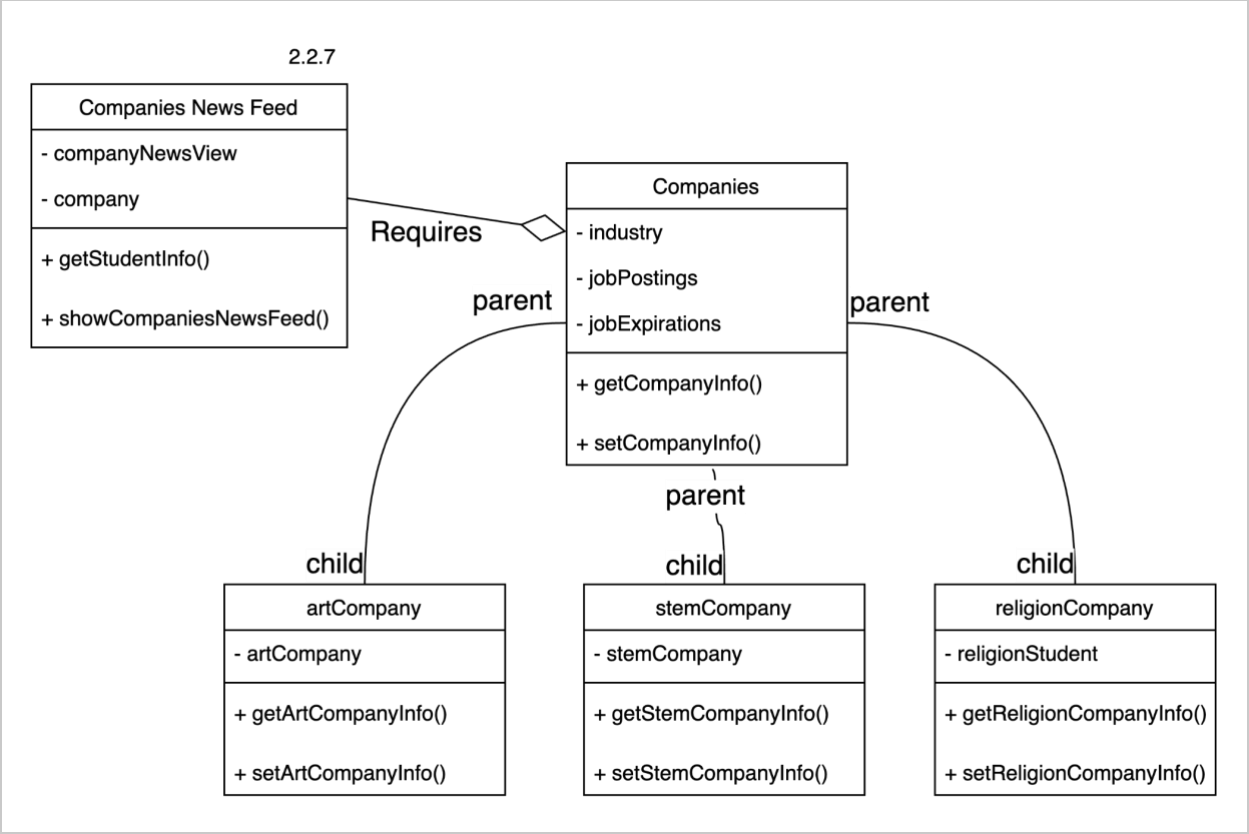
Name	**Company Notifications UML Class Diagram (Backend)**
Purpose	Highlight the different entities relating to company-specific notifications and reference them to the behaviors their respective front-end components call upon.
Description	Companies are notified of updates of student user profiles and resumes of the student users the company profile page is following. Both students and companies can send and receive messages.
Requirements	2.2.1, 2.2.3, 2.2.4, 2.2.5, 3.2.3
Elements	**Company Inbox:** Entity representing the location where notifications and messages can be viewed from the UI by the company representative.
	Notification: The listed statuses and behaviors of standard notifications.
	Notification Templates: Used for other types of notifications whose messages are automated, such as profile updates and resume match of students the company is following.
	Student User: Any student user sending messages to companies.
	Notification Listener: A listener made available in the cloud environment that constantly watches for events that trigger notifications.
Referenced By	2.2.25-27
Viewpoint	UML Class Diagram

2.2.6. Companies Search: UML Class Diagram



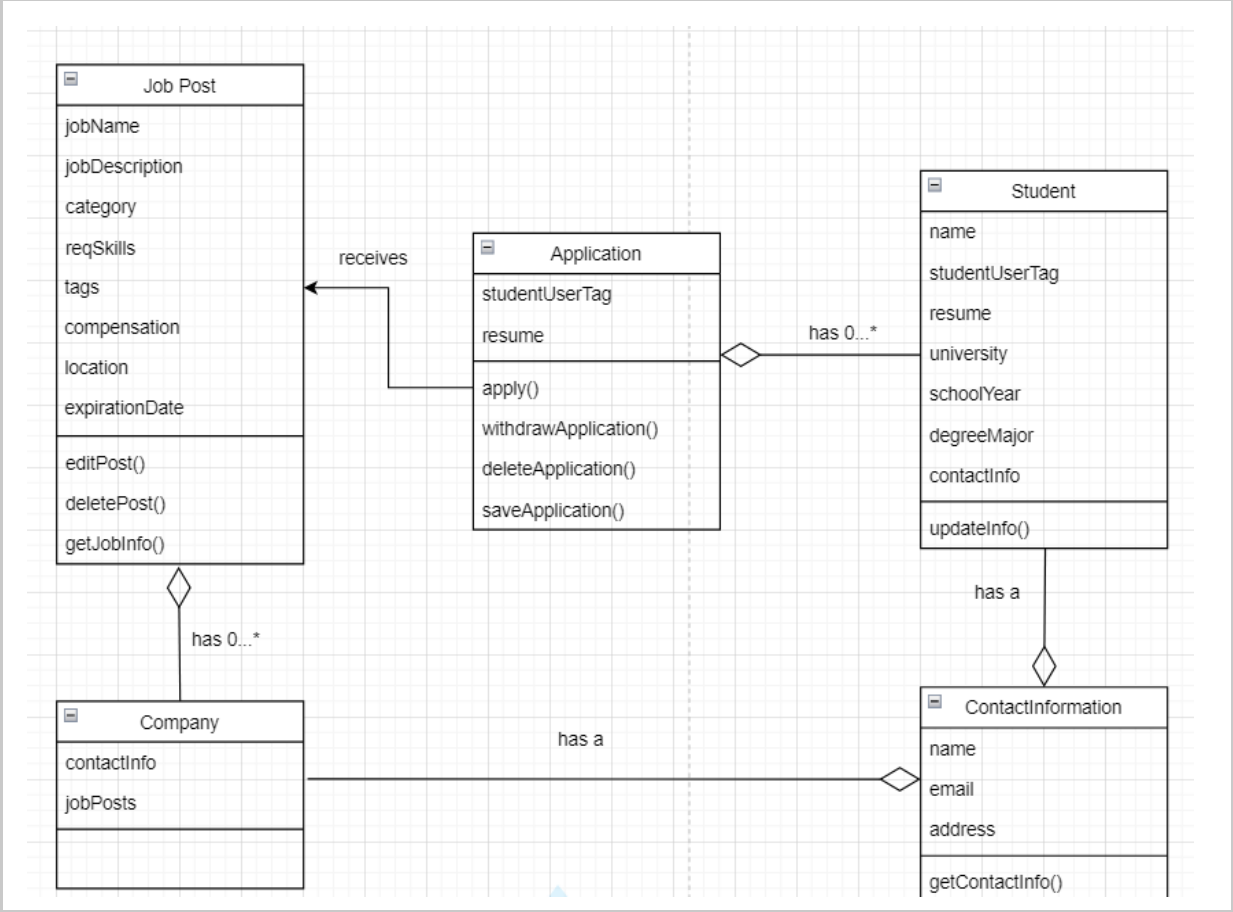
Name	**Search UML Class Diagram**
Purpose	Describes how the classes work together to search for student profile users.
Description	This describes how the classes would work together to search for a candidate for a company.
Requirements	2.2.2, 2.3.1, 2.3.3, 2.3.4, 2.3.8
Elements	**Student:** A user profile page and personal information.
	Resume: This would include name, education level, and work history.
	Filters: Company, for example, could search by education level.
	Display Results: Search result information from database is sent to frontend.
	Search [for entity]: Path to query database for different entities.
	SearchBar: A HTML/CSS component where the user enters search parameters.
	Search History: User's previous search parameters.
	Autocomplete: A method that predicts the input from a user.
Referenced By	2.3, 2.3.10, 2.3.11, 2.3.22, 2.3.40, 2.3.41
Viewpoint	Class diagram

2.2.7. Companies News Feed: UML Class Diagram



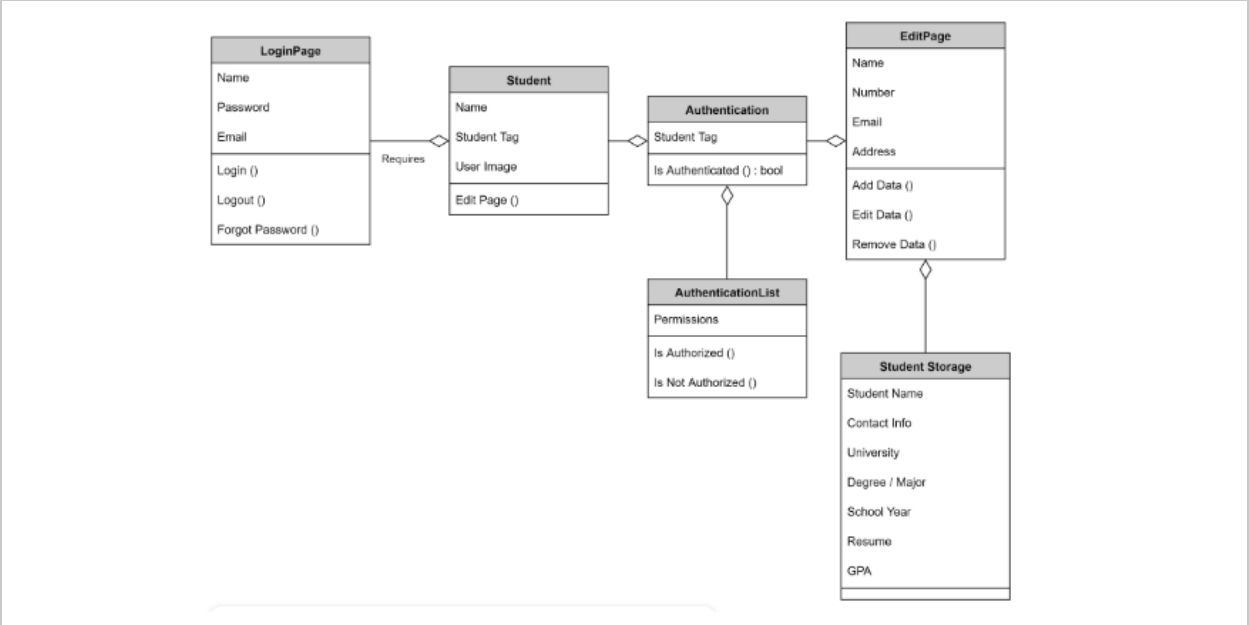
Name	**Companies News Feed UML Class Diagram**
Purpose	Present UML class diagram for a company's news feed
Description	A company news feed shall display the company's post as well as display student's users post. A company will want to see relevant news information in their feed, such as but not limited to the student information, student preferences, etc.
Requirements	2.4.1 - 2.4.6
Elements	**Companies News Feed:** Information displayed on the company news feed.
	Company: Holds company information, including industry, job posting, job expirations, job salary, etc.
	Company Type: Child class of Parent Company, used to show news relevant to the company type.
Referenced By	2.2.7
Viewpoint	**UML Class Diagram**

2.2.8. Companies Job Postings: UML Class Diagram



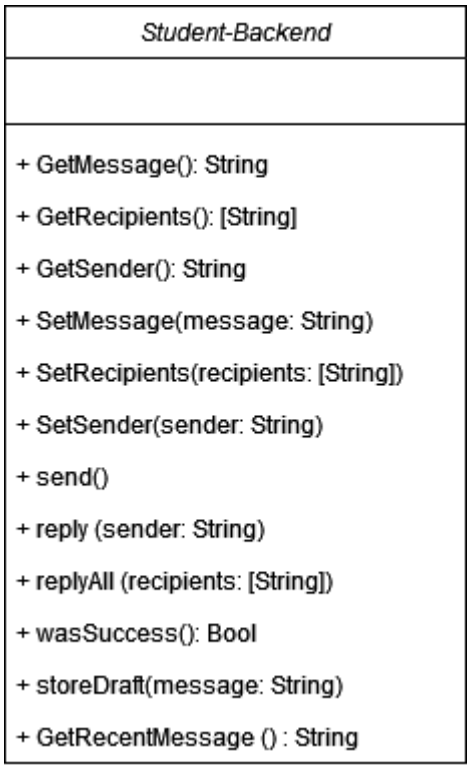
Name	**Job Postings (Backend): UML Class Diagram**
Purpose	Describes the connection between different entities, their attributes, and the actions performed by each entity.
Description	Contains the relationship of classes in the logic layer for the creation, alter, or remove of a job posting and the relationship between students and a job posting.
Requirements	2.5.1 - 2.5.7
Elements	**Company:** Organization consisting of employees seeking to hire additional employees.
	Student: User studying at a school or college.
	Application: Form used by a company to gather information on prospective employees.
	Job Post: Post consisting of job information made by the employer.
	Contact Information: Contact details about an entity, including name, phone number, email, and address, etc.
Referenced By	2.2, 2.2.28 - 2.2.34, 2.2.38
Viewpoint	UML Class Diagram

2.2.9. Students Profile Page: UML Class Diagram



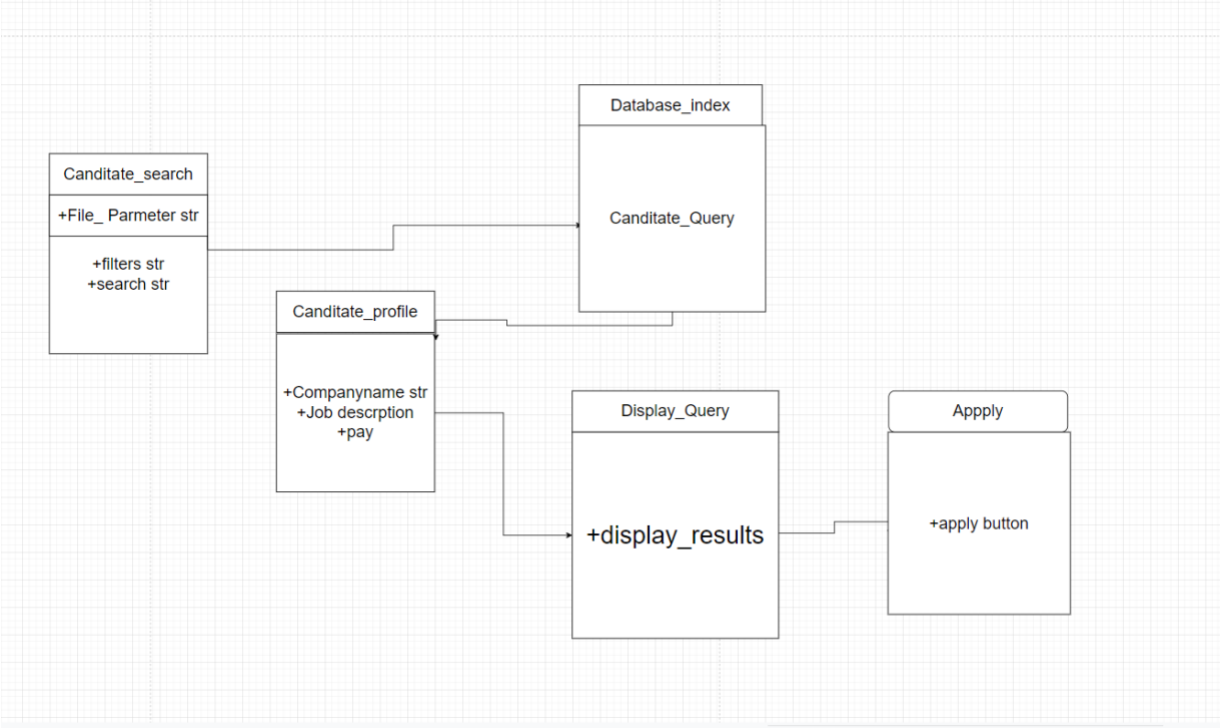
Name	**Students Profile Page (Backend): UML Class Diagram**
Purpose	Describes the required components for the user to interact with the backend via the student profile page.
Description	The above components are used in tandem with page elements to interface the user to log in, edit their profile page, and otherwise interact with the backend in an encapsulated, loosely coupled way.
Requirements	3.1.1 - 3.1.3.4
Elements	**Login Page:** Initial login authentication page where the user enters credentials for confirmation.
	Student: Holds basic student information and allows users to edit information.
	Authentication: Specific permission is given to students.
	Authentication List: List of authentications or granted permissions given to a student, as well as helper methods specifying if the student has necessary authentication.
	Student Storage: Holds all information about students retrieved from the backend or cache.
	Edit Page: Interface to edit student information.
Referenced By	3.1
Viewpoint	UML Class Diagram

2.2.10. Students Notifications: UML Class Diagram



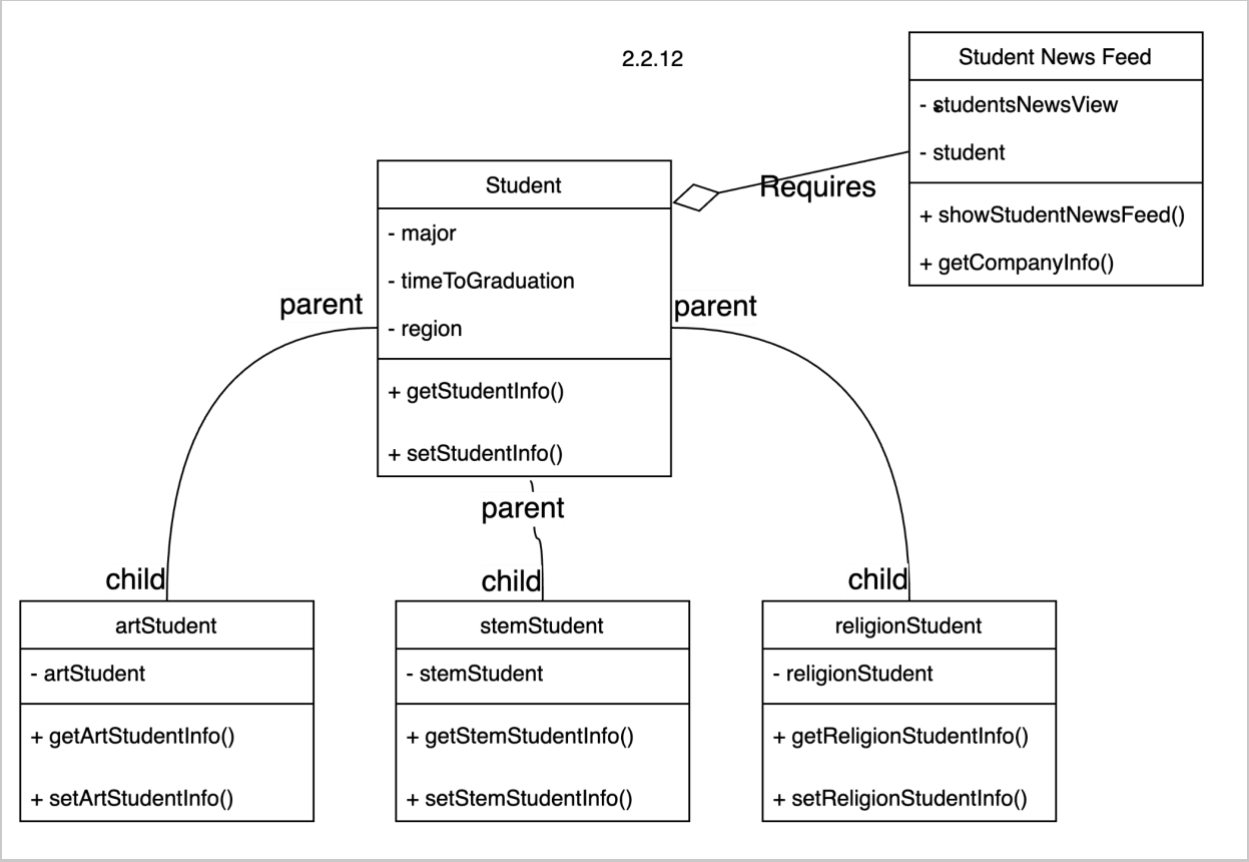
Name	**Students Notifications: UML Class Diagram**
Purpose	Dictate the behaviors of student notifications.
Description	A listing of getters and setters for the components of the notification class for updating and reading the contents of a message along with its sender and recipients.
Requirements	3.2.1 - 3.2.7
Elements	**Getters -** GetRecipients(), GetSender(), GetMessage(): Encapsulated retrieval of private attributes.
	Setters - SetMessage(), SetRecipients(), SetSender(): Encapsulated setting of private attributes.
	Network callbacks - send(), wasSuccess(): Sending messages and verifying they were successfully sent.
	Status saves - storeDraft(), getRecentMessage(): Utilize local caching of unsent messages.
	Directing messages - reply(), replyAll(): Determine to whom the messages are directed.
Referenced By	2.2.1
Viewpoint	UML Class Diagram

2.2.11. Students Search: UML Class Diagram



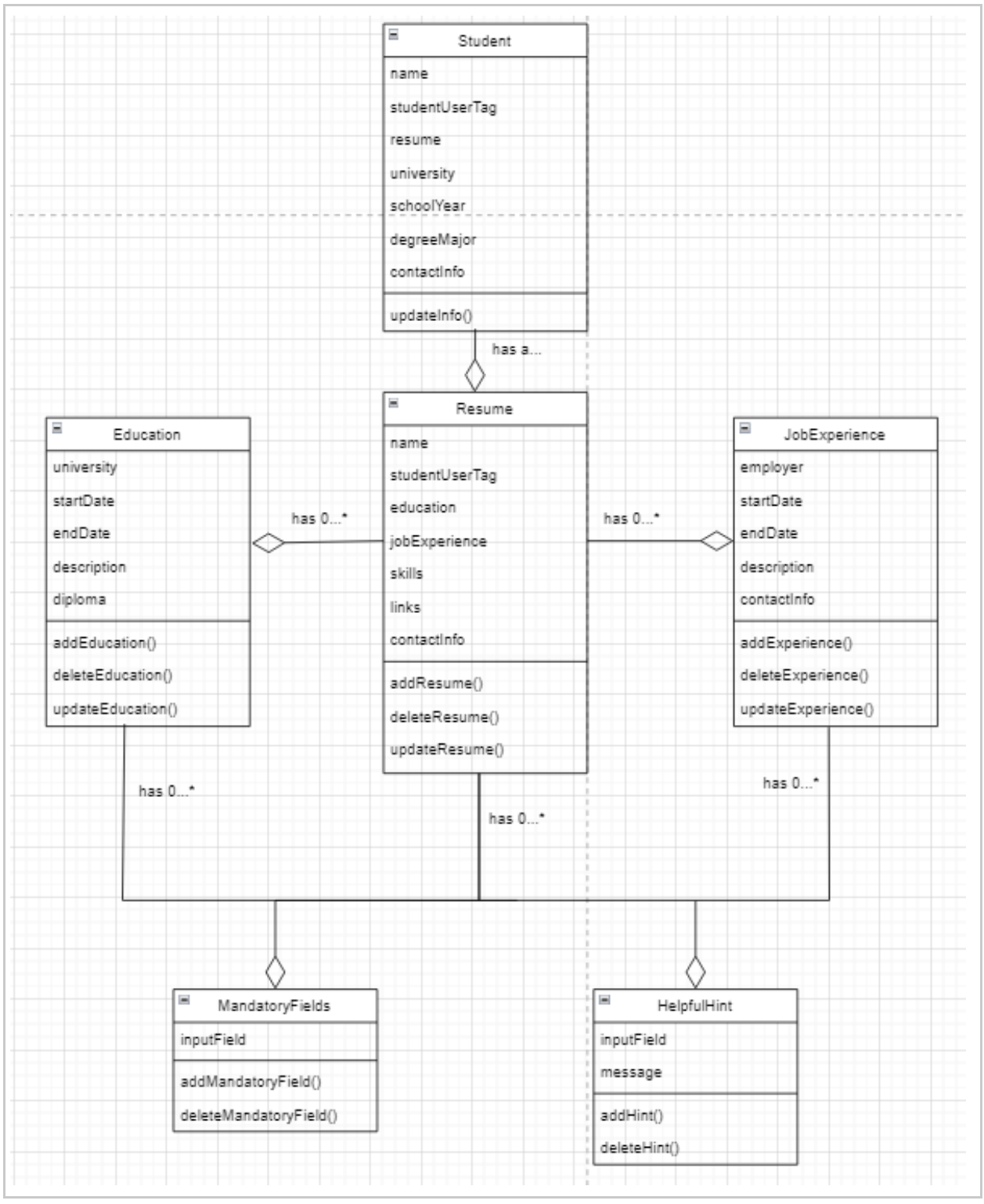
Name	**Search UML Class Diagram**
Purpose	Describes how classes work together for student users.
Description	This describes how classes work for a student user seeking employment using the job posting search program.
Requirements	3.3.1, 3.3.3, 3.3.4
Elements	**Job Posting**: This would have pay, company things along those lines.
	Job Database: Collection of job postings data to be searched, namely company name, job title, required skills, job description, hours , position type, compensation, .location, contact information, and link to apply for the job.
	Filters: Education level needed, pay things along those lines.
	Display: Display query results.
	A way to apply: A method that job seekers can send their information to a job seeker.
Referenced By	2.2
Viewpoint	Class diagram

2.2.12. Students News Feed: UML Class Diagram



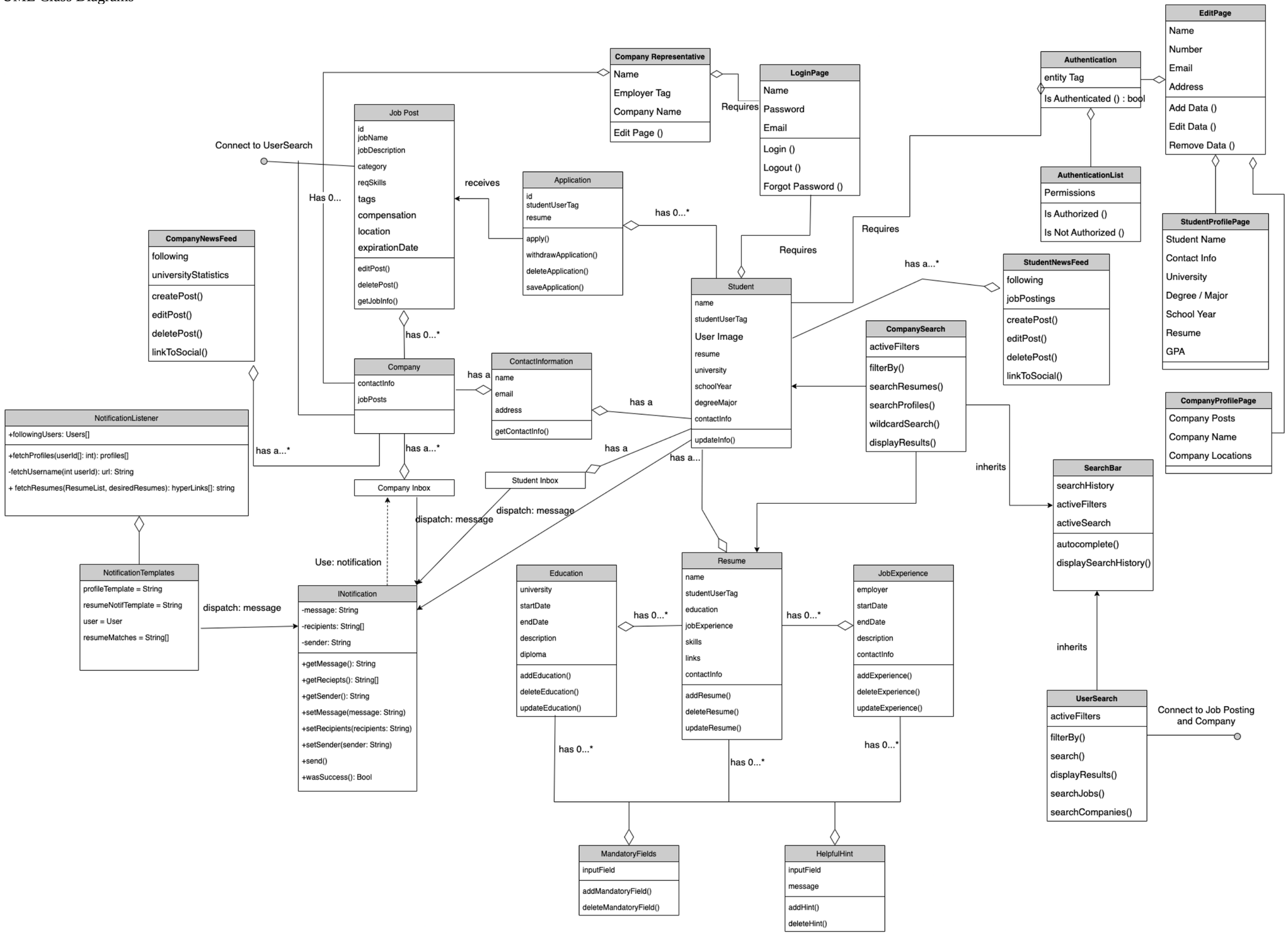
Name	**Student News Feed UML Class Diagram**
Purpose	Describes how the student views the news feed based on student information and company information.
Description	UML for a student news feed, including examples of different student types.
Requirements	.4.1 – 3.4.2
Elements	**Student News Feed:** Displays job posting from the company the student user is following on the student news feed.
	Student: Class that holds student information. Important to note but not limited to their time to graduate, region, major, interests, GPA, contact information, resume, portfolio, and so on.
	Student Type: Child class of Student class, used to show news relevant to student based on type.
Referenced By	2.2.12
Viewpoint	UML Class Diagram

2.2.13. Students Resume: UML Class Diagram



Name	**Resume (Backend): UML Class Diagram**
Purpose	Describes connection between different entities, their attributes, and actions performed by each entity.
Description	Contains the relationship of classes in the logic layer for the creation of resumes.
Requirements	3.5.1, 3.5.2
Elements	**Mandatory Field:** Text field that requires user input before the submission is possible.
	Helpful Hint: Information is available to the user to help indicate what should be entered into a text field.
	Resume: Plain text document that provides an employer with a student user's information.
	Education: Any experience gained while attending a college or trade school.
	Job Experience: Any experience gained while working in a field or occupation.
	Student: A person who is studying at a school or college. (“Student | Encyclopedia.com”)
Referenced By	2.2
Viewpoint	UML Class Diagram

2.2.14. All Backend UML Class Diagrams



Name	**All Backend UML Class Diagrams**
Purpose	Describes the interfaces among all Backend classes.
Description	Contains the relationship of all classes in the logic layer.
Requirements	2.1 – 3.4
Elements	**Application:** A page containing the form for an application to a company
	Authentication: The process of verifying the identity and permissions of the user
	Authentication List: The list of permissions granted to users
	Company: A page containing the information for a company
	Company Inbox: The page that company representatives will go to see any incoming messages or notifications
	Company News Feed: The main landing page for company users
	Company Profile Page: Contains all relevant information for the specific Company
	Company Representative: A page containing the information for a company representative
	Company Search: The access for the user to search the company database
	Contact Information: A page containing the contact info for an individual
	Edit Page: Give ability to edit (Profile) page
	Education: The information about the education for an individual
	Helpful Hint: A tooltip or other front-end element that provides insight or instruction for an element, usually a form field
	Job Experience: The information about the job experience for an individual
	Job Post: A page containing the details and info about a job
	Login Page: The page with which the user will interface to undergo authentication
	Mandatory Field: A field in the form the user will submit that, if empty, prevents the user from continuing or submitting
	Notification: The instance of a notification as seen in the notification template
	Notification Listener: An event listener that listens for a response from users.
	Notification Template: A page snippet or object with the front-end elements which compose the notification view
	Resume: The resume document of an individual
	Search Bar: The UI element which the user can interact with to access the User Search
	Student: The information about a student
	Student Inbox: The page that the student will go to see any incoming messages or notifications
	Student News Feed: The main landing page for student users
	Student Profile Page: Contains all relevant information for the Student
	User Search: The access for the user to search the user database
Referenced By	2.2.4 - 2.2.13, 2.2.15, 2.2.16 – 2.2.64
Viewpoint	UML Class Diagram

2.2.15. REST API Endpoints

Endpoint	Method	Request Body / URL Query Parameters	Response Body	Effect(s) on Servers
/student?...	GET	first_name=<search-string, char 1-50> last_name=<search-string, char 1-50> full_name=<search-string, char 1-75>	JSON array of all students with a name containing the letters in {name}	Fetch specific student's data from the server
/student?...	GET	skill=<search-string> gpa-min=<0 - 4> yrs-complete=<0 - 9>	JSON array of all students who meet the specified search criteria	Fetch specific student's data from the server
/student/{id}	GET	-	JSON with the student's details fetched from the unique {id}	Fetch specific student's data from the server
/student	POST	JSON with the student's details	{result: true | false}	Create a new student
/student/{id}	PUT	JSON with the student's details	{result: true | false}	Update a student
/student/{id}	DELETE	-	{result: true | false}	Make a student and their resume inactive
/company/{name}	GET	-	JSON array of all companies with a name containing the letters in {name}	Fetch specific company's data from the server
/company?...	GET	name=<search-string, char 1-50> industry=<string> location=<string>	JSON array of all companies who meet the specified search criteria	Fetch specific company's data from the server
/company/{id}	GET	-	JSON with the company's details	Fetch specific company's data from the server
/company	POST	JSON with the company's details	{result: true | false}	Create a new company
/company/{id}	PUT	JSON with the company's details	{result: true | false}	Update a company
/company/{id}	DELETE	-	{result: true | false}	Make a company inactive
/companySearch/Resume/{string}	GET	-	JSON formatted data of all resumes matching search criteria	Fetch specific resumes data from the server
/companySearch/Profile/{string}	GET	-	JSON formatted data of all profiles matching search criteria	Fetch specific profiles data from the server
/userSearch/filterBy	POST	JSON with the filter details.	{result: true | false}	The result is that data must fit into parameters set in the filters
/userSearch/{string}	GET	-	JSON formatted data of all jobs from database.	This will return many jobs and companies without the boundaries of filters
/userSearch/Display	GET	-	JSON formatted off data obtained from the other function.	This will display results from the database.
/userSearch/{job}	GET	-	JSON formatted data of jobs with a certain job title.	This will return jobs with a certain job title.
/userSearch/{company}	GET	-	JSON formatted data of job openings at a certain company.	This will return all jobs open at a certain company.
/jobPost/{edit job}	PUT	JSON with the job post details	JSON with the job post details	Update a job post
/jobPost/{delete job}	PUT	JSON with the job post details	{result: true | false}	Remove a job post
/jobPost/{job}	GET	JSON with the job post detail	JSON with the job post details	Fetch specific job post data from the server
/resume?...	GET	resumeName=<search-string, char 1-100>	JSON array of resumes that match the name request parameter	Fetch resumes from storage that match the input name

Endpoint	Method	Request Body / URL Query Parameters	Response Body	Effect(s) on Servers
/resume?...	GET	studentUserTag=<search-string (studentUserTag length?)>	JSON with the resume details	Fetch a specific resume from storage
/resume	POST	JSON with resume details	{result: true | false}	Create a new resume
/resume/{studentUserTag}	PUT	JSON with new resume details	{result: true | false}	Update an existing resume
/resume/{studentUserTag}	DELETE	-	{result: true | false}	Delete an existing resume
/resume/jobExperience/{id}	PUT	JSON with job experience details	{result: true | false}	Adding a job experience to a resume
/resume/jobExperience/{id}	DELETE	-	{result: true | false}	Deletes a job experience section from a resume
/resume/education/{id}	PUT	JSON with education experience details	{result: true | false}	Adding education to a resume
/resume/education/{id}	DELETE	-	{result: true | false}	Deletes an education section from a resume
/feed	GET	-	JSON List of viewable Posts	Fetch list of posts from database
/feed/{postId}	POST	JSON with a new news feed post	{result: true | false}	Create and stores a new post in the database
/feed/{postId}	PUT	JSON with an updated news fee post	{result: true | false}	Update a given news feed post
/feed/{postId}	DELETE	-	{result: true | false}	Delete a given news feed post from database
/studentFeed?...	GET	createTime=<date> message=<string> id=<string, char 24> postId=<string>	JSON array of the published posts for all following companies and students.	Fetch a list of company posts from companies the student is following

/studentFeed/{postId}	POST	commentary: <string>,
		visibility: “public” | ”private”,
		distribution: {
		feedDistribution: “mainFeed”,
		share: <string>
		},
		content: {
		article: {
		source: <string>,
		title: <string>,
		description: <string>
		}
		}
			JSON list of viewable Posts
			A successful response returns a 201 CREATED HTTP status code and the ID along with a new post in the news feed.
				A successful response returns a 201 CREATED HTTP status code and the ID along with a new post in the news feed.

/studentFeed/like/{postId}	POST	commentLikePost: <Boolean>	{result: true | false} 201 Posted HTTP	Like or unlike a post or comment
/studentFeed/{postId}	PUT	commentary: <string>, visibility: “public” | “private”, distribution: {	{result: true | false}	Updates a given student post from id.

Endpoint	Method	Request Body / URL Query Parameters	Response Body	Effect(s) on Servers

		feedDistribution: “mainFeed”,
		share: <string>
		},
		content: {
		article: {
		source: <string>,
		title: <string>,
		description: <string>
		}		

/studentFeed/{postId}	DELETE		Result: true || false	Deletes a given student post from the database
/companyFeed?...	GET	createTime=<date>, message=<string>, id=<string, 24> postId=<string>	JSON array of the published posts for all following companies and students.	Fetches a list of student posts from students the company is following

/companyFeed/{postId}	POST	commentary: <string>,
		visibility: “public” | “private”,
		distribution: {
		feedDistribution: “mainFeed”,
		share: <string>
		},
		content: {
		article: {
		source: <string>,
		title: <string>,
		description: <string>
		}
			JSON List of viewable Posts
			A successful response returns a 201 CREATED HTTP status code and the ID along with a new post in the news feed.
				A successful response returns a 201 CREATED HTTP status code and the ID along with a new post in the news feed.

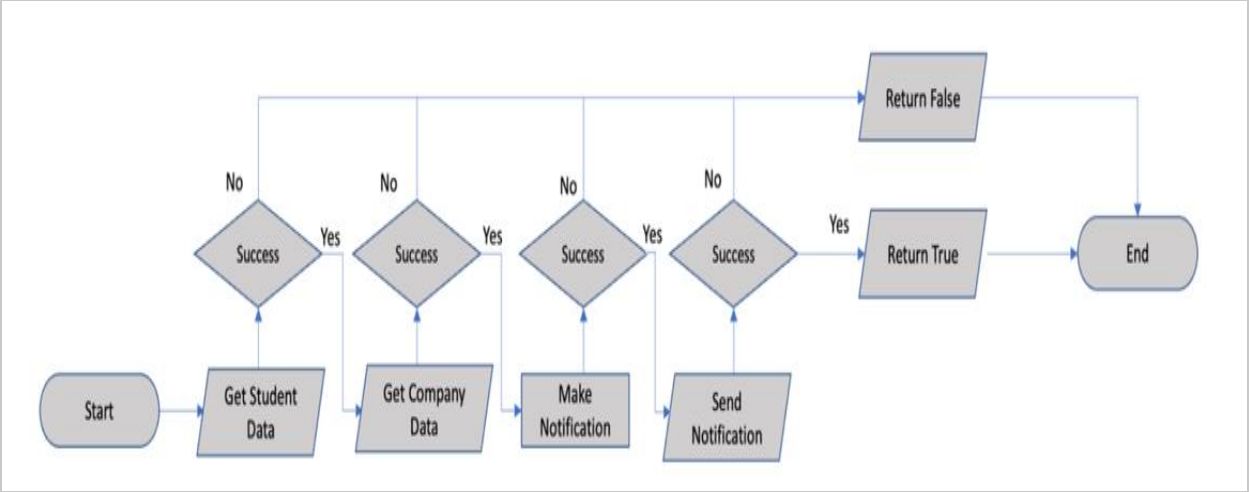
/companyFeed/like/{postId}	POST	commentLikePost: <Boolean>	{result: true | false} 201 Posted HTTP	Like or unlike a post or comment

/companyFeed/{postId}	PUT	commentary: <string>,
		visibility: “public” | “private”,
		distribution: {
		feedDistribution: “mainFeed”,
		share: <string>
		},
		content: {
		article: {
		source: <string>,
			{result: true | false}	Updates a given company post from postId.

Endpoint	Method	Request Body / URL Query Parameters	Response Body	Effect(s) on Servers
		title: <string>, description: <string> }		
/companyFeed/{postId}	DELETE	-	{result: true | false}	Deletes a given company post from the database
/helpfulHint/{messageId}	DELETE	-	{result: true | false}	Remove a hint from the database
/notification/profileUpdate	POST	studentId: {student_id}, companyId: {company_id}	{result: true | false}	Tells the server to send a notification to the company that the student has updated their profile.
/notification/resumeMatch	POST	studentId: {student_id}, companyId: {company_id}	{result: true | false}	Tells the server to send a notification to the company when a student's resume matches 50% of the job post's qualifications.
/notification/profileView	POST	studentId: {student_id}, companyId: {company_id}	{result: true | false}	Tells the server to send a notification to the student which company viewed their profile.
/notification/company/post	POST	companyId: {company_id} folllowingUsers: {users}	{result: true | false}	Tells the server to send a notification to all the students following a company when a post is updated or edited.
/notification/company/post	GET	companyId: {company_id} folllowingUsers: {users}	JSON of the notification	Enables each student that is following a company to be able to see the notification sent to them in the web app.
/notification/company/post	PUT	companyId: {company_id} folllowingUsers: {users}	{result: true | false}	Tells the server if a student following a company reads the notification that has been sent to the student.
/notification/company/message	POST	companyId: {company_id} folllowingUsers: {users}	{result: true | false}	Tells the server to send a notification to all the students following a company when a message is sent, updated, or edited.
/notification/company/message	GET	companyId: {company_id} folllowingUsers: {users}	JSON of the notification	Enables each student that is following a company to be able to see the notification sent to them in the web app.
/notification/company/message	PUT	companyId: {company_id} folllowingUsers: {users}	{result: true | false}	Tells the server if a student following a company reads the notification that has been sent to the student.
/message	POST	senderId: {company_id} followingUsers {users} message: {message_text}	{result: true | false}	Tells the server to create and send a message to the inbox of all the students following a company.
/message	GET	messageId {message_id}	JSON of the message	Enables each student that is following a company to be able to see the message sent to them in their web app inbox.

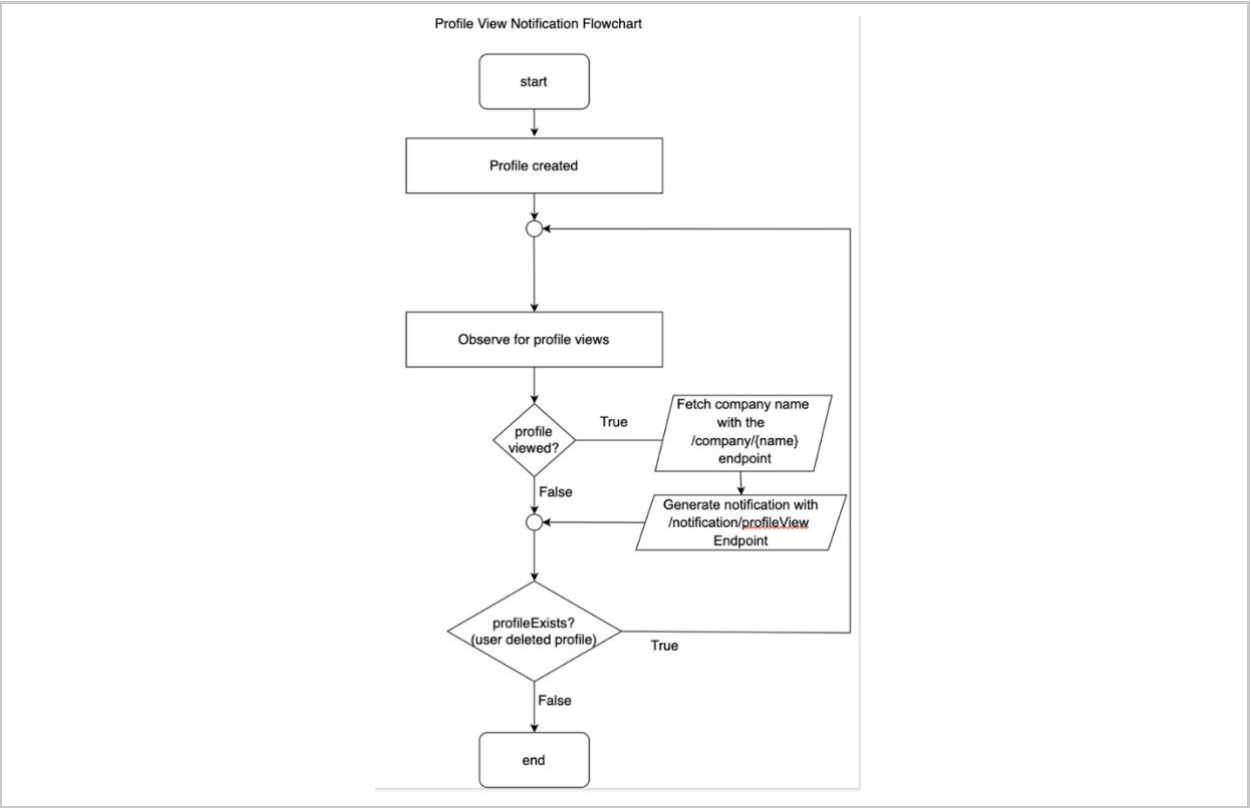
Name	**Backend REST API Endpoints**
Purpose	Lists all the available URIs for the frontend/end users to access.
Description	Representation of the REST API Endpoints, which serve as a simplified bridge between the front and backends, both of which will encapsulate and handle them.
Requirements	2.1 – 3.4
Elements	**Student and Company GET methods:** ‘{name}’, ‘?’, ‘id’ and ‘ ’. These will simply retrieve the appropriate data as described above for their respective databases.
	Student and Company POST, PUT and DELETE methods: ‘id’ and ‘ ’. These are the ‘SET’ methods, but specialized. They will write or modify the data in their respective databases.
Referenced By	2.2.17 – 2.2.20, 2.2.25, 2.3.38, 2.3.49, 2.3.53 – 2.3.56, 2.3.58 – 2.3.59, 2.3.73, 2.3.76
Viewpoint	API Endpoint Table

2.2.16. Profile Update Notification Flowchart



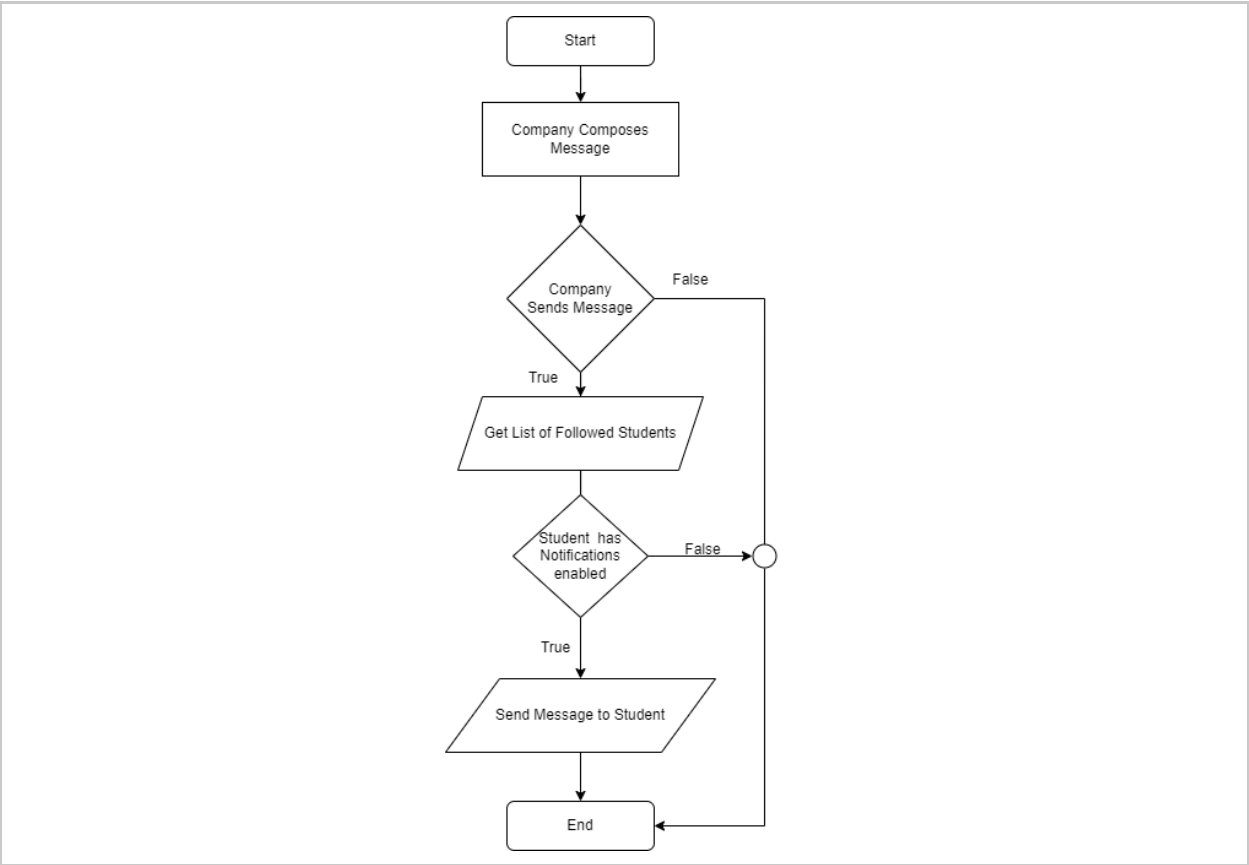
Name	**Profile Update Notification Flowchart**
Purpose	The working flow of the profile update notification.
Description	The flow chart indicates how the data that is provided is used when called, the API retrieves the needed student user profile and company profile page data. The data is used to create and send notifications. At any point there is a failure, the API returns a failure message. If successful, return a success message.
Requirements	2.2.4, 3.2.1, 3.2.2, 3.2.3, 3.2.5
Elements	**Oval Symbols:** Represent the start and end of the process.
	Diamonds: Represent a decision that is being made. Branching paths handle the outputs.
	Parallelogram: Represents input and output.
	Arrow: Represents the direction to read.
	Arrow with Label: Only go that direction if the decision made matches the text.
Referenced By	2.2, 2.2.10, 2.2.17, 2.2.18, 2.2.19, 2.2.20
Viewpoint	Flowchart

2.2.17. Profile View Notification Flowchart



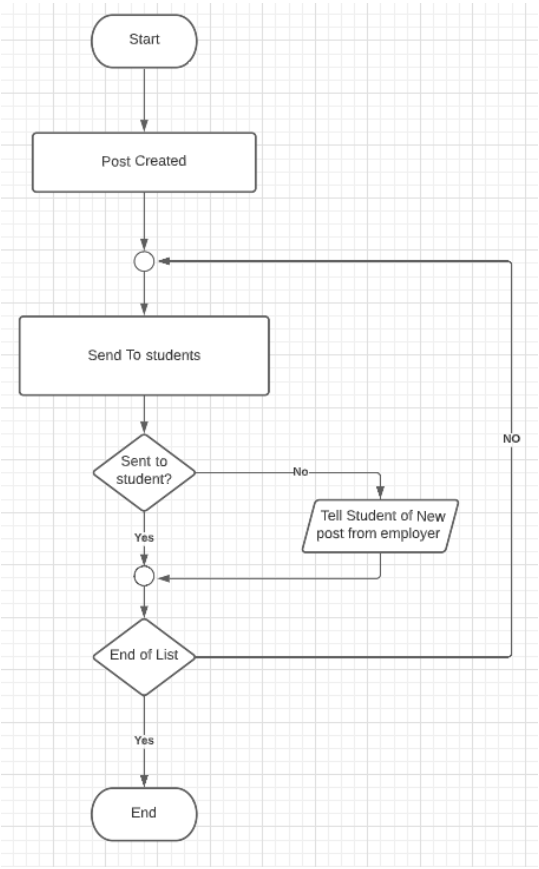
Name	**Profile View Notification Flowchart**
Purpose	Describe the basic logic for user profile view notification trigger.
Description	When the profile is viewed by a company profile, both "/company/{name}" and "notification/profileView" endpoints are applied. The viewNotification monitors a profile for views, it begins when a student profile is created and ceases when the profile is deleted from the server.
Requirements	3.2.5
Elements	**Oval Symbols:** Represent the start and end of the process.
	Rectangles: The processes of the algorithm named, including watching for profile views and **retrieving:** • An observer is used for each student profile to determine when it is being viewed by a company representative.
	Diamonds: Decisions that dictate which processes are ran: • If profileViewed is True, the endpoint API generates a notification and will be called • If the profile still exists, keep the observer active.
	Parallelograms: inputs and outputs: the endpoints being called.
	Circles: Connection points
Referenced By	2.2
Viewpoint	Flowchart

2.2.18. Company Notification to Followers Flowchart



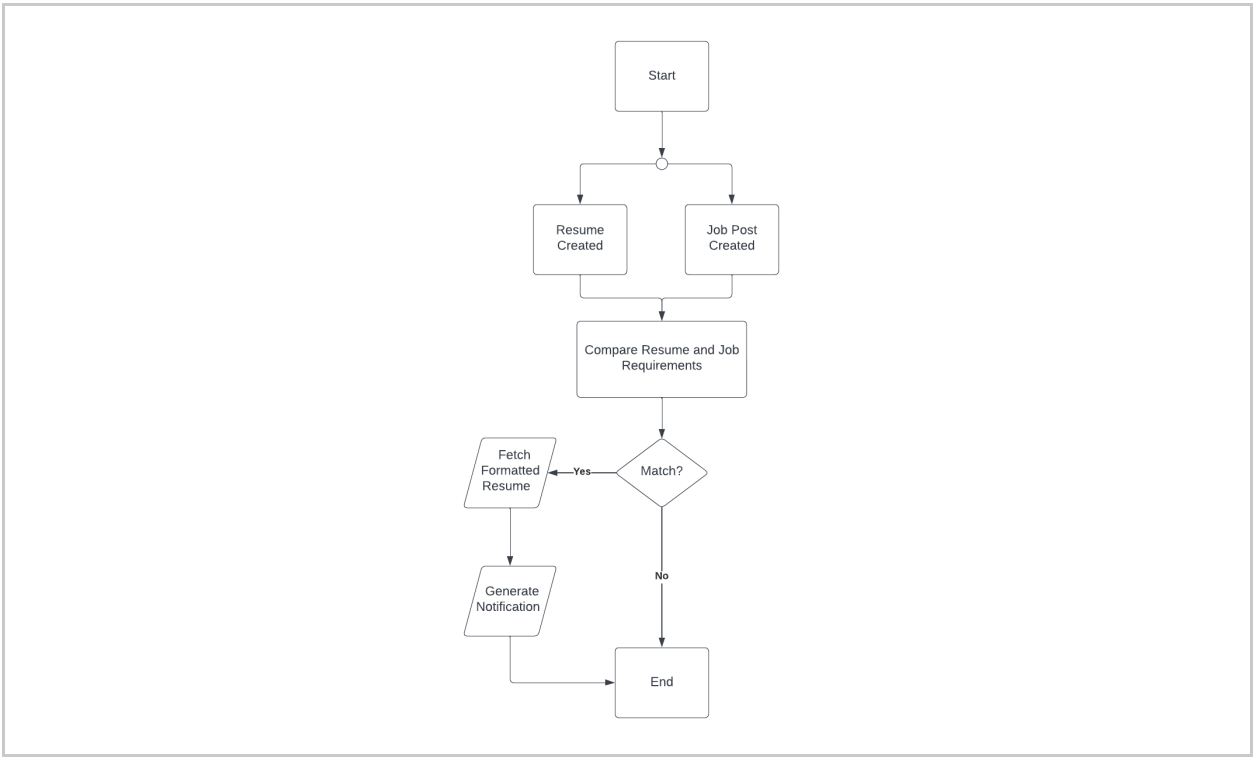
Name	**Company Notification to Followers**
Purpose	The flowchart describes the basic logic for a company to send a notification to their followers
Description	Company representatives will be able to send a notification push to all student user followers after a new post is made is made on the company profile page. .
Requirements	2.2.2
Elements	**Oval Symbols:** Represent the start and end of the process.
	Rectangles: The processes of the algorithm named
	Diamonds: Decisions that impact which processes are ran:
	Parallelograms: inputs and outputs: the endpoints being called.
	Circles: Connection points
	Oval Symbols: Represent the start and end of the process.
Referenced By	2.2.2
Viewpoint	Flow Chart

2.2.19. Company Notification when Post Created Flowchart



Name	**Company Notification when Post Created Flowchart**
Purpose	The flowchart describes the basic logic for sending notifications when a Company creates a Post
Description	Once a post is created, the system will send a notification to the users following the Company. It will confirm the notification status to confirm that the user has received a notification; if not, it will send one; if it has, it will confirm the status of the transmission and the location in the queue. if it is at the end of the list and will repeat if it is not.
Requirements	2.2.2, 3.2.1, 3.2.2, 3.2.7
Element	**Oval Symbols:** Represent the start and end of the process.
	Rectangles: The processes of the algorithm named, including watching for profile views and retrieving: • An observer is used for each Company to see when a post has been created
	Diamonds: Decisions that impact which processes are ran: • If profileViewed weighs out to True, the endpoint API for generating a notification will be called • If the profile still exists, keep the observer active.
	Parallelograms: inputs and outputs: the endpoints being called.
	Circles: Connection points
Referenced By	2.2.5, 2.2.10, 2.2.16, 2.2.27
Viewpoint	Flow Chart

2.2.20. Resume Match Company Notification



Name	**Resume Match Company Notification**
Purpose	Demonstrate logic for generating a resume match notification for company job postings
Description	When either a new resume or new job posting is created, the system will compare the job requirements with the student qualifications found in their resume. If at least 50% of the job requirements are met, a notification will be generated with the resume information.
Requirements	2.2, 2.3.5, 2.3.6, 2.3.7
Elements	**Resume Created/Job Post Created:** Triggers to start the program. Both begin resume search and scraping.
	Compare Resume and Job Requirements: Black box scraping algorithm used to determine requirements match percentage.
	Match: Determines if match percentage meets minimum threshold.
	Fetch Formatted Resume: Retrieves resume from profile and formats it into JSON.
	Generate Notification: Creates notification.
	Endpoints: /resume?... and /jobPost/{job}
Referenced By	2.2.15
Viewpoint	Flowchart

2.2.21. CompanyNewsFeed.createPost()

```
createPost(newPost)
SET createPost = api_dev_key, media_type, require(body), require(title),
require(description), date, distribution, reshareOption,
require(visibility), require(targetEntities), ObjectId,
       require(userID)
SET ContentType = “Content-Type: “application/Json”
SET contentLength = Content-Length: 3000\r\n\r\n
TRY
IF userID IS NOT  userID  THEN
RETURN status 401 “Not Authenticated”
IF body or title, or visibility, or targetEntities, IS empty
Return status 400 “Input cannot be empty”
SET newPost require(body), require(title), require(description), date,
                 distribution, reshareOption, require(visibility),
                 require(targetEntities),      ObjectId, require(userID)
CALL save ()
CALL then(data)
PRINT console log(data)
PRINT status 201
CATCH error PRINT STATUS 500
```

Name	**CompanyNewsFeed.createPost()**
Purpose	To create a new post on the Company’s page.
Description	This is the backend part of creating a news feed post where the user sends their new post to a function that saves the post with the old post in the JSON file.
Requirements	2.4.1
Inputs	There are twelve inputs that are sent to save the edit post function: 1. The API dev key is required to make a post 2. The media type needs to be indicated 3. The way the post needs to be distributed 4. The reshare Boolean needs to be noted 5. The option on who may see the post 6. The target entries need to be indicated 7. The object id will need to be entered 8. The userId will need to be confirmed 9. The post’s placement id in the JSON file. 10. The post’s title. 11. The post’s description. 12. The date the post was added.
Outputs	The new post saved to the JSON file for storage.
Elements	Creates possible new post submission list.
	Checks if user has authentication.
	Checks if the new post’s information has been properly filled out and is not blank.
	Sends the new post to be saved in Json file.
	Send a prompt to the user that the post has been submitted and saved.
Referenced By	
Viewpoint	Pseudocode

2.2.22. CompanyNewsFeed.editPost()

```
FUNCTION save_editPost(json_post_placement_id, new_title, new_description,
new_edit_date)
      OPEN json_file or news post file
      json_dictionary ß json_file
      post_info ß json_dictionary[json_post_placement_id]
      post_info[title] ß new_title
      post_info[description] ß new_description
      post_info[edit_date] ß new_edit_date
      json_dictionary[json_post_placement_id] ß post_info
      json_file ß json_dictionary
```

Name	**CompanyNewsFeed.editPost()**
Purpose	Allows users to edit and save edits to posts
Description	This is the backend part of an edited news feed post where a user sends their newly edited post to a function that replaces and saves the edited post with the old post in the Json file.
Requirements	2.4.1
Inputs	There are five inputs that are sent to save edit post function: 1. The post's placement id in the Json file. 2. The edited post's title. 3. The edited post's description. 4. The date the post was edited. 5. The Json file that the posts are being stored in.
Outputs	The newly edited post saved to the Json file for storage.
Elements	Function save_editPost is called on and sent the proper inputs.
	Opens Json file
	Make a copy of the Json file's dictionary.
	Make a copy of the post from the dictionary using the post's placement id to find it in the dictionary.
	Replace title, description, and edit date in the copy post with new title, description, and date.
	Replace the dictionary's post with a newly changed copy post.
	Replace JSON file with an edited dictionary copy.
Referenced By	2.2.15
Viewpoint	Pseudocode

2.2.23. StudentNewsFeed.createPost()

```
createPost(newPost)
SET createPost = api_dev_key, media_type, require(body), require(title),
       require(description), date, distribution, reshareOption, require(visibility),
       require(targetEntities), ObjectId, require(userID)
SET ContentType = “Content-Type: “application/Json”
SET contentLength = Content-Length: 3000\r\n\r\n
TRY catch {
IF userID IS NOT  userID  THEN
RETURN status 401 “Not Authenticated”
IF body or title, or visibility, or targetEntities, IS empty
RETURN status 400 “Input cannot be empty”

SET newPost require(body), require(title), require(description), date,
       distribution, reshareOption, require(visibility), require(targetEntities),
         ObjectId, require(userID)
CALL save ()
CALL then((data)
PRINT console log(data)
PRINT status 201
CATCH error PRINT STATUS 500
```

Name	**StudentNewsFeed.createPost()**
Purpose	To create a new post
Description	This is the backend part of creating a news feed post where the user sends their new post to a function that saves the post with the old post in the JSON file it is contained in.
Requirements	3.4.2
Inputs	There are twelve inputs that are sent to save the edit post function: 1. The API dev key is required to make a post 2. The media type needs to be indicated 3. The way the post needs to be distributed 4. The reshare Boolean needs to be noted 5. The option on who may see the post 6. The target entries need to be indicated 7. The object id will need to be entered 8. The userId will need to be confirmed 9. The post’s placement id in the JSON file. 10. The post’s title. 11. The post’s description. 12. The date the post was added. The JSON file that the posts are being stored in.
Outputs	The new post saved to the JSON file for storage.
Elements	Creates possible new post submission list.
	Checks if user has authentication.
	Checks if the new post’s information has been properly filled out and is not blank.
	Sends the new post to be saved in Json file.
	Send a prompt to the user that the post has been submitted and saved.
Referenced By	2.2.14
Viewpoint	Pseudocode

2.2.24. NotificationListener.fetchProfiles()

```
NotificationListener::fetchProfile(userID array)
    profiles <- [] the size of userID

    sqlCon <- createSomeSqlConnectionObject()
    selectString <- "SELECT nm_student, gpa, nm_school, major,
                phone_number, email
                            FROM Student
                            WHERE id_student = ?"
    selectProfile <- sqlCon.prepareStatement(selectString)

    sqlCon.setAutoCommit(false)
    FOR id in UserID
        selectProfile.setInt(1, id)
        result <- selectProfile.executeQuery()
        sqlCon.commit()

        IF result IS NOT empty
            profiles.append(result)

    RETURN profiles
```

Name	**NotificationListener.fetchProfiles()**
Purpose	Retrieves a student profile data from the database.
Description	Profile data is retrieved based on selected user IDs. ASSUMES: The language the backend is implemented in has capabilities for prepared statement creation and execution. Authorization of access controls have been implemented and checked elsewhere. Another function is responsible for the presentation of the fetched profiles.
Requirements	2.2.1, 2.3.1
Inputs	**userID[]:** a collection of userIDs for accessing the correct rows in the database
Outputs	**Profiles:** a collection of data objects retrieved from the database containing the requested columns
Elements	**sqlCon:** an SQL connection to allow the generation and execution of SQL statements in the database
	prepareStatment: precompiled SQL statement to provide confidentiality security from anything hidden in the userID input
	i**d:** individual IDs from userID
	result: data collection of all the selected columns from the given ID row
Referenced By	2.2.5
Viewpoint	Pseudocode

2.2.25. NotificationListener.fetchUsername()

```
NotificationListener::fetchUsername(int userID):

    studentResponse = GET "/student/{userID}"
    IF studentResponse IS NOT empty:
        RETURN "/student/{userID}"
    ELSE:
        companyResponse = GET "/company/{userID}"
        IF companyResponse IS NOT empty:
            RETURN "/company/{userID}"
    RETURN ""
```

Name	**NotificationListener.fetchUsername()**
Purpose	Determine the URL pointing to the details of a user
Description	Provided a userid (as an int), queries the backend for the userID and returns a URL pointing to that user's information. ASSUMES: All ids are unique for students and companies.
Requirements	2.2.3
Inputs	**userID:** an integer representing a user's id. Maps to at most one user.
Outputs	A URL in string format pointing to all that user's information, including their username. OR an empty string if no user is found with that ID.
Elements	**Endpoint:** "/student/{userID}" Used to fetch all the information on a student with the given userID. May return an error condition if no such user exists.
	Endpoint: "/company/{userID}" Used to fetch all the information on a company with the given userID. May return an error condition if no such user exists.
Referenced By	2.2.25 2.2.27
Viewpoint	Pseudocode

2.2.26. NotificationListener.fetchResumes()

```
fetchResumes( ResumeList, desiredResumes )
    retrievedResume ← []

    FOR i ← 0 ... desiredResumes.end()
        ThisResume = ResumeList.ResumName.find(desiredResumes(i))
        retrievedResume ← retrievedResume + ThisResume
        i ← i+1

    RETURN retrievedResume
```

Name	**NotificationListener.FetchResumes()**
Purpose	The purpose is to fetch a sub list of resumes from the overall larger list that exists.
Description	Allows a manager to search for a list of Resumes by name from the entirety of the resume
Requirements	2.2.5
Inputs	**ResumeList:** The full list of Resumes received for this application **DesiredResumes:** the sub list of names that are in the resumeList
Outputs	A list of the full resumes
Elements	**ResumeList.ResumeName :** retrieves the name field from a resume object
Referenced By	2.2.42, 2.2.52
Viewpoint	Pseudocode

2.2.27. INotification.send()

```
INotification::send ()
    sqlConnectionObject ← createSomeSqlConnectionObject()
    INSERT (next_id, self.message)  into database's "push_notifications" table
    FOR recipient IN self.recipients:
        UPDATE (notification_IDs = notification_IDs.append(next_id)) in database's
"student" table WHERE student_id == recipient
        UPDATE (notification_IDs = notification_IDs.append(next_id)) in database's
"company" table WHERE company_id == recipient
        IF update was not successful:
            self.success = False
        ELSE:
            self.success = TRUE
```

Name	**INotification.send()**
Purpose	Sends a notification to each recipient.
Description	Updates the database with a notification for each user in the INotification's recipients. If successful, it will set this INotification's success member variable to true. **ASSUMES:** All user ids are unique across students and companies. INotification's recipient's member variable is a list of userIDs (int), not usernames (string).
Requirements	2.2.2
Inputs	None
Outputs	None
Elements	**sqlConnectionObject:** Some object used to connect to the SQL databases in the backend.
	next_id: The next available id in the push_notifications table, as an int.
	self.message: This INotification's message, as a string.
	push_notifications: A SQL database containing a collection of push_notifications.
	self.recipients: This INotification's collection of recipients, each as an integer user id.
	notification_IDs: A number assigned to a student or company in a database, the user's collection of integer ids of push_notifications that were sent to them.
	student: The student SQL database.
	company: The company SQL database.
	Self.success: This INotification's indicator of whether it was successful or not.
Referenced By	2.2.14
Viewpoint	Pseudocode

2.2.28. JobPost.editPost()

```
editPost(updatedElements)
 jobName         ←  updatedElements[0]
 jobDescription  ←  updatedElements[1]
 category        ←  updatedElements[2]
 reqSkills       ←  updatedElements[3]
 tags            ←  updatedElements[4]
 compensation    ←  updatedElements[5]
 location        ←  updatedElements[6]
 expirationDate  ←  updatedElements[7]
 UPDATE (name, description, skills, tags, compensation, location, expiration) in
database's "jobs" table WHERE job_id == id
```

Name	**JobPost.editPost()**
Purpose	The purpose of EditPost is to update a job posting.
Description	Describes how the job post is updated.
Requirements	2.5.1, 2.5.9
Inputs	updatedElements: an array of attribute values which will overwrite current attributes.
Outputs	None
Elements	jobName: The name of the job.
	jobDescription: A description of the job.
	category: A listing of the type of Job this is for Engineering, Sales, etc...
	reqSkills: A list of skills required for the job.
	tags: A list of keywords associated with this job post.
	compensation: How an employee would benefit from this job.
	location: Where the job is located.
	expirationDate: When this job posting is no longer valid.
	“jobs” table: A database table containing information on job postings.
Referenced By	2.2.14
Viewpoint	pseudocode

2.2.29. JobPost.deletePost()

```
deletePost( Jobposting )
    Jobposting.CompanyName ← NULL

    Jobposting.JobTitle ← NULL

    FOR NOT i ← 0 ..... Jobposting.RequiredSkills.end
        Jobposting.RequiredSkills[i] ← NULL

    Jobposting.JobDescription ← NULL
    Jobposting.hours ← NULL
    Jobposting.position ← NULL
    Jobposting.type ← NULL
    Jobposting.compensation ← NULL
    Jobposting.location ← NULL
    Jobposting.contact ← NULL
    Jobposting.information ← NULL
    Jobposting.links ← NULL

    REMOVE Jobposting
```

Name	**JobPost.DeletePost()**
Purpose	Remove a post and all its values from the JSON file.
Description	Delete Post sets all values to Null before Removing the post in its entirety
Requirements	2.5.1
Inputs	**Jobposting:** The post that the user is attempting to delete
Outputs	None
Elements	**RequiredSkills:** A list of the required skills from deemed by the poster
Referenced By	2.2.66
Viewpoint	Pseudocode

2.2.30. JobPost.getJobInfo()

```
getJobInfo(jobPosting)
   GET jobData from data store
   RETURN jobData
```

Name	**JobPost.getJobInfo()**
Purpose	Retrieve relevant data
Description	Retrieves Job Posting Data from the data store
Requirements	2.2.
Inputs	**JobPosting:** The element that the user wishes to view
Outputs	**JobData:** An object containing all the info on the job posting
Elements	**N**one
Referenced By	2.3.82
Viewpoint	Pseudocode

2.2.31. Application.apply()

```
apply(query)
    IF query Not has JobId
        RETURN Error("Application should have jobId. Please reapply.")

    savedApplication ← saveApplication(
        new Application (
            resume.getId(),
            query.getJobId(),
            this.studentUserTag
            )
        )

    applicationNotification   ←  NotificationTemplate
                            .getApplicationNotificationTemplate()

    job ← JobService.getJobById(query.getJobById)
    student  ←  StudentService.getStudentByUserTag(this.studentUserTag)

    companyEmail ← job.getJobInfo().getContactInfo().getEmail()

    preparedMessage ← format(applicationNotification,
      companyEmail,
      student.getFirstName(),
      student.getLastName(),
      resume.getLink()
    )
    preparedMessage.submit()
    RETURN savedApplication
```

Name	**Application.apply()**
Purpose	Submit the request for position
Description	Saves the application, submits the application to the job post, sends a notification to the company, and sends an email to the company.
Requirements	2.1.2
Inputs	Payload with JSON body that has certain fields such as jobId and additional information
Outputs	The main output is submitted request with id that it might be fetch by users involved in the processing flow.
Elements	Query is a payload given by external request
	JobService: Service layer that is responsible for the Job entity management.
	StudentService**:** Service layer that is responsible for the Student entity management.
	this: a reference to this Application object.
Referenced By	2.2.14
Viewpoint	Pseudocode

2.2.32. Application.withdrawApplication()

```
withdrawApplication(query)
    IF query NOT CONTAINS applicationId
        RETURN Error("Application should have an applicationId. \
            Please try again later.")

    deletedApplication ← deleteApplication(query.getApplicationId())

    RETURN deletedApplication
```

Name	**Application.withdrawApplication()**
Purpose	Begin the process of withdrawing the application from a job posting.
Description	This method allows a user (who previously applied for the position) to withdraw his/her application to a job posting.
Requirements	2.1.2
Inputs	JSON payload that holds fields such as applicationId and other required for the processing.
Outputs	Deleted application.
Elements	Query – is a payload provided by user request that holds necessary data pass by to the job posting service layer.
Referenced By	2.2.14
Viewpoint	Pseudocode

2.2.33. Application.deleteApplication()

```
deleteApplication(applicationId)
    If Not exist applicationId
        return Error("No application Id was provided. Please Provide
applicationId")
    RETURN applicationService.delete(applicationId)
```

Name	**Application.deleteApplication()**
Purpose	Deactivate the submitted application
Description	Sends a request to the target Service to deactivate the active application
Requirements	2.1.2
Inputs	ApplicationId is assigned to an application that will be used to be deactivated by the system.
Outputs	Deleted/Deactivated application request
Elements	ApplicationService represents a Service layer for the target
	Applicationid – is an item id that corresponds to the id in the database.
Referenced By	2.2.32
Viewpoint	Pseudocode

2.2.34. Application.saveApplication()

```
saveApplication(application)
    If not exist application
        return Error("No Application was provided. Please provide the
application")
    return applicationService.save(application)
```

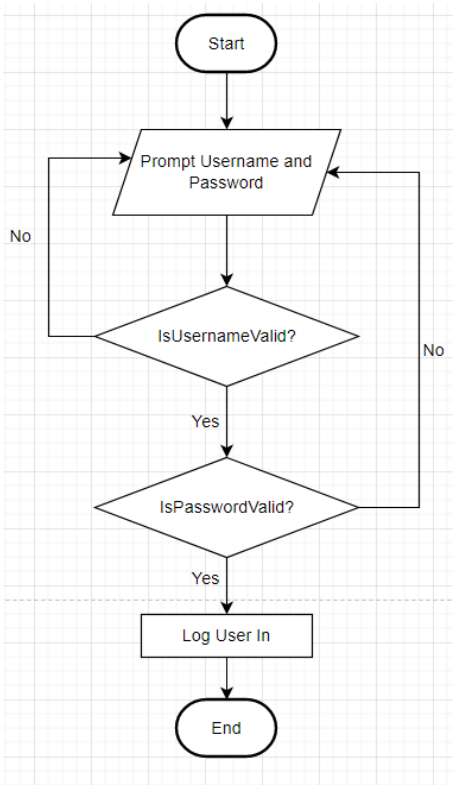
Name	**Application.saveApplication()**
Purpose	Creates an application request in the DB and provides the bindings between a user resume and a job posting position
Description	Creates a reference in the DB by creating an application request in the DB with reference to a resume, user, and a target position
Requirements	2.1.2
Inputs	**Application:** is an entity of the DB table where the application will be saved with all fields that need to be filled in from a job applicant.
Outputs	A saved entity with a DB id assigned
Elements	ApplicationService that contains the main company logic and produces the request for the repository layer.
	Application is a domain/model that represents the record that holds the references between other models such as student, resume, and job.
Referenced By	2.2.31
Viewpoint	Pseudocode

2.2.35. CompanyRepresentative.editPage()

```
editPage(Data, choice)
IF Check_authorized(user)
             (Choice == add)
                    addData(Data)
ELSE IF (Choice == edit)
             editData(Data)
ELSE IF (Choice == remove)
             RemoveData(Data)
UpdatePage()
```

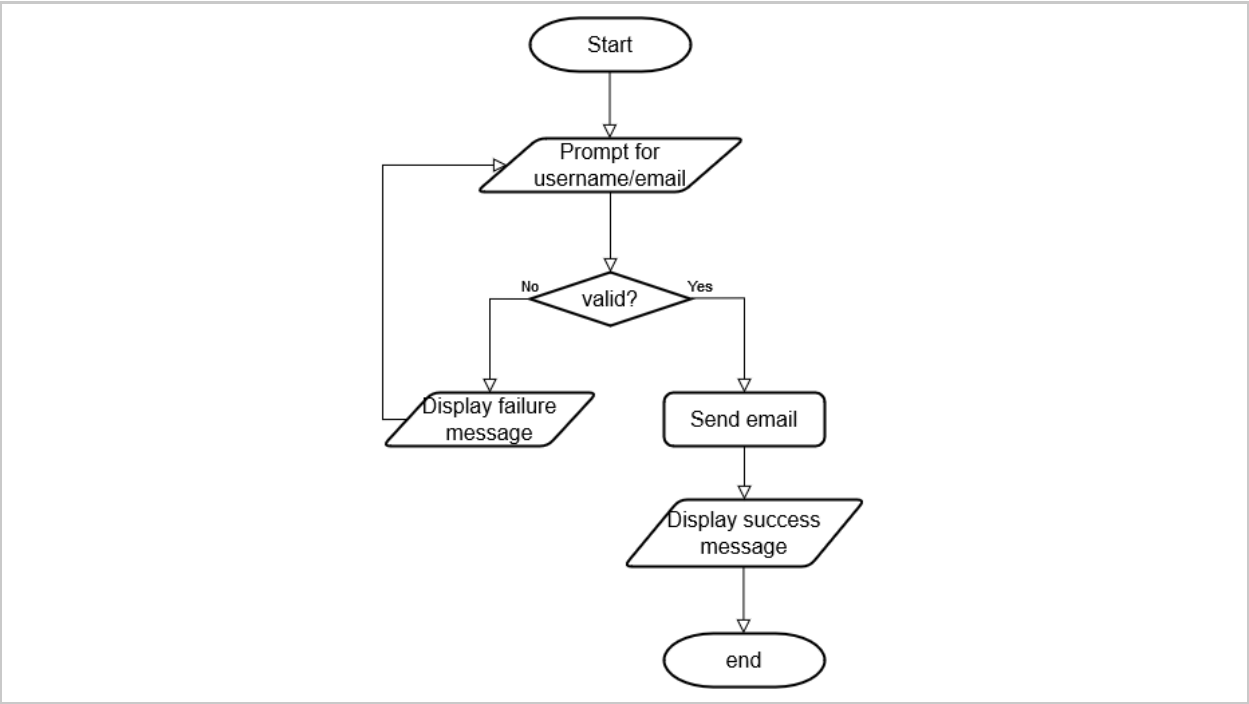
Name	**CompanyRepresentative.editPage()**
Purpose	Allows a company representative to add, alter, and remove any data on the profile page
Description	Checks if the user is an authorized company representative before adding, removing, or editing the data on the page based on the user's prior input
Requirements	2.1.3.10
Inputs	**d**ata: the change to be made to the page choice: whether the user is requesting an addition, removal or edit
Outputs	The altered Data sent to the Backend to be saved to the database and displayed in the browser.
Elements	Company profile page
Referenced By	2.2.15
Viewpoint	Pseudocode

2.2.36. LoginPage.login()



Name	**LoginPage.login()**
Purpose	Verify the user's login information, and allows them access into the system, only if the information is valid
Description	Allows a user to attempt to login to the system, the user is prompted to input their username and password. The username, and the password will be verified before the user has access to their account
Requirements	2.1.3.3, 2.1.3.10, 3.2.5
Inputs	Username and Password
Outputs	Boolean (successful login or not)
Elements	**Username**: Unique username for the user. Input by the user
	Password: Unique password for the user, input by the user
	IsUsernameValid: Verifies that the username is in the database – Must happen before Password check
	IsPasswordValid: Verifies that the password is correct for the specific Username/User - Must happen AFTER a correct username is entered
	LogUserIn: This logs the user into their account. This only happens if both username and password are verified
Referenced By	2.2.14
Viewpoint	Flowchart

2.2.37. LoginPage.forgotPassword()



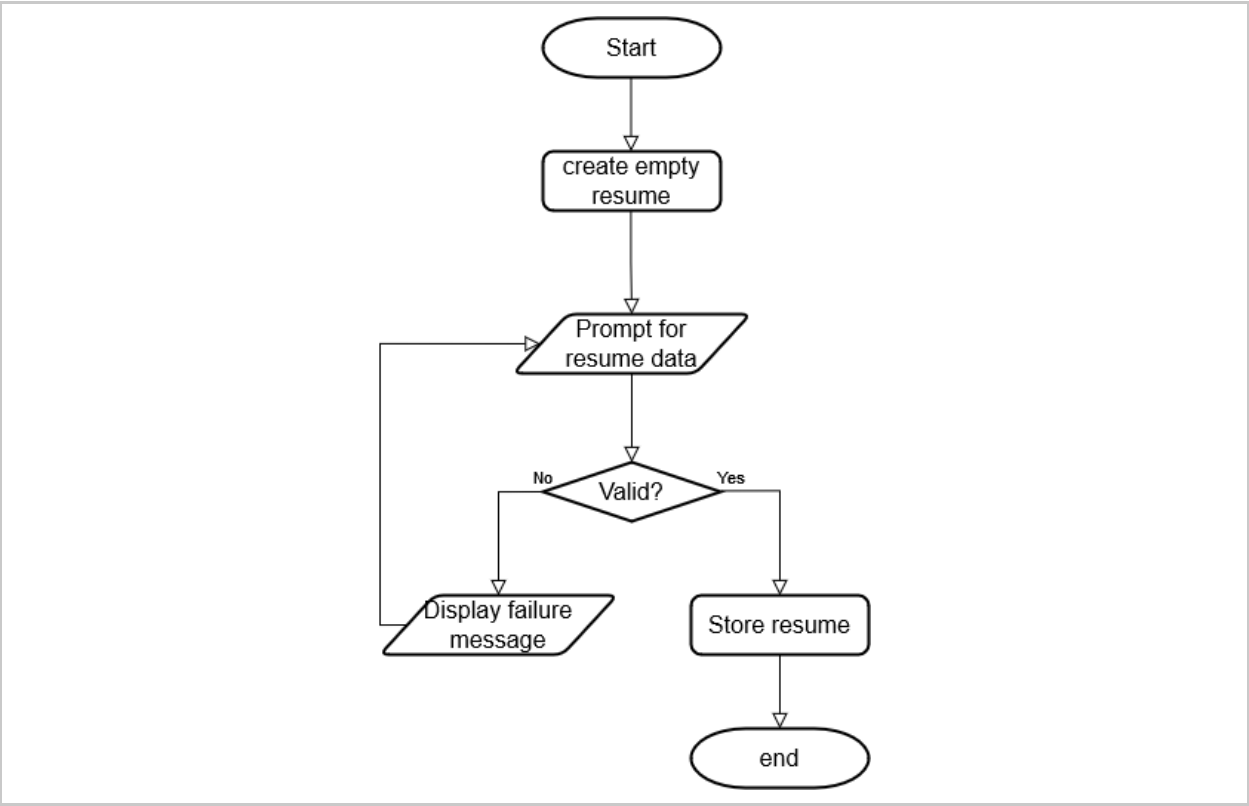
Name	**LoginPage.forgotPassword()**
Purpose	Show the program flow for the forgot password function
Description	When the user has forgotten their password, they are given the option to input their email or associated username, which will send the user an email with instructions on how to recover or change the password
Requirements	2.1.3.3, 2.1.3.10, 3.2.5
Inputs	Username, Email address
Outputs	Notifications, emails
Elements	Prompt for username/email: Prompt for user to provide a username or email for verification prior to password reset or retrieval
	Valid: Verification that the username or email is tied to an existing account
	Display failure message: Output to inform user if no email is sent
	Send email: Automated email is sent to the associated email address if verification passes
	Display success message: Output to user to direct them back to their email for further instructions
Referenced By	2.2.14
Viewpoint	Flowchart

2.2.38. Student.updateInfo()

```
UpdateInfo(elementChanges):
   For it from 0 -> sizeOf(elementChanges)
      SWITCH(it):
         CASE 0: name           <- elementChanges[it]
         CASE 1: studentUserTag <- elementChanges[it]
         CASE 2: university     <- elementChanges[it]
         CASE 3: schoolYear     <- elementChanges[it]
         CASE 4: degreeMajor    <- elementChanges[it]
         CASE 5: contactInfo    <- elementChanges[it]
         CASE 6: userImage      <- elementChanges[it]
```

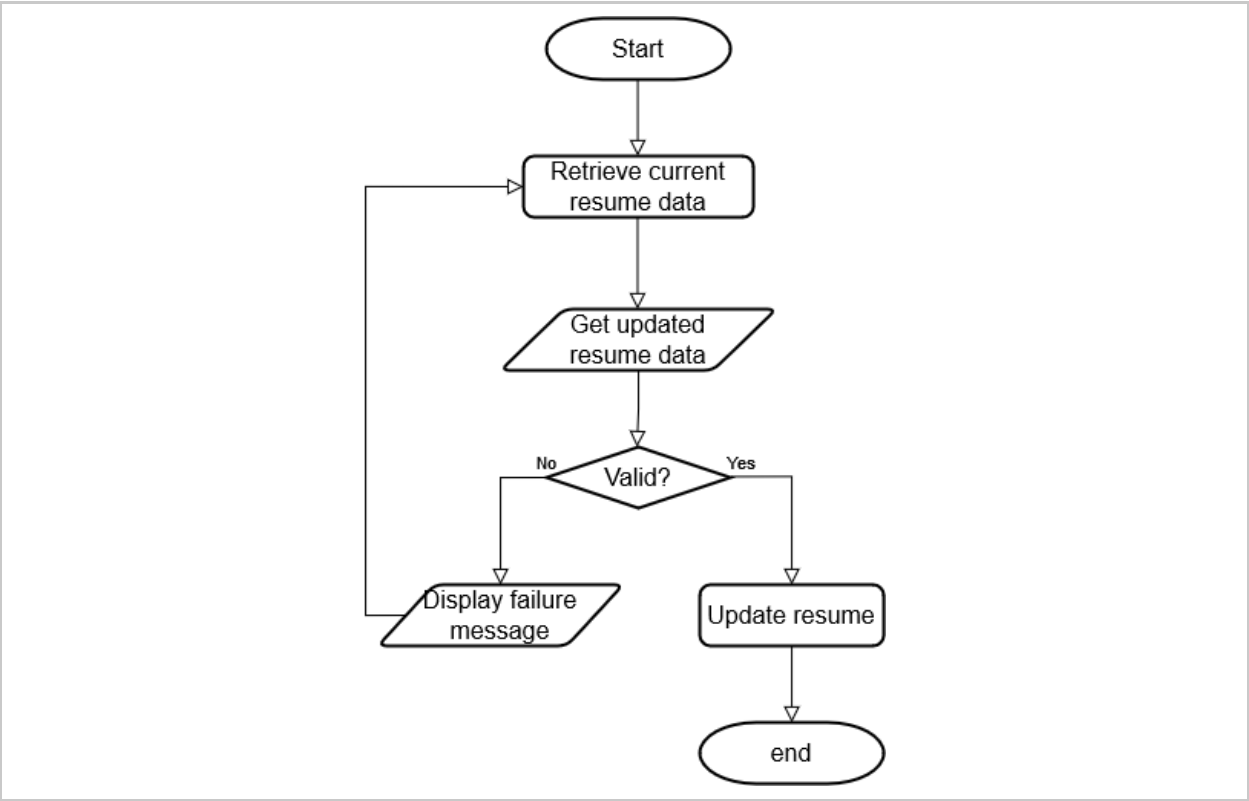
Name	**UpdateInfo.pseudocode()**
Purpose	Enables the student to update the info of their profile
Description	The user can select fields that they would like to update and change their information accordingly.
Requirements	3.1.2.
Inputs	**elementChanges:** Map of changed elements with the key being an integer and the value being the new field
Outputs	**NONE**
Elements	Name
	studentUserTag: The generated ID given to the student by the database.
	university: telling what university the student attends
	schoolYear: What is the student classified as: Freshman, Sophomore, etc...
	degreeMajor: What Major is the student pursuing.
	contactInfo: Way’s to contact the student
	userImage: an Image of the student
Referenced By	2.2.14
Viewpoint	Pseudocode

2.2.39. Resume.addResume()



Name	**Resume.addResume()**
Purpose	Show the program flow for the addResume function
Description	When a new resume is created it starts empty, and after all required fields are filled it will be validated before being stored in the database
Requirements	3.5.1, 3.5.2
Inputs	Resume data
Outputs	New resume instance
Elements	**Create empty resume**: Instantiation of a resume object prior to data storage
	Prompt for resume data: Prompt sent to user to create the new resume
	Valid: Validation to ensure that the resume data has required fields and that values are not invalid
	Display failure message: Output to user to inform of failure in the event the input doesn't pass the verification
	Store resume: Database update with the new resume entry
Referenced By	2.2.13, 2.2.14, 2.3.31, 2.3.32
Viewpoint	Flowchart

2.2.40. Resume.updateResume()



Name	**Resume.updateResume()**
Purpose	Show the program flow for the updateResume function
Description	When a resume is to be updated, the data will be retrieved before edits are made. After the update is confirmed, the changes will be validated before updating the record in the database
Requirements	3.5.1, 3.5.2
Inputs	Existing resume
Outputs	Updated resume
Elements	**Start:** Event triggers process
	Retrieve Current Resume Data: Database query for an existing resume
	Get Updated Resume Data: Prompt to a user resulting in new data
	Valid: A check to make sure the updated information will not negatively affect the rest of the program
	Display Failure Message: Output to user if the data is invalid
	Update Resume: Database update with the new resume data
	End: Returns success message
Referenced By	2.2.14
Viewpoint	Flowchart

2.2.41. Education.addEducation()

```
addEducation(String education, Integer studentId)
    sqlUpdateString <- stringFormat("UPDATE student_info
                                     SET education = '%s'
                                     WHERE student_id = %d;", education, studentId)
    sqlConnectionObject <- createSomeSqlConnectionObject()
    rowsAffected <- sqlConnectionObject.update(sqlUpdateString)
    return sucsess <- toBoolean(rowsAffected)
```

Name	**Education.addEducation()**
Purpose	Add a string containing a description of a student's education to their profile
Description	This updates a row in a SQL database with information about a student's education
Requirements	3.1.2
Inputs	Education: a string containing education information StudentId: An integer representing the student in the database. Most likely the primary key of the student table
Outputs	Success: A Boolean that returns true if the SQL statement was successful.
Elements	SqlUpdateString: A formatted string that includes an update statement with the student's education information and student id inserted into the statement
	SqlConnectionObject: this represents a function from an external library that handles interfacing with an SQL server
	RowsAffected: The number of rows affected when running our statement. This is a standard return for update functions in SQL interface libraries.
	Success: This Boolean is calculated by casting the rowsAffected to a Boolean, as 0 rows affected would mean a failure, and 1 would mean a success. While this will not detect multiple rows affected as a failure, since the primary key is included in the where clause of the SQL statement, it should never be more than one row affected.
Referenced By	2.2.15
Viewpoint	Pseudocode

2.2.42. Education.deleteEducation()

```
deleteEducation(Integer studentId)
    sqlUpdateString <- stringFormat("UPDATE student_info
                                     SET education = ''
                                     WHERE student_id = %d;", studentId)
    sqlConnectionObject <- createSomeSqlConnectionObject()
    rowsAffected <- sqlConnectionObject.update(sqlUpdateString)
    return succsess <- toBoolean(rowsAffected)
```

Name	**Education.deleteEducation()**
Purpose	To delete a student's education information
Description	It removes the student's education information without deleting the whole student
Requirements	3.1.1
Inputs	**Education:** a string containing education information StudentId: An integer representing the student in the database. Most likely the primary key of the student table
Outputs	Success: A Boolean that returns true if the SQL statement was successful.
Elements	**SqlUpdateString:** A formatted string that includes an update statement with the student's education information and student id inserted into the statement. We use update instead of deleting because we only need to delete the information from a single column in a single row, not a whole row. We simply replace the education string with an empty string.
	SqlConnectionObject: this represents a function from an external library that handles interfacing with an SQL server
	RowsAffected: The number of rows affected when running our statement. This is a standard return for update functions in SQL interface libraries.
	Success: This Boolean is calculated by casting the rowsAffected to a Boolean, as 0 rows affected would mean a failure, and 1 would mean a success. While this will not detect multiple rows affected as a failure, since the primary key is included in the where clause of the SQL statement, it should never be more than one row affected.
Referenced By	2.2.15
Viewpoint	Pseudocode

2.2.43. Education.updateEducation()

```
updateEducation(String education, Integer studentId)
    sqlUpdateString <- stringFormat("UPDATE student_info
                                     SET education = '%s'
                                     WHERE student_id = %d;", education, studentId)
    sqlConnectionObject <- createSomeSqlConnectionObject()
    rowsAffected <- sqlConnectionObject.update(sqlUpdateString)
    return succsess <- toBoolean(rowsAffected)
```

Name	**Education.updateEducation()**
Purpose	To update the education information of a student
Description	Updates the education column of a single row in the student's table
Requirements	3.1.2
Inputs	**Education:** a string containing education information **StudentId:** An integer representing the student in the database. Most likely the primary key of the student table
Outputs	**Success:** A Boolean that returns true if the SQL statement was successful.
Elements	**SqlUpdateString:** A formatted string that includes an update statement with the student's education information and student id inserted into the statement
	SqlConnectionObject: this represents a function from an external library that handles interfacing with an SQL server
	RowsAffected: The number of rows affected when running our statement. This is a standard return for update functions in SQL interface libraries.
	Success: This Boolean is calculated by casting the rowsAffected to a Boolean, as 0 rows affected would mean a failure, and 1 would mean a success. While this will not detect multiple rows affected as a failure, since the primary key is included in the where clause of the SQL statement, it should never be more than one row affected.
Referenced By	2.2.15
Viewpoint	Pseudocode

2.2.44. JobExperience.addExperience()

```
addExperience(String experience, Integer studentId)
    sqlUpdateString <- stringFormat("UPDATE student_info
                                     SET experience = '%s'
                                     WHERE student_id = %d;", experience, studentId)
    sqlConnectionObject <- createSomeSqlConnectionObject()
    rowsAffected <- sqlConnectionObject.update(sqlUpdateString)
    return sucsess <- toBoolean(rowsAffected)
```

Name	**JobExperience.addExperience()**
Purpose	To add experience to a student's profile
Description	Updates the experience column of a single row in the student's table
Requirements	3.1.1
Inputs	**Experience:** A string describing the student's experience Student Id: The primary key of the students table, each integer corresponds to a student
Outputs	**Success:** A Boolean returning true if the update was successful and false if it was not.
Elements	**SqlUpdateString:** A formatted string that includes an update statement with the student's experience information and student id inserted into the statement
	SqlConnectionObject: this represents a function from an external library that handles interfacing with an SQL server
	RowsAffected: The number of rows affected when running our statement. This is a standard return for update functions in SQL interface libraries.
	Success: This Boolean is calculated by casting the rowsAffected to a Boolean, as 0 rows affected would mean a failure, and 1 would mean a success. While this will not detect multiple rows affected as a failure, since the primary key is included in the where clause of the SQL statement, it should never be more than one row affected.
Referenced By	2.2.15
Viewpoint	Pseudocode

2.2.45. JobExperience.deleteExperience()

```
deleteExperience()
   connection ← user_database.connect()
   connection.send(
      DELETE FROM experience WHERE id IS currentPageData.selectedExperience.id
   )
```

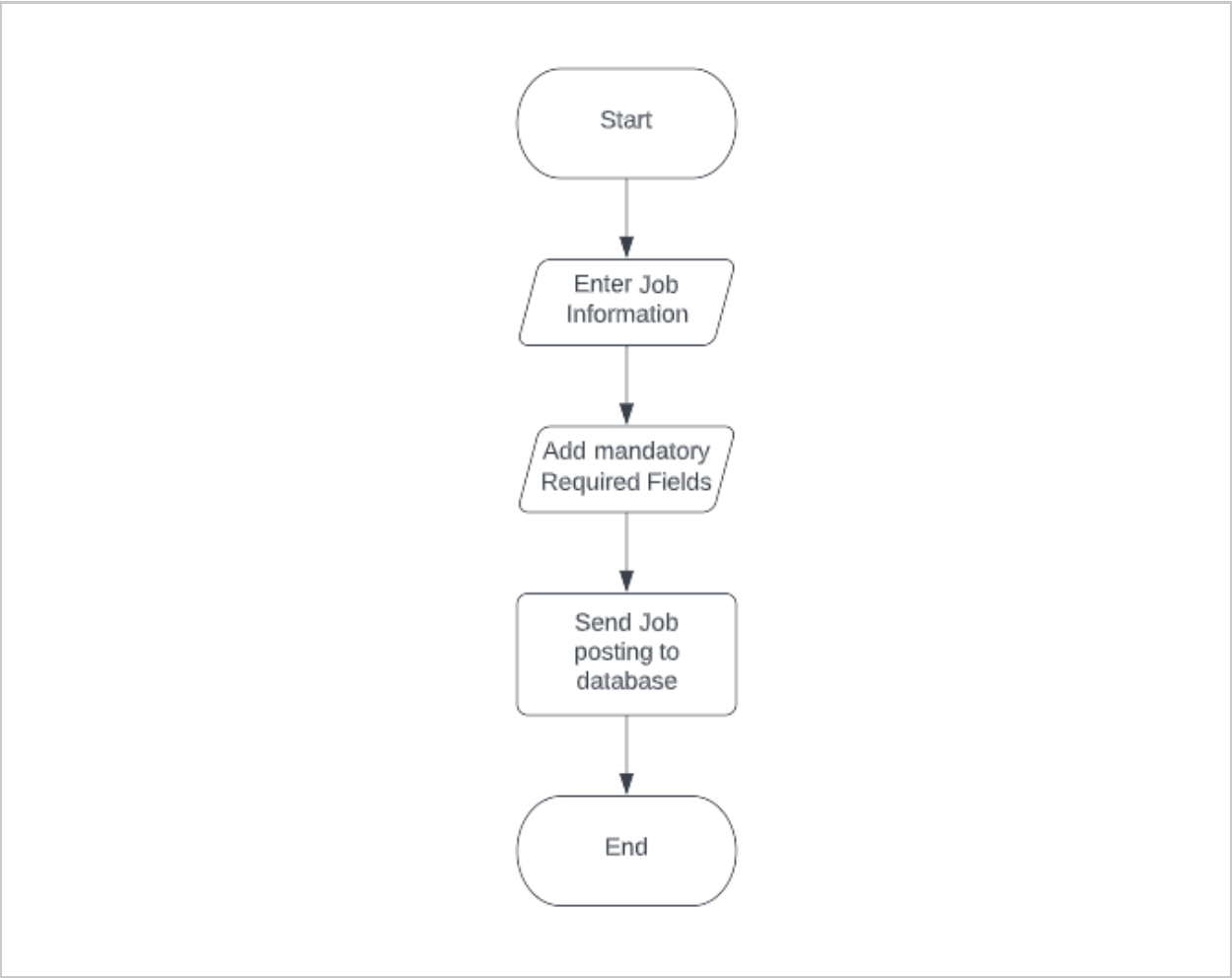
Name	**JobExperience.deleteExperience()**
Purpose	This will be a function that will delete selected experience from the user database.
Description	This function creates a connection to the user database, then runs a query to delete the experience associated with a user.
Requirements	3.1.2
Inputs	None
Outputs	None
Elements	**connection** : the object representing a sql cursor in a functional programming language.
	user_database : a reference to the database which contains the users' experience or probably an experience table.
Referenced By	2.2.15
Viewpoint	Pseudocode

2.2.46. JobExperience.updateExperience()

```
UpdateExperience (education, StudentId)
   UpdateExperienceString <-- UPDATE student_info SET education, AT student_Id
   StudentObject <-- createStudentObject
   rowAffected <-- StudentObject.update(UpdateExperianceString)
   RETURN rowAffected true
```

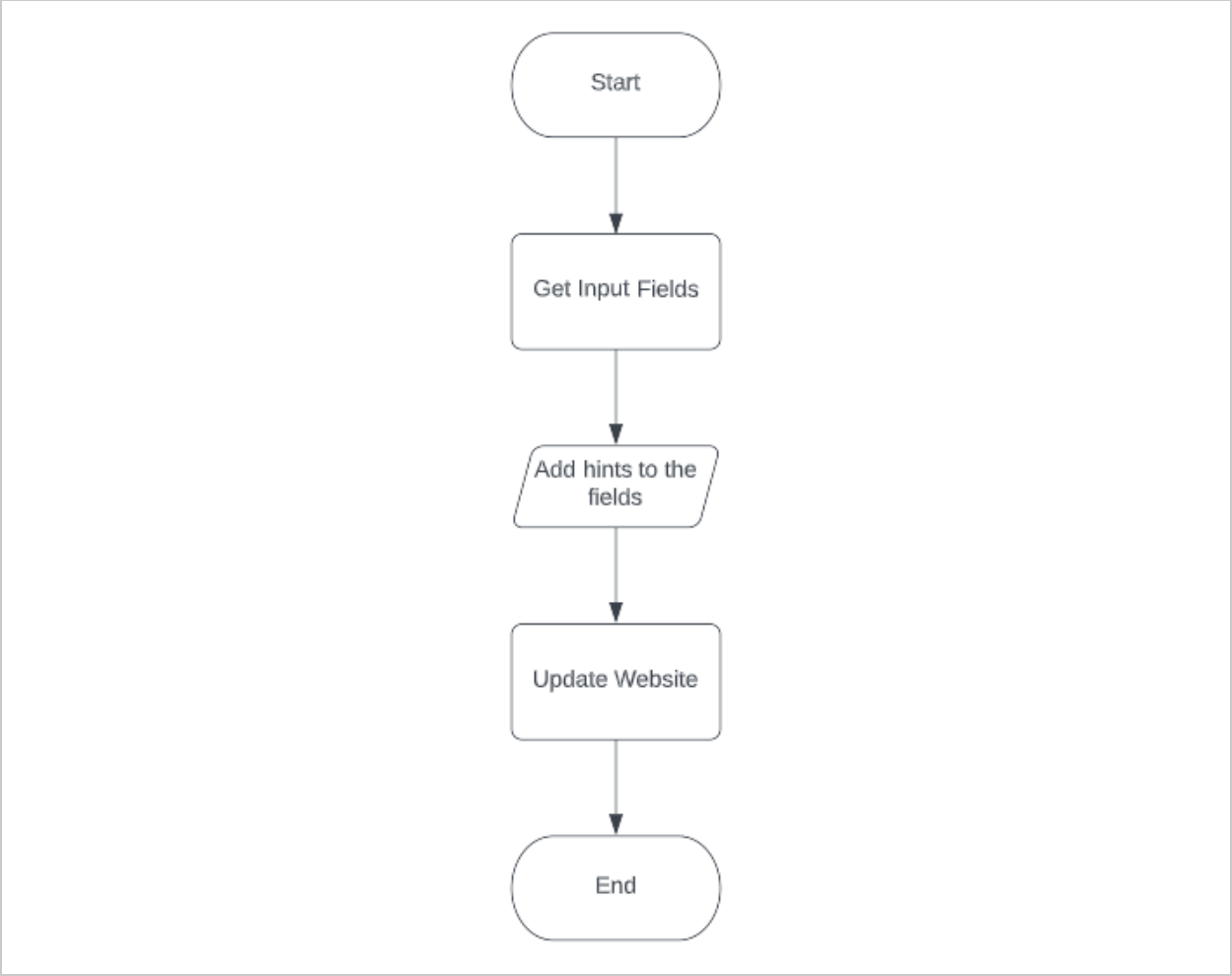
Name	JobExperience.updateExperience()
Purpose	Update the job experiences on the student user profile.
Description	Updates the experience column of a single row in the student's table
Requirements	3.1.2
Inputs	Experience: the studentInfo, Student ID: the unique student ID
Outputs	The rows affected if they were affected it returns true or false
Elements	**UpdateExperianceString:** The string that has the update statement, which updates the student's info at the unique student_id
	StudentObject: The function that is the object that handles the interfacing with the database
	True: Return a bool of true of false depending on if the rows were successfully updated
	RowAffected: Rows affected with the update statement
Referenced By	2.2.15
Viewpoint	Pseudocode

2.2.47. MandatoryFields.addMandatoryField()



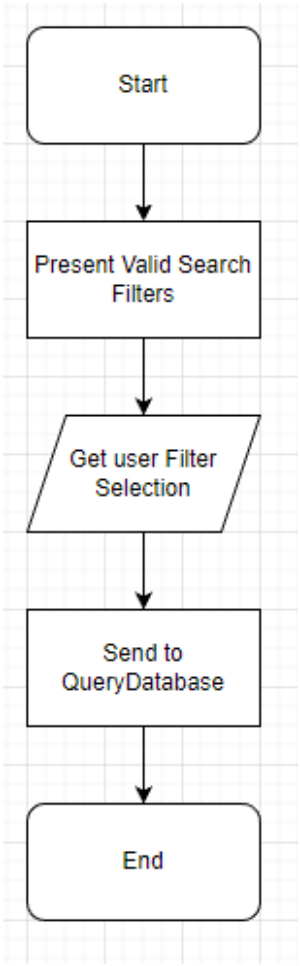
Name	MandatoryFields.addMandatoryField()
Purpose	Make a required input field
Description	This will allow the company representative of a job posting to add a required input field for the student user to fill out.
Requirements	3.5.5.2 - 3.5.5.3
Inputs	Input Field Name
Outputs	A required input field
Elements	Input field
Referenced By	2.2.13
Viewpoint	Flow Chart

2.2.48. HelpfulHint.addHint()



Name	**HelpfulHint.addHint()**
Purpose	Add a hint for input fields
Description	This allows there to be a hint/ help for a given input field
Requirements	3.5.4.2
Inputs	N/A
Outputs	Hints for fields
Elements	**Get Input Fields:** Checks available input fields for that page
	Add Hints to The Fields: Query for hints associated with input fields
	Update Website: Adds hint elements by their associated input field
Referenced By	2.2.14
Viewpoint	Flow chart

2.2.49. CompanySearch.filterBy()



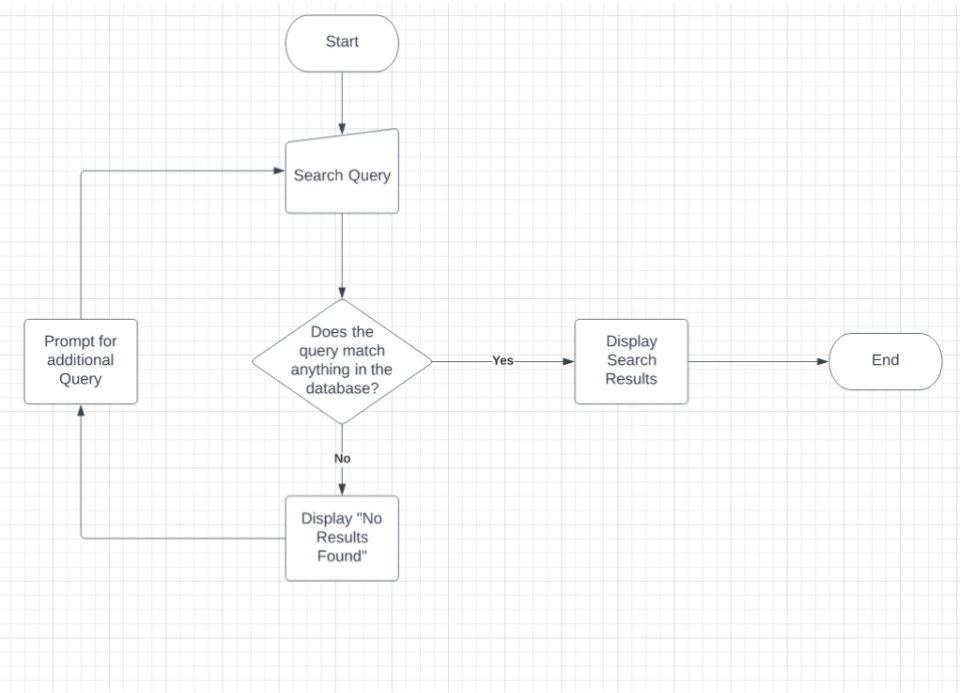
Name	**CompanySearch.filterBy()**
Purpose	Apply filters requested by the company to search for candidates
Description	System will present a list of valid filters options for the present company search. The list may be different depending on the type of company completing the search.
Requirements	2.3.2, 2.3.4, 2.3.5, 2.3.8
Inputs	Filter Options
Outputs	Filtered Query
Elements	**Filter parameters:** Valid list of options for user to filter by
Referenced By	2.2.6, 2.2.50, 2.2.51, 2.3.10
Viewpoint	Flowchart

2.2.50. CompanySearch.searchResumes()

```
searchResumes(select_name, name, select_eduction,education,
select_state,state)
Result ← []
IF name = select_name
      If education = select_education
      If state = slect_state
Result.append(resume)
```

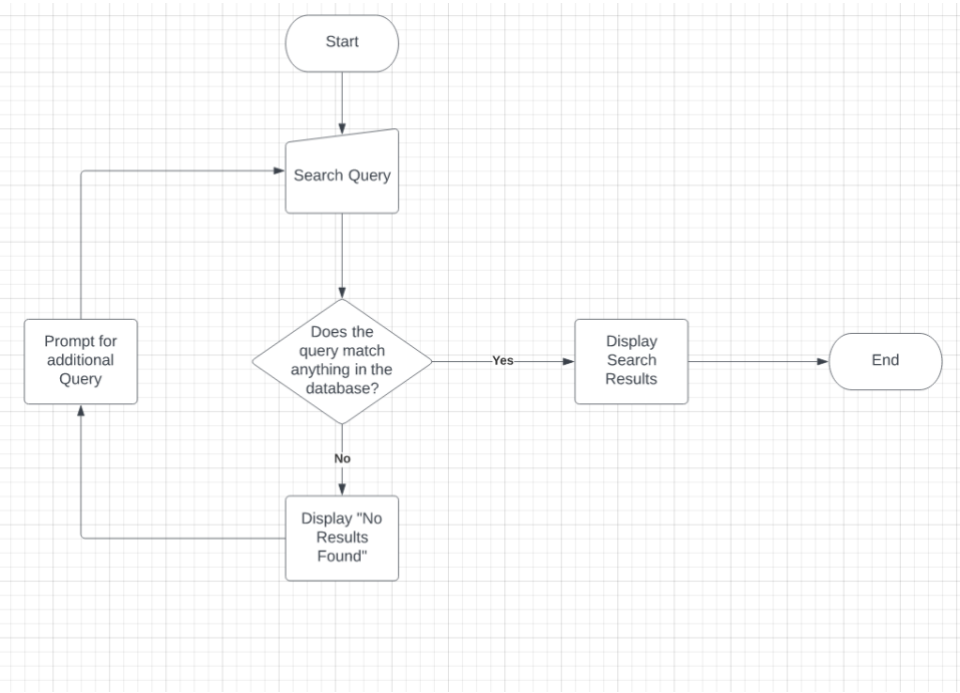
Name	**CompanySearch.searchResumes()**
Purpose	The purpose of CompanySearch is to allow a company representative to search for a matched student user with the filters they have selected
Description	This will display resumes that match with the required filters.
Requirements	2.3.8
Inputs	The query sent by the filter by function
Outputs	Resumes that match the query sent by the filter function
Elements	The query sent by the filterby function
	Resumes in the system
	The resumes returned based on the filter.
	Select variables: they are the parameters being set in the filter function
	Other variables are what names and things along those lines are getting stored in.
Referenced By	2.2.14, 2.2.15
Viewpoint	Pseudocode

2.2.51. CompanySearch.searchProfiles()



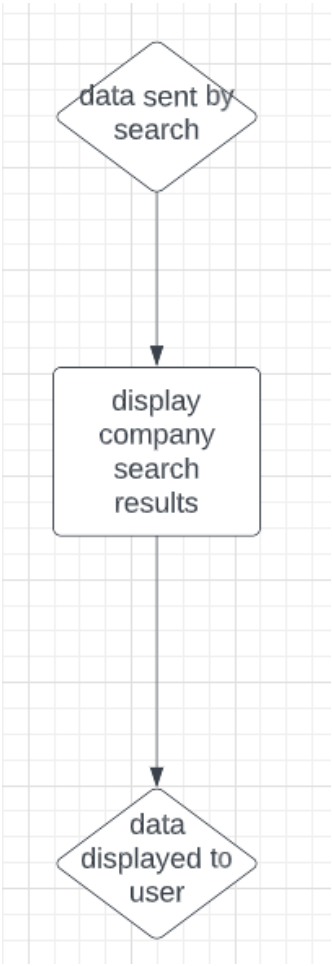
Name	**CompanySearch.searchProfiles()**
Purpose	To search for profiles with a query.
Description	This is a search function that will search the database based on set parameters.
Requirements	2.3.1 – 2.3.8
Inputs	Manual Input from User in form of a query or filter.
Outputs	Returns search results or if null, will display no results and prompt for another search.
Elements	**Manual Input:** Given by the user with a prompt for a query topic or filter
	Actual Database Search: Will search the database looking for the query match results
	Display: Will display search results or display “not found”
	Re-prompt: Will re-prompt the user in case of no possible matches
Referenced By	2.2.14
Viewpoint	Flowchart

2.2.52. CompanySearch.wildcardSearch()



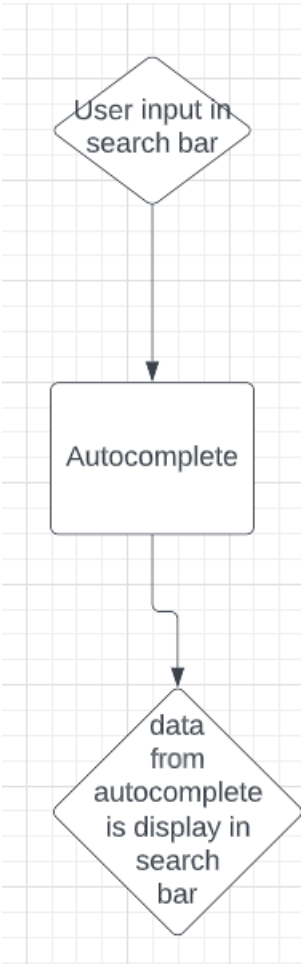
Name	**CompanySearch.wildcardSearch()**
Purpose	To perform a wildcard search
Description	To perform a wildcard search using regular expressions to find matches of a part of a whole query
Requirements	2.3.7
Inputs	Manual Input from User in form of a query or filter.
Outputs	Returns search results or if null, will display no results and prompt for another search.
Elements	**Manual Input:** Given by the user with a prompt for a query topic or filter
	Actual Database Search: Will search the database looking for the query match results
	Display: Will display search results or display “not found”
	Re-prompt: Will re-prompt the user in case of no possible matches
Referenced By	2.2.14
Viewpoint	Flowchart

2.2.53. CompanySearch.displayResults()



Name	**CompanySearch.displayResults()**
Purpose	If a criterion is met, this will display the company the match the user searched. Either job title or company name. This should display all the results that the other function has filtered
Description	This function will get what it is supposed to display and what the filter functions give it.
Requirements	2.3.1 – 2.3.8
Inputs	Information sent by a search function or filter function
Outputs	The packaged information sent to front end
Elements	**Search Data:** Search parameters entered by user, received from front end.
	Display Data: Information retrieved from database, sent to front end.
Referenced By	2.2.6, 2.2.14, 2.2.15, 2.2.51, 2.2.52, 2.3.11
Viewpoint	Flowchart

2.2.54. SearchBar.autocomplete()



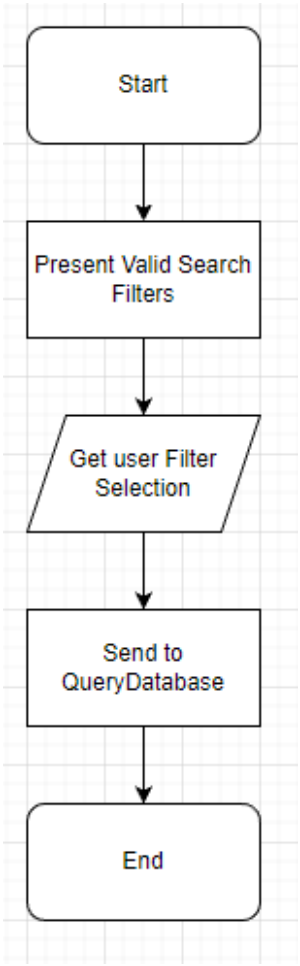
Name	**SearchBar.autoComplete()**
Purpose	To auto complete what the user is typing
Description	This would suggest what the user might be trying to type based off what has already been typed
Requirements	2.3.1 – 2.3.8
Inputs	String typed by user in search bar
Outputs	Suggestions on what the user might be typing
Elements	**Search Bar:** The element that receives the user's input
	Auto Suggestion: Algorithm that offers suggestions based upon the characters typed so far
Referenced By	2.2.14
Viewpoint	Flowchart

2.2.55. SearchBar.displaySearchHistory()

```
displaySearchHistory()
    SET history <-- GET searchHistory
    FOR entry in history
        PUT entry.searchString
        PUT entry.date
```

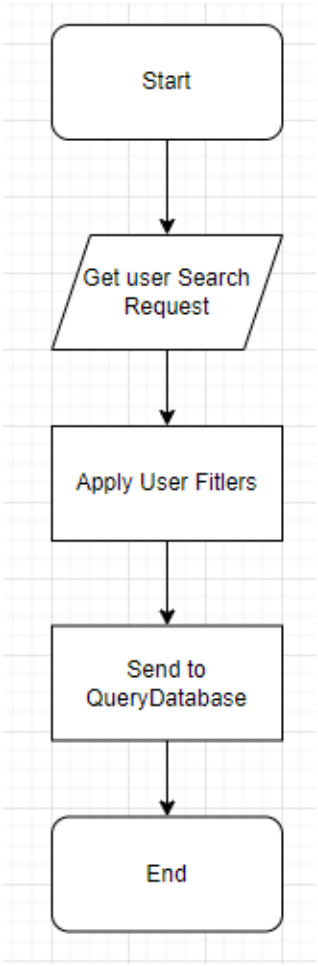
Name	**SearchBar.displaySearchHistory()**
Purpose	Pseudocode for implementing the displaySearchHistory function
Description	When a request is made for a user's search history, this function will retrieve the data for display via this function
Requirements	2.3, 2.3.2, 3.3
Inputs	None
Outputs	Searchstring records
Elements	History: A list of previously queried strings in the search bar
	Entry: A single string in the history list
	entry.searchString: The actual string that was previously searched
	entry.date: The date that the string was searched for
Referenced By	2.3.2, 3.3
Viewpoint	Pseudocode

2.2.56. UserSearch.filterBy()



Name	**UserSearch.filterBy()**
Purpose	Apply filters requested by the user candidates to search for companies
Description	Apply the filters requested by the candidate and validated by the system on whether the filter options are applicable to the user.
Requirements	2.1.3.6, 3.3.3, 3.3.4
Inputs	Filter Options
Outputs	Filtered Query
Elements	**Filter parameters:** Valid list of options for user to filter by
Referenced By	2.2.57, 2.2.58, 2.3.40, 2.3.41, 2.3.42
Viewpoint	Flowchart

2.2.57. UserSearch.search()



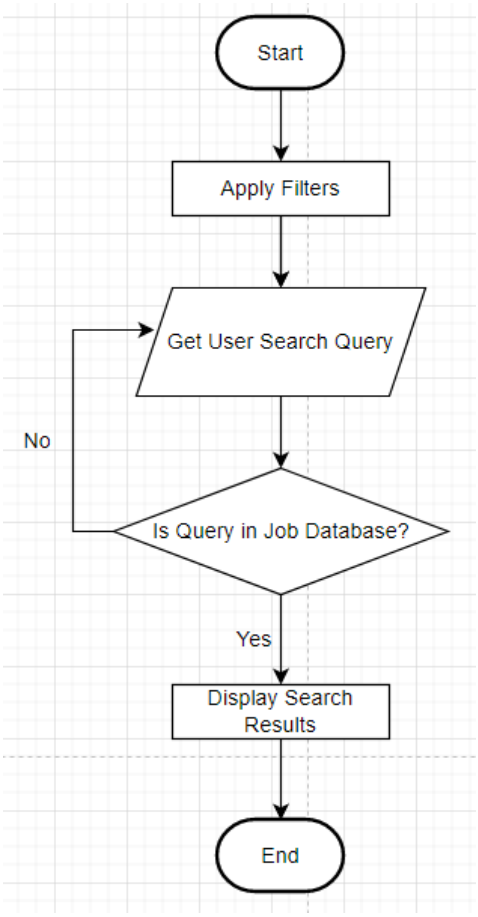
Name	**UserSearch.search()**
Purpose	Executes the users requested search
Description	The system will apply and run the requested search query to the database
Requirements	2.1.3.6, 3.3.3, 3.3.4
Inputs	Query Request
Outputs	Requested Query Result
Elements	**Query Request:** a requested query to a database
Referenced By	2.2.56, 2.2.58, 2.2.59, 2.2.60, 2.2.15, 2.3.41
Viewpoint	Flowchart

2.2.58. UserSearch.displayResults()

```
UserSearchDisplayResults()
    SET Results <-- GET UserSearchResults
    FOR result IN Results
        PUT result
```

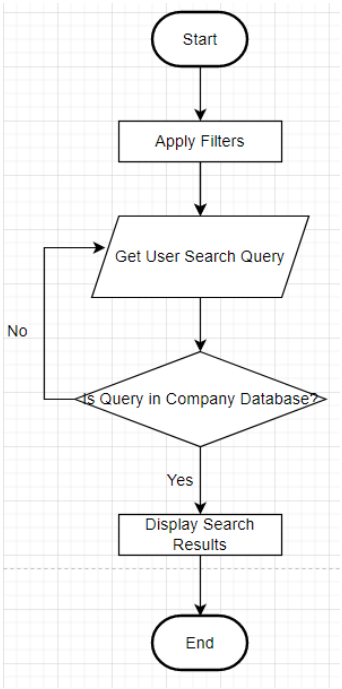
Name	**UserSearch.displayResults()**
Purpose	Display Information to User
Description	Display all Jobs queried for the user
Requirements	3.3.1 – 3.3.4
Inputs	Search Parameters
Outputs	Display Results
Elements	**Results:** the items returned to the user after a search
Referenced By	2.2.61
Viewpoint	Pseudocode

2.2.59. UserSearch.searchJobs()



Name	**UserSearch.searchJobs()**
Purpose	Search the job database for the user specified job request
Description	This assumes the user has already selected to search Companies. It takes the string/text of information the user has used in their query, uses that information and filters, to search the database for any jobs that fit/match the query.
Requirements	3.3.1
Inputs	Query
Outputs	List/Dictionary of Jobs
Elements	**Query:** A string from the user used to specify what kind of Jobs the user wants to find
	Database: where the collective jobs and respective information is stored
	Filters: other search criteria, like job length, job type, etc. for the user to specify that further narrow the results that the database returns
	List/Dictionary: a collection of Jobs that match the user specified criteria
Referenced By	2.2.14
Viewpoint	Flowchart

2.2.60. UserSearch.searchCompanies()



Name	**UserSearch.searchCompanies()**
Purpose	Search the Company database for the user specified Companies
Description	This assumes the user has already selected to search Companies. It takes the string/text of information the user has used in their query and uses that information, as other specified filters, to search the database for any jobs that fit/match the query.
Requirements	3.3.1
Inputs	Query
Outputs	List/dictionary of Companies
Elements	**Query:** A string from the user used to specify what kind of Jobs the user wants to find
	Database: where the collective Companies and respective information is stored
	Filters: Other search criteria, like Company type, etc., for the user to specify that further narrow the results that the database returns
	List/Dictionary: A collection of Companies that match the user specified criteria
Referenced By	2.2.14
Viewpoint	Flowchart

2.2.61. AuthenticationList.isAuthorized()

```
isAuthorized(authorized)
    IF authorized
        RETURN true
    ELSE
        RETURN false
```

Name	**AuthenticationList.isAuthorized()**
Purpose	Check if student user or company representative is authorized to perform an action on the account
Description	This is to make sure people are not doing actions they are not authorized to do
Requirements	2.1.3.3, 2.1.3.10, 3.2.5
Inputs	User input to do an action
Outputs	Confirms authorization
Elements	**Authorized:** User has appropriate credentials to access certain data.
	True/False: Represents the status of authorization.
Referenced By	2.2.14, 2.2.9
Viewpoint	Pseudocode

2.2.62. EditPage.addData()

```
function addData()
   id ← currentPageData.id
   connection ← users_database.connect()
   INPUT user → data[id].fields
   connection.send(
      INSERT users ← data WHERE id IS currentPageData.id
   )
```

Name	**EditPage.addData()**
Purpose	Function that adds data to a user entry, the entry is used by the EditPage page
Description	This function takes information from input fields on the EditPage page and sends the updated information to the backend. It initiates the backend to update the backend entry that matches the id corresponding to the current page's data.
Requirements	3.1.2
Inputs	No parameters, input comes from reading the input/form from the current page.
Outputs	No return values.
Elements	**currentPageData:** the data corresponding to the user who is logged in
	id: currentPageData.id
	users : the appropriate user's database (referring to either the student's or company's databases)
Referenced By	2.2.14
Viewpoint	Pseudocode

2.2.63. EditPage.editData()

```
function editData()
   id ← currentPageData.id
   connection ← users_database.connect()
   INPUT user → data[id].fields
   connection.send(
      UPDATE users ← data WHERE id IS currentPageData.id
   )
```

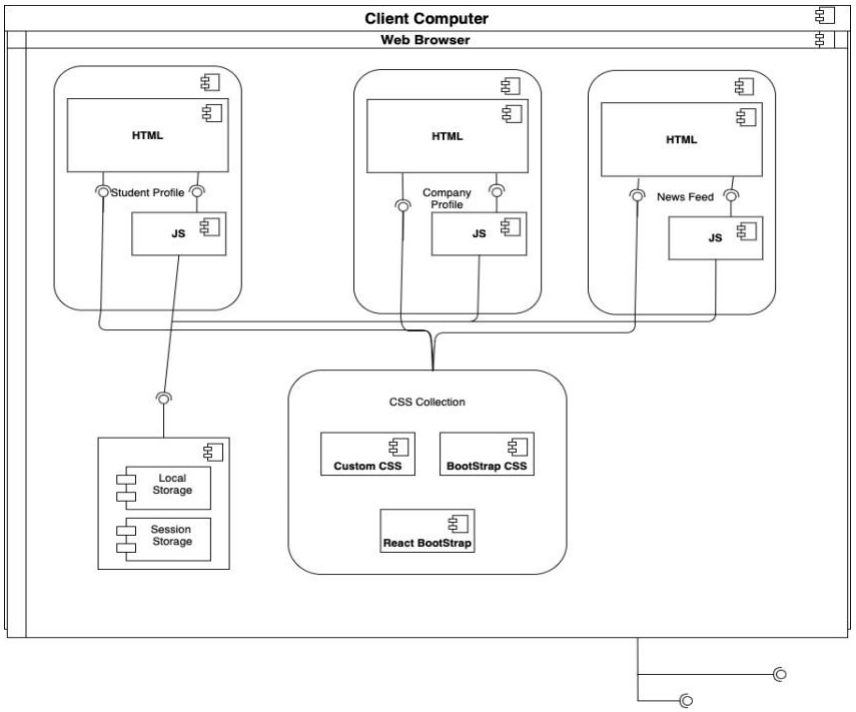
Name	**EditPage.editData()**
Purpose	Function that updates data in user entry, to be used by the EditPage page
Description	This function takes information from input fields on the EditPage page and adds the information to the backend. We will instruct the backend to insert the new backend entry.
Requirements	3.1.1
Inputs	No parameters, input comes from reading the input/form from the current page.
Outputs	No return values.
Elements	**currentPageData** : the data corresponding to the user who is logged in
	id : currentPageData.id
	users : the appropriate users database (referring to either the students or company's databases)
Referenced By	2.2.15
Viewpoint	Pseudocode

2.2.64. EditPage.removeData()

```
function removeData()
   connection ← users_database.connect()
   connection.send(
      DELETE FROM users WHERE id IS currentPageData.id
   )
```

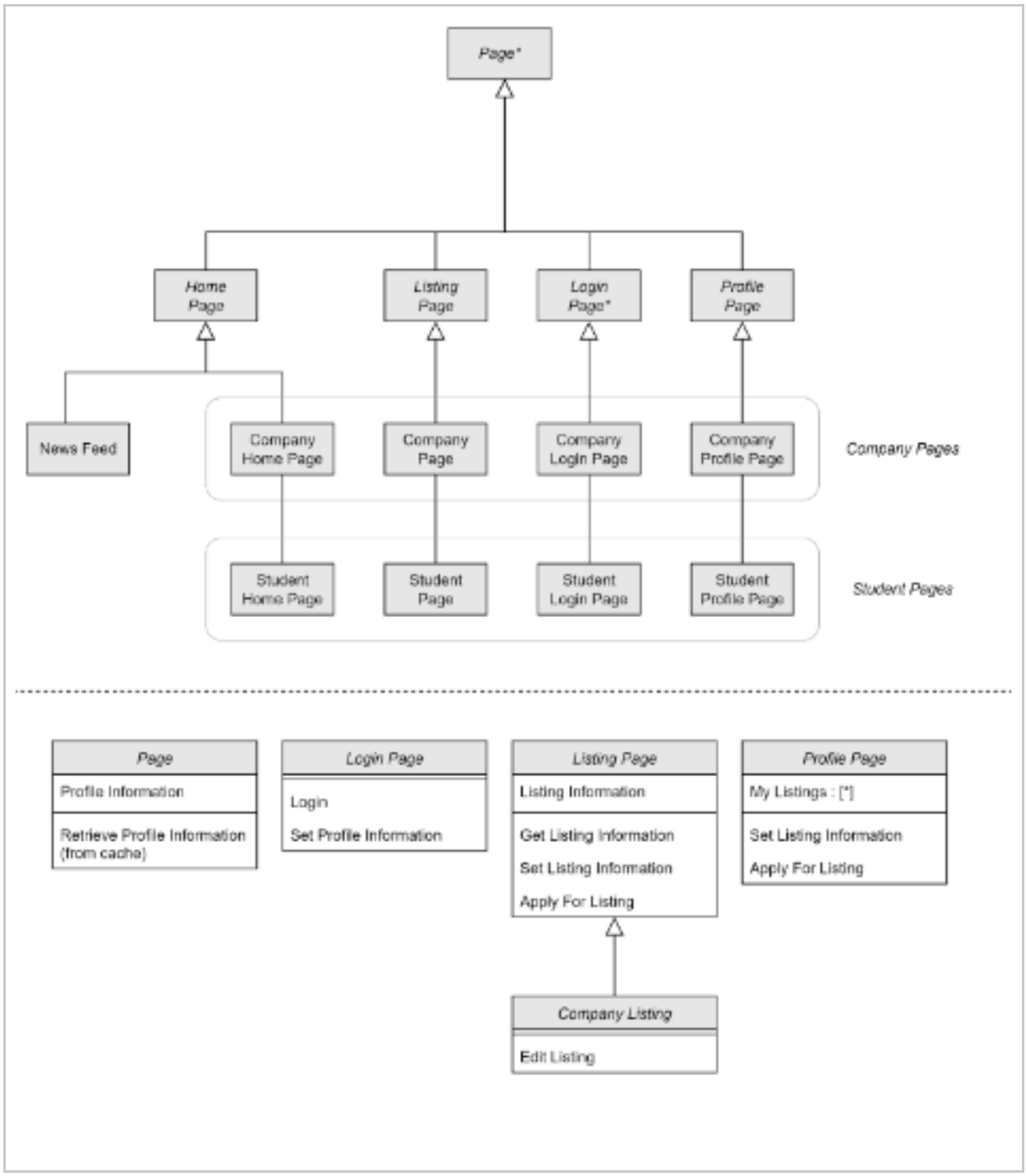
Name	**EditPage.deleteData()**
Purpose	Function that deletes data from user entry, to be used by the EditPage page
Description	This function uses the id corresponding to the user accessing the EditPage page and sent the backend to delete the entry corresponding to the user.
Requirements	2.1.3.3
Inputs	No parameters.
Outputs	No return values.
Elements	**currentPageData** – the data corresponding to the user who is logged in
	users – the appropriate users database (referring to either the student or company databases)
Referenced By	2.2.14
Viewpoint	Pseudocode

2.3. Presentation Tier - Frontend



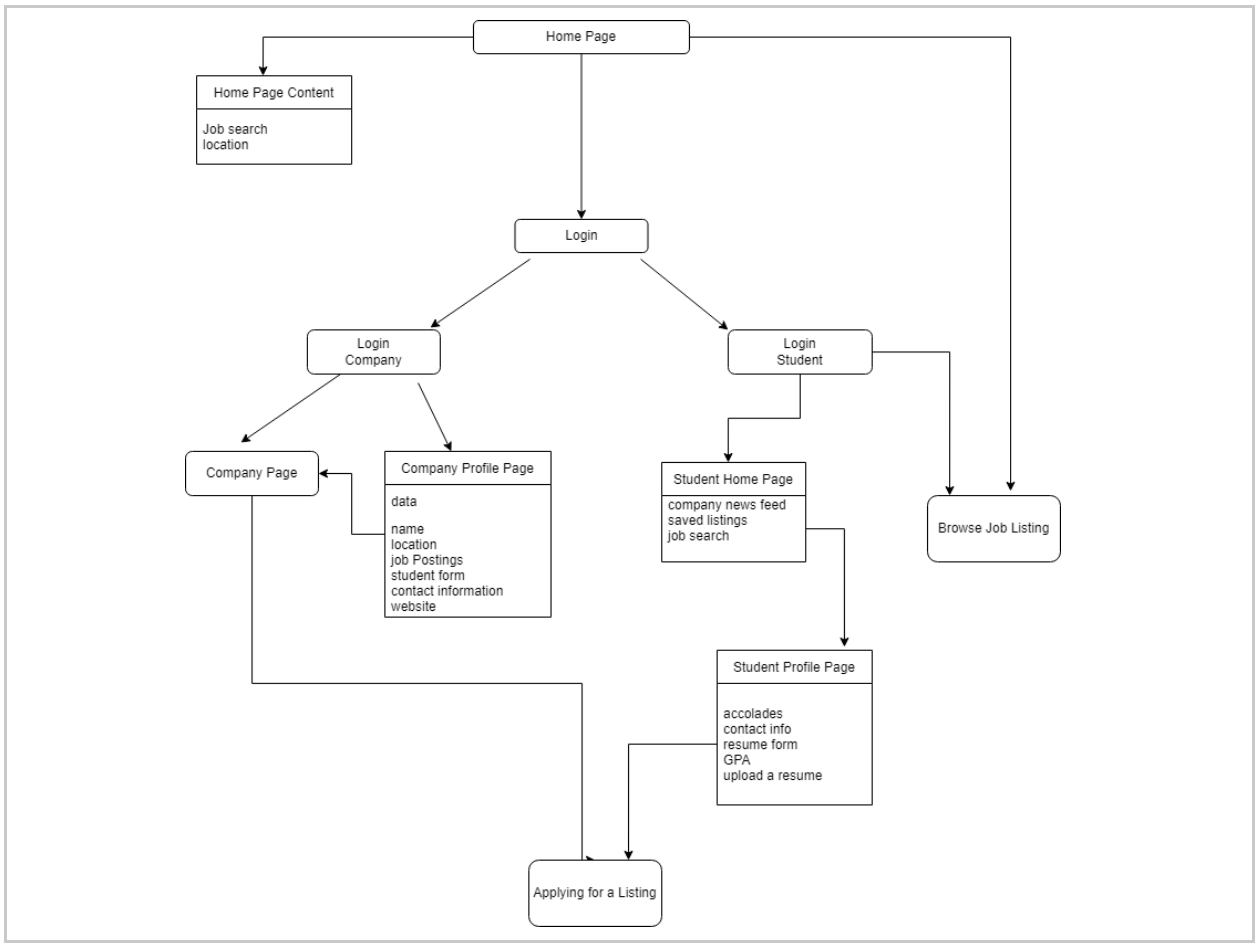
Name	**Component Diagram for the Presentation Tier — Frontend**
Purpose	This provides a high-level view of the frontend portion of the project.
Description	Representation of the News Feed and Student and Company Profiles at a high-level view.
Requirements	2.1, 2.4 - 2.5, 3.1, 3.4 – 3.5
Elements	**Student Profile:** Page containing the students' profile and links.
	Company Profile: Page containing the company's profile and links.
	News Feed: The landing page for the application.
	Logical and Session Storage: Storage.
	Custom, Bootstrap and React CSS: The global and application-level frameworks and styling.
Viewpoint	Component Diagram

2.3.1. UML Class Diagram View of the Frontend Object Types



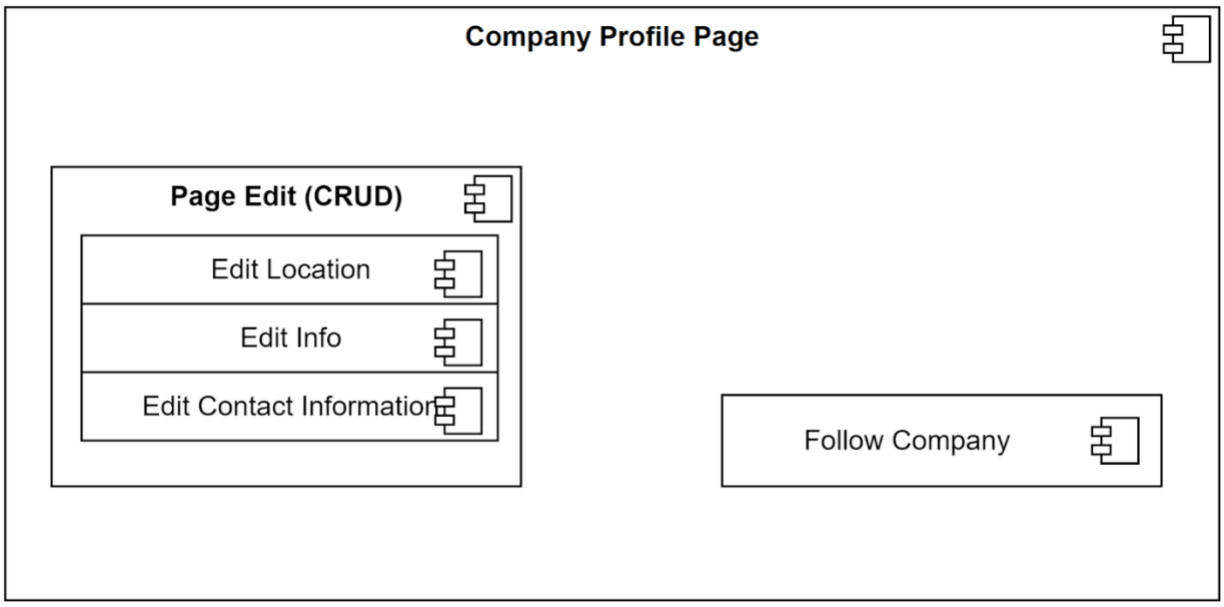
Name	**UML Class Diagram of Frontend Object Types**
Purpose	Illustrates class definitions and relationship between different object types
Description	Representation of the News Feed, as well as the Company and Student pages in terms of inherited classes, demonstrates how pages and local data are related
Requirements	2.1. - 2.1.2. , 3.1. - 3.1.2 , 3.1.3.1, 3.4. - 3.4.2.
Elements	**Student Home Page:** Page containing the students' company feed.
	Company Home Page: The page where a company can configure its profile.
	Student Profile: Profile page of the student.
	Company and Student Login Pages: Login pages for the company and student; the student login page is accessible by one individual, while a company may have multiple accounts.
	Student Profile Page: Public page with specific information about the student.
	Company Profile Page: Public page with specific information about the company.
Referenced By	2.2.4, 2.2.9, 2.2.36, 2.3
Viewpoint	Class Diagram

2.3.2. Structure View of Frontend Application.



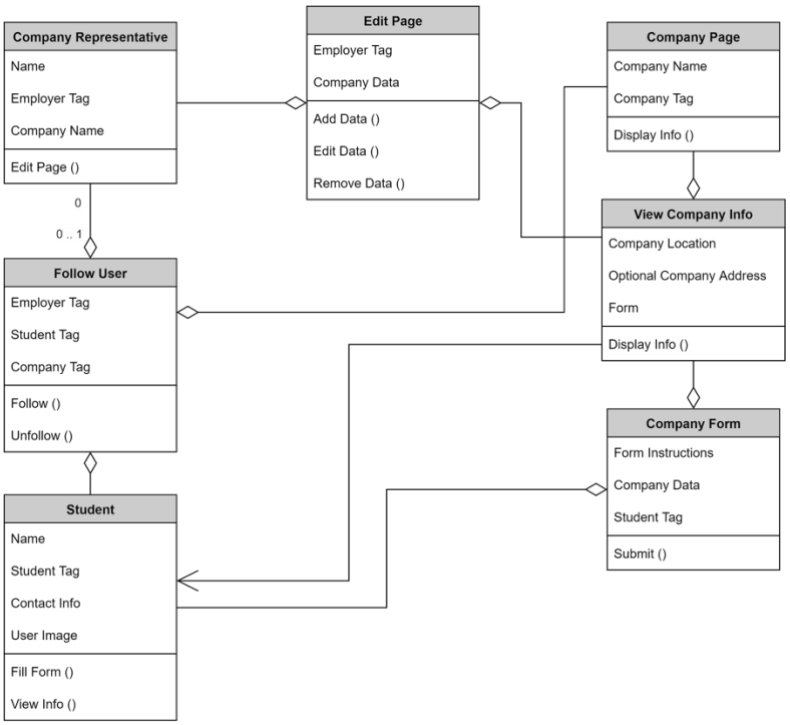
Name	**Structure View of Frontend Application**
Purpose	Describes how the front-end functions and front pages call each other.
Description	Breaks down class diagrams and describes how each class functions.
Requirements	2.1.3 - 2.1.3.10. , 3.1.3. - 3.1.3.4. , 3.4. - 3.4.2.
Elements	**Home Page:** The website homepage for an account that is not logged in.
	Login Company: The login page for a company profile.
	Login Student: The login page for a student.
	Company Page: A page of a company's profile.
	Company Profile Page: The public page of a single company.
	Applying for a Listing: A page that facilitates the application process and may be accessed from a company page or the job listing page.
	Browse Job Listing: A page to look at job listings.
	Student Home Page: A custom version of the home page for a student that is logged in.
	Student Profile Page: The public profile page of a student.
Referenced By	2.3
Viewpoint	Structure Chart

2.3.3. Companies Profile Page: UI Components



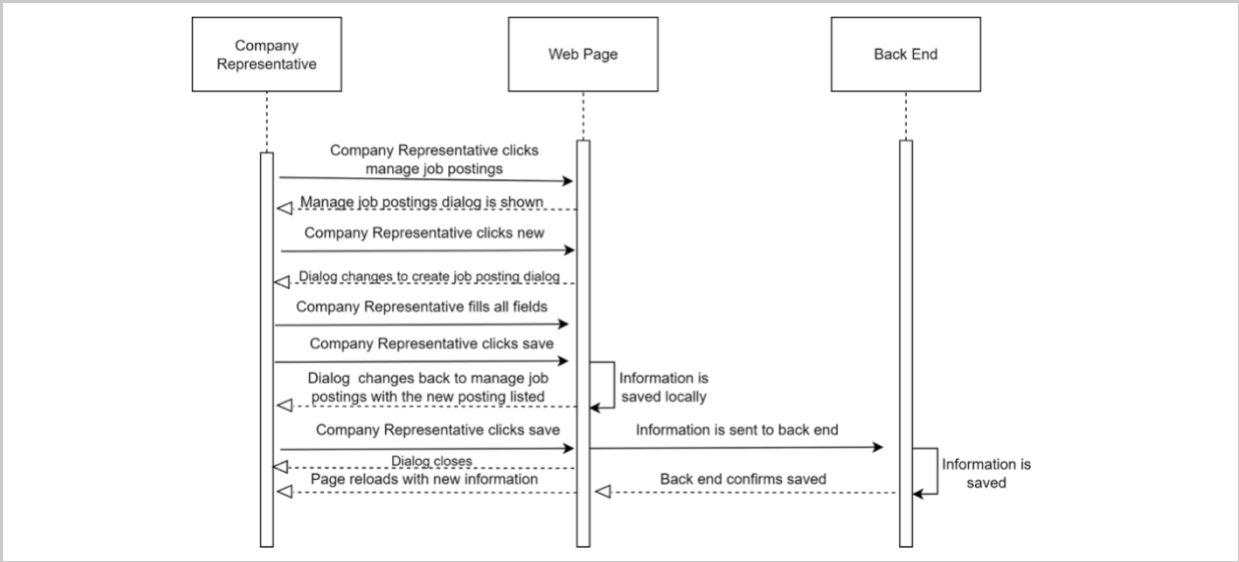
Name	**Companies Profile Page: UI Components**
Purpose	List the UI Components of the Company Profile Page.
Description	These UI elements are only accessible to those allowed to edit the company page.
Requirements	2.1.3.1 - 2.1.3.10
Elements	**Page Edit:** This will allow users to edit company profile information.
	Location: The address of your company's headquarters. Individual fields for street address, city, country, province, etc.
	Contact Information: Socials, public email, phone number.
	Info: A section for information about the company. This section will be a large open text field with formatting options.
	Follow Company: This button allows students to follow a company
Referenced By	2.1
Viewpoint	Component Diagram

2.3.4. Companies Profile Page: UML Class Diagram



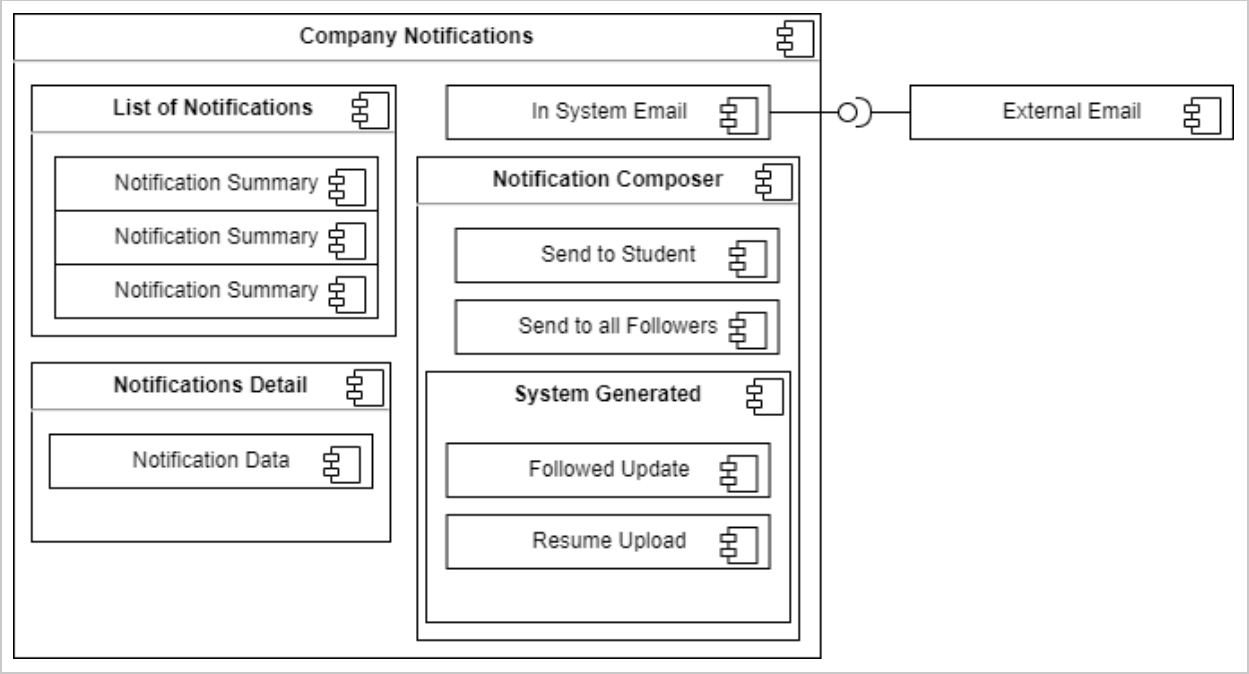
Name	**Company Profile Page (Frontend): UML Class Diagram**
Purpose	List the UI Components of the Company Profile Page.
Description	These components will represent the objects containing all the information on the Company profile page.
Requirements	2.1
Elements	**Company Representative:** Information about the user (company representative).
	Company Page: information about the company.
	Edit Page: interface which allows the user to edit the company page.
	Follow User: interface which enables the user to follow a specific student or other company representatives.
	Student: information about a student.
	View Company Info: interface which allows the user to view the company information.
	Company Form: interface which enables the user to edit the information for a company.
Referenced By	2.3
Viewpoint	UML Class Diagram

2.3.5. Companies Profile Page: Sequence Diagram



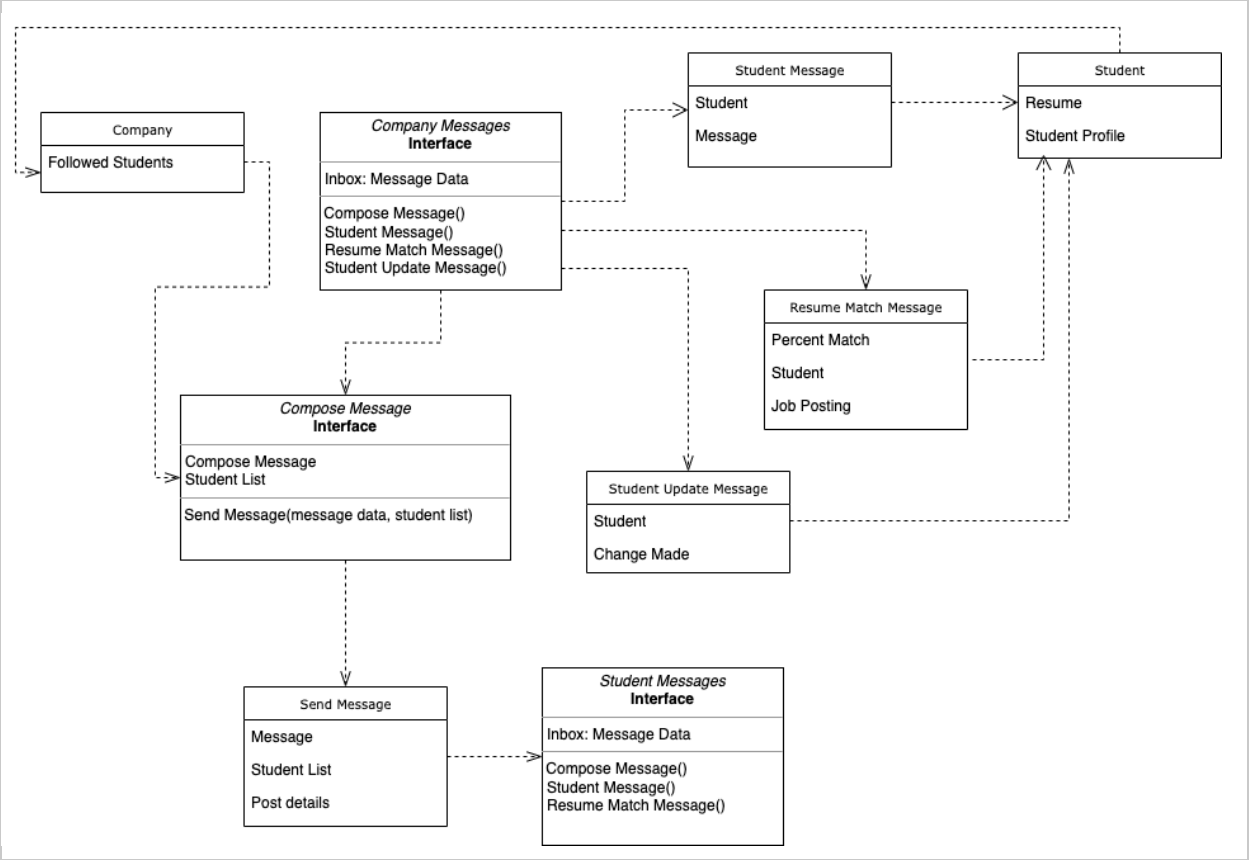
Name	**Companies Profile Page Sequence Diagram**
Purpose	Describe an example sequence on a company profile page.
Description	The process a company representative uses to change the company information.
Requirements	2.1.3.1 - 2.1.3.10
Elements	**Company Representative clicks edit page:** The company representative clicks the Page edit button.
	Edit dialog is shown: A dialog opens showing a checkbox list of editable fields.
	Company Representative updates info: The company representative makes some edits in the info field. They could fix a typo in the company motto, add a link to their website, etc.
	Representative Clicks Save: The button is clicked to save.
	Company representative fills all fields: This dialogue has multiple editable fields. See Companies profile page UI components.
	Company Representative clicks save: The company representative clicks the save button on the dialog.
	Information is sent to the back end: Information is sent to the back end to be saved to the database.
	Edit dialog: closes and shows the unedited page for a moment.
	Information is saved: Information is saved in the database.
	Back end confirms save: The back end tells the web page that the information has been saved.
	Page reloads with new information: The page reloads, populating its fields with current information from the database as usual (except the information has just been changed so that the web page will show the newly edited information)
Referenced By	2.1
Viewpoint	Sequence Diagram.

2.3.6. Companies Notifications: UI Components



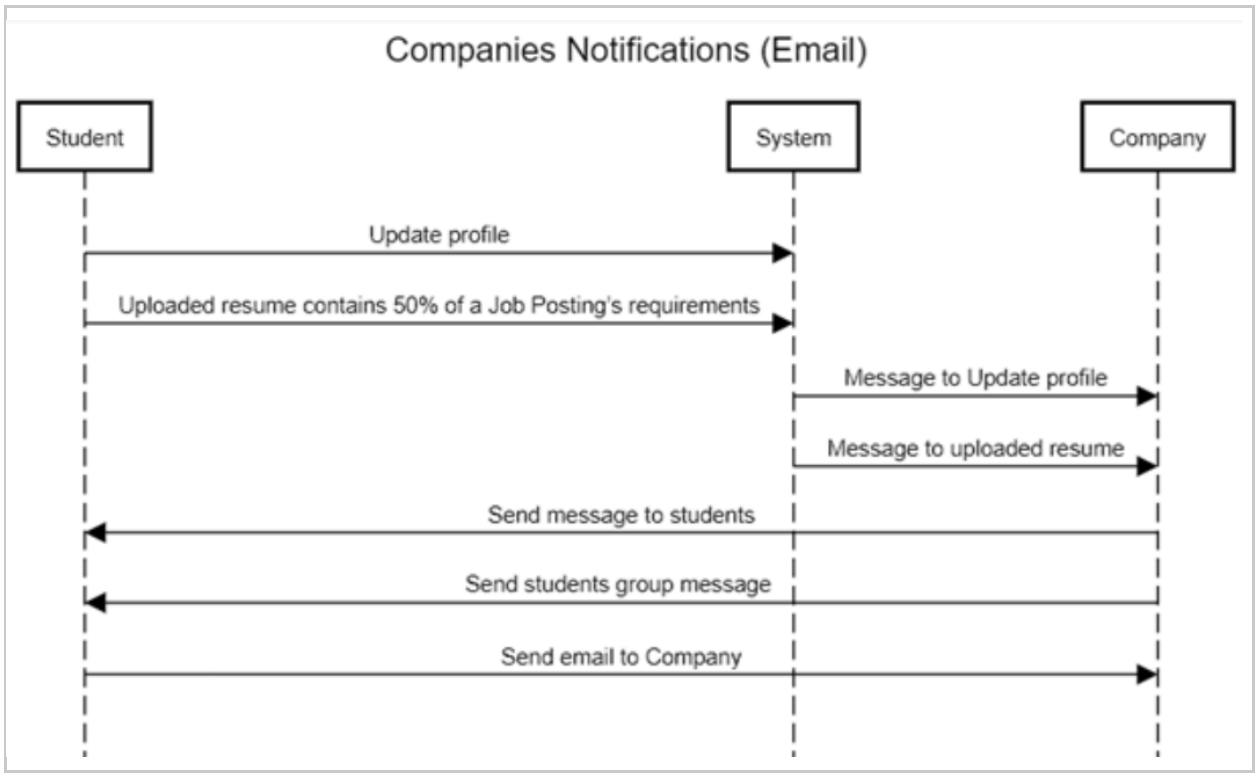
Name	**Company Notification UI Component Diagram**
Purpose	Layout of notification components
Description	Link the two systems of email together.
Requirements	2.2.1 - 2.2.5
Elements	**List of Notifications**: A shortened version of the message that can be visible before selecting the message.
	Notification detail: All details will be listed once notification is selected.
	System Generated: Non-message notifications
	Notification Composer: UI for determining the recipients of messages
	In System Email: Embedded inbox to contain all the notifications received in chronological order
	External Mail: Inbox of the user at the email address linked on their profile. External to the application
Referenced By	2.2, 3.2
Viewpoint	Component Diagram

2.3.7. Companies Notifications: UML Class Diagram



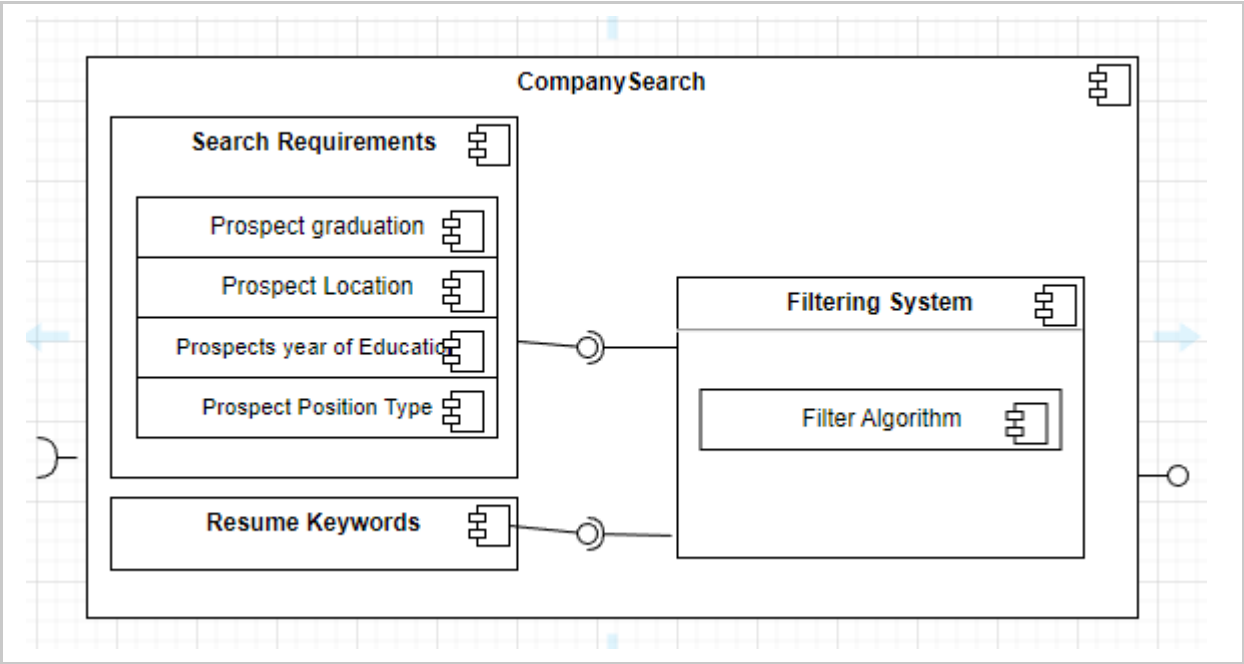
Name	**Companies Notifications: UML Class Diagram (front end)**
Purpose	Describes the class connections within the company's notifications.
Description	UML class diagram for Company notifications.
Requirements	2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5
Elements	**Company Messages:** Consists of messages received and being able to send a new message/notification.
	Student Messages: Messages from students include basic info about each student.
	Resume Match Message: Message with student resume, match percentage, and what job posting they matched with.
	Student update message: Messages with information about followed student's profile updates.
Referenced By	2.2
Viewpoint	UML Class Diagram

2.3.8. Companies Notifications: Sequence Diagram



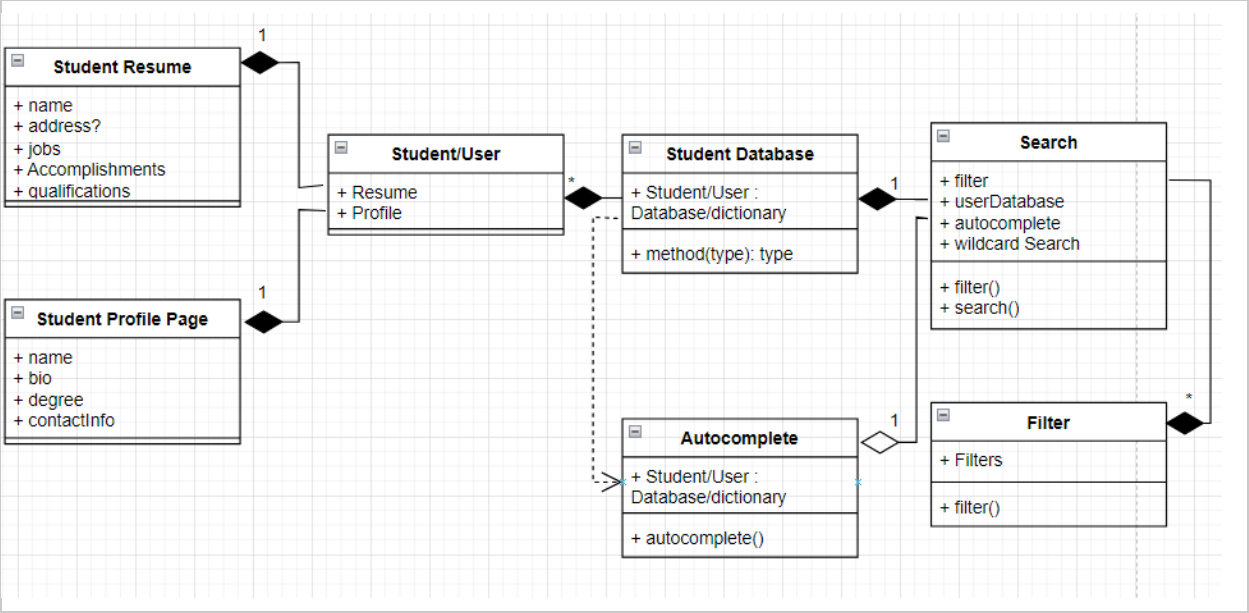
Name	**Companies Notifications: Sequence Diagram**
Purpose	To show the sequence of messages between students, companies, and system.
Description	Simple Companies Notifications: Sequence Diagram showing the triggers of the email exchange.
Requirements	2.2.2, 2.2.3, 2.2.4, 2.2.5
Elements	**Student:** A representation of the student performing actions against the system.
	System: A representation of the logic that communicates between the user and the database.
	Company: A representation of the company performing actions against the system.
	Events: Actions performed by any of the three entities throughout the session.
Referenced By	2.3.8
Viewpoint	Sequence Diagram

2.3.9. Companies Search: UI Components



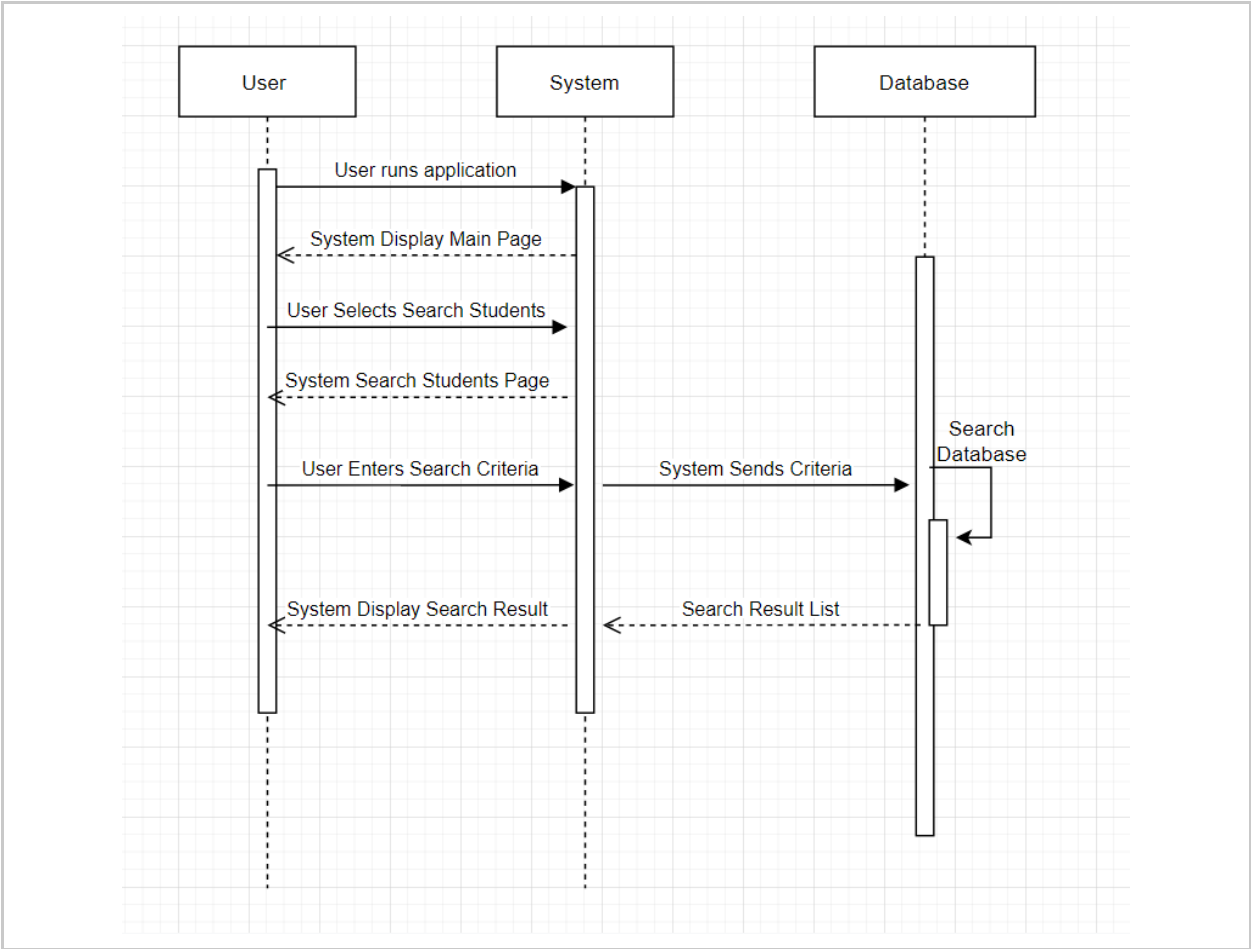
Name	**UI Components Company Search**
Purpose	To show recommended components for functioning company side search.
Description	Simple component diagram demonstrating system for company search.
Requirements	2.3.1 - 2.3.8
Elements	**Filter Year Of Education:** Years currently completed in school.
	Filter Location: Candidate's current locations.
	Filter Graduation Date: Year of graduation.
	Filter Resume Keywords: a company may search for keywords and phrases in a resume.
	Filter Position Type: A candidate may look for a job position by using the filter option.
Referenced By	2.3
Viewpoint	Component Diagram

2.3.10. Companies Search: UML Class Diagram



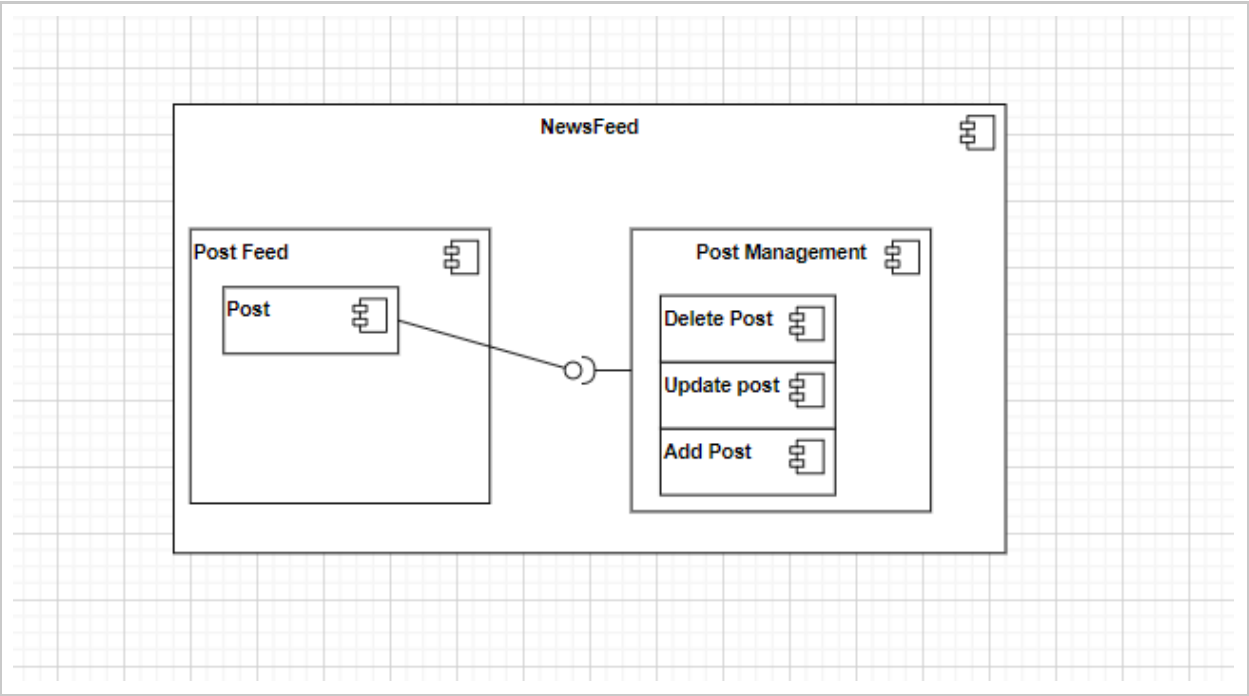
Name	**Companies Search UML Class Diagram**
Purpose	Lay out the basic classes and Inheritance between said classes for the Company side of the Search Page.
Description	The Diagram describes a structured class diagram of the necessary classes and a supposed structure to make the Company side Search page work.
Requirements	2.3.1 - 2.3.8
Elements	**Search**: The search function that runs everything.
	Filter: Allows the user to filter the results according to the different specifications taken from the user database and other information.
	Student Database: The database where the users are stored.
	Autocomplete: completes the user's search text taking data from the database.
	Student: the overall student account contains both the profile information and a resume.
	Profile: Basic Student Information, a summary of Resume information.
	Resume: Detail information on Student's work, education, etc.
Referenced by	2.2.53, 2.3
Viewpoint	UML Class Diagram

2.3.11. Companies Search: Sequence Diagram



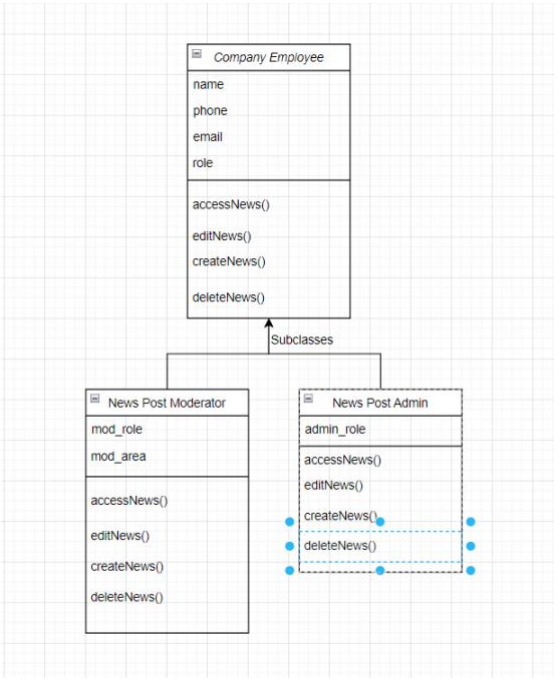
Name	**Company Search Sequence Diagram**
Purpose	To illustrate the sequencing of events from the company's perspective during a search.
Description	A diagram showing the sequence of events for a search from a company.
Requirements	2.3
Elements	**User:** A representation of the company performing actions against the system.
	System: A representation of the logic that communicates between the user and the database.
	Database: A representation of the collection of data in need of searching.
	Events: Actions performed by any of the three entities throughout the session.
	Results: Returned search results from the database to the system, then displayed to the user.
Referenced By	2.3.2
Viewpoint	Sequence Diagram

2.3.12. Companies News Feed: UI Components



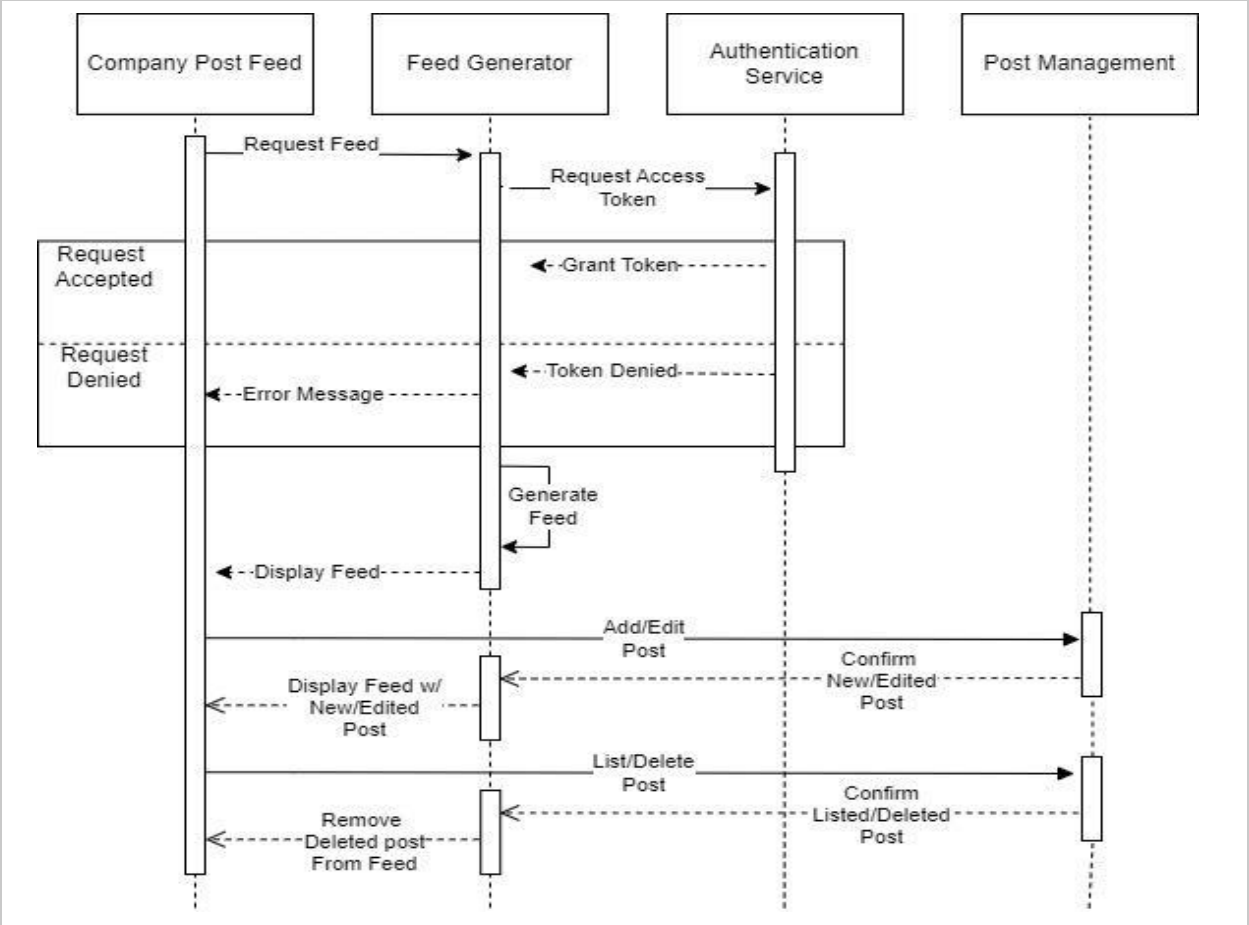
Name	**Companies News Feed UI Components**
Purpose	Show connected components of News Feed and their interactions with other components.
Description	Post feed is populated with posts. These posts interact with Post Management, giving the ability to add, delete, remove, and update posts that a user owns.
Requirements	2.4.1, 2.4.3
Elements	**Post Feed:** List all posts for viewing.
	Post: An event, activity, or message of a company.
	Post Management: System to control updates on posts, integrates with ACL.
	Delete Post: Removes post from feeds.
	Update Post: Changes content of Post in Feeds.
	Add Post: Adds post into Feeds.
Referenced By	2.3
Viewpoint	Component Diagram

2.3.13. Companies News Feed: UML Class Diagram



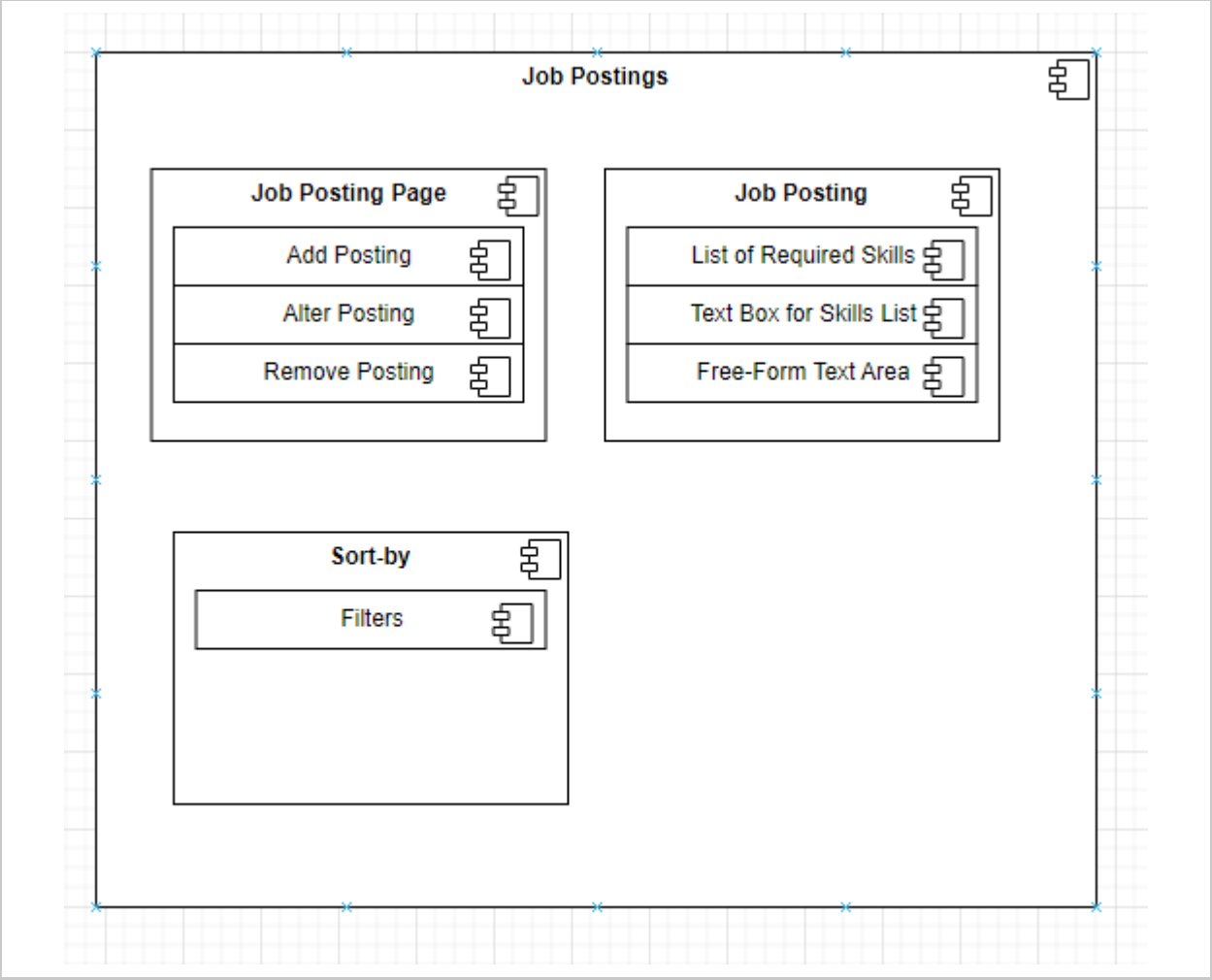
Name	**Companies News Feed: UML Class Diagram**
Purpose	This Diagram describes classes and subclasses for the user's authorization to interact with components of the company's news feed.
Description	This describes what posts of the company news feed individual users can access, edit, create, and delete based on their authorization. For example, a standard employee can only access and interact with their department's area of the news feed, but only to a certain degree. Moderators are allowed to access and interact with everything in their department's area of the news feed but not in other departments' areas of the news feed. Finally, admins can access and interact with all parts of the news feed regardless of department.
Requirements	2.4.1, 2.4.3
Elements	**Authorization:** The level users can interact with the news feed components
	News Feed Access: Ability to access the news feed posts
	Edit: Edit posts on the news feed
	Create: Creates posts on the news feed
	Delete: Deletes posts on the news feed
	Company Employee: A standard employee class
	News Post Moderator: A class for an employee that has full control over their department's news feed
	News Post Admin: A class for an employee that has full control over the entire company's news feed
Referenced By	2.3.33
Viewpoint	UML Class Diagram

2.3.14. Companies News Feed: Sequence Diagram



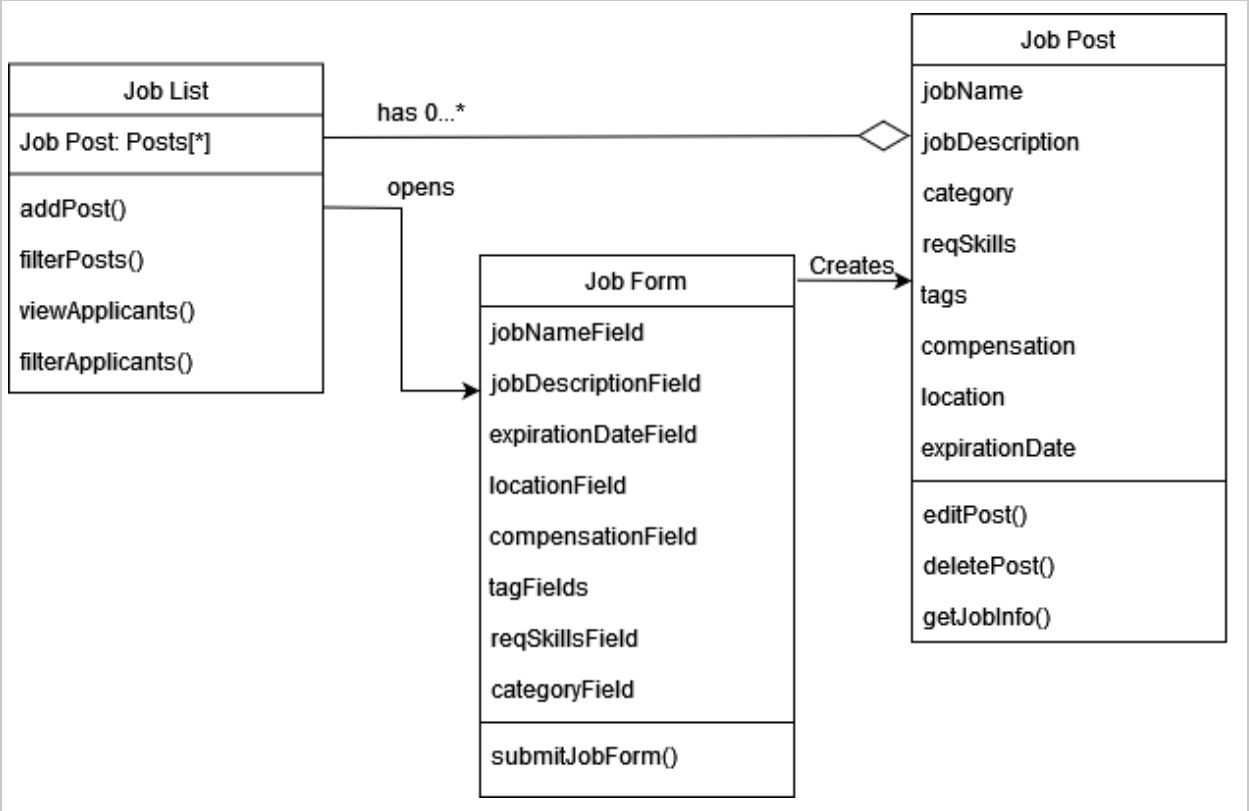
Name	**Company News Feed Sequence Diagram**
Purpose	Show the order and extent of functions concerning the news feed displayed to users labeled as companies.
Description	The news feed is generated after confirming credentials and displayed to the user. Creating, editing, and deleting user posts are also permitted.
Requirements	2.4
Elements	**Company Post Feed:** Where posts and news are displayed to the user.
	Feed Generator: Generates feed to display based on followed users and universities.
	Authentication Service: Confirms user authentication information.
	Post Management: Allows for creating, editing, and deleting users' posts.
Referenced By	2.3.12
Viewpoint	Sequence Diagram

2.3.15. Companies Job Postings: UI Components



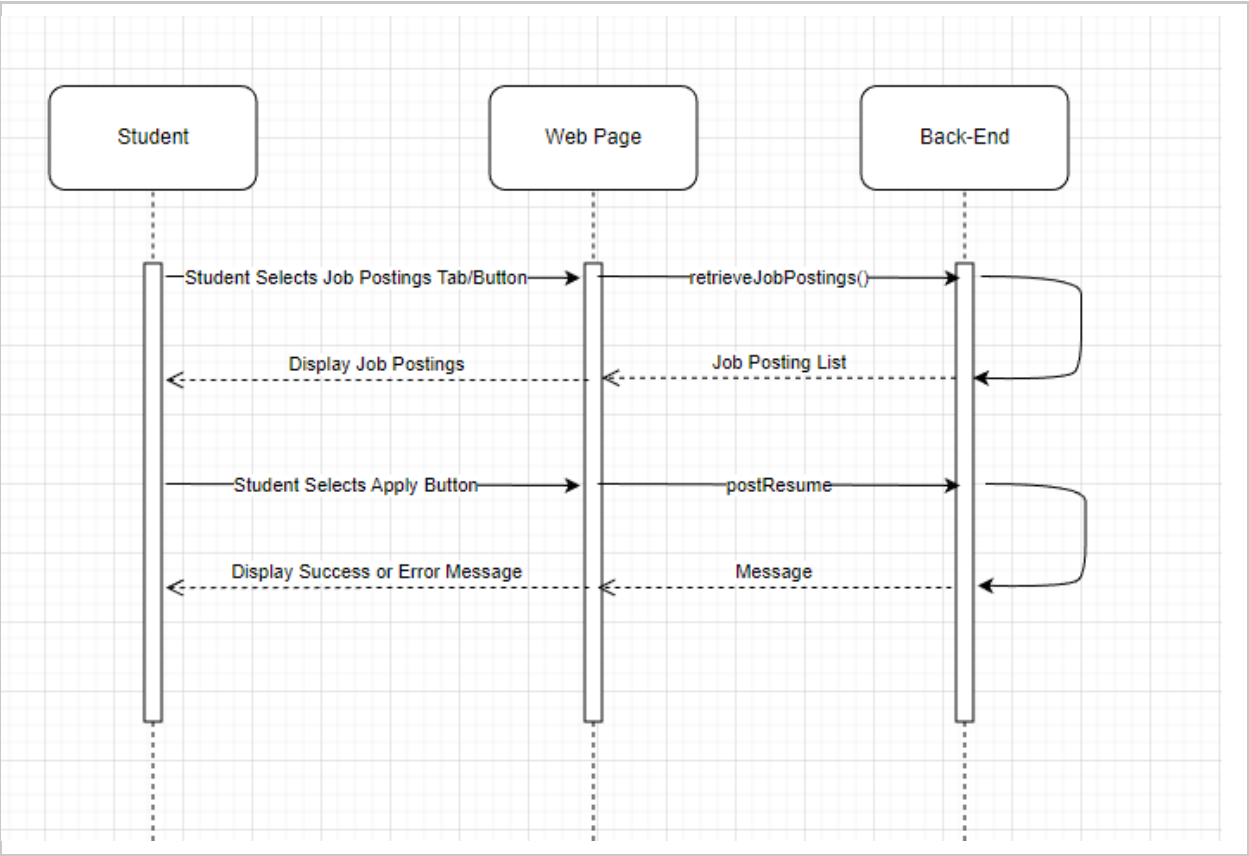
Name	**Job Postings UI Components**
Purpose	Describes the different UI components of the Job Postings Page
Description	Lists out the different UI Components of the Job Postings Page.
Requirements	2.5, 3.3
Elements	**Job Postings Page:** Allows for adding, altering, and removing a posting.
	Job Posting: List of required skills, a text box for the skills list, and a free-form text area.
	Sort-By: Includes a set of filters or by text to filter postings.
Referenced By	2.3
Viewpoint	UI Components

2.3.16. Companies Job Postings: UML Class Diagram



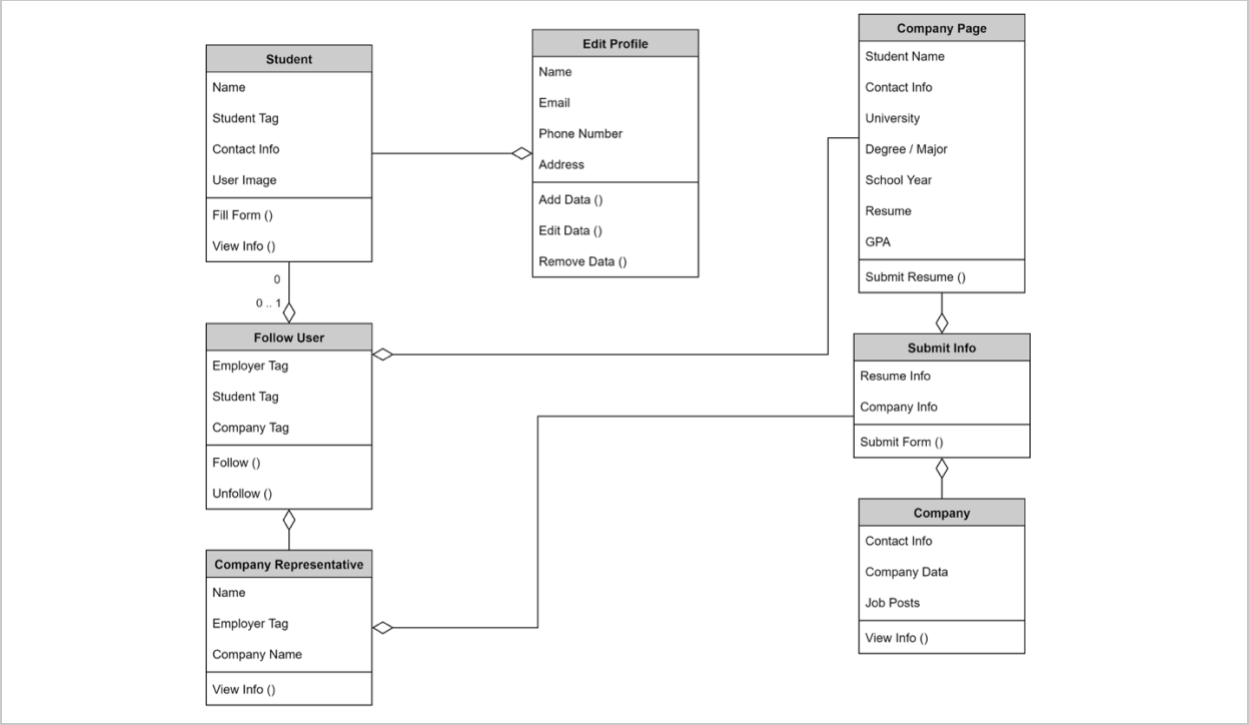
Name	**Company Job Postings Diagram**
Purpose	To show the logic behind the Company Job Postings frontend.
Description	A diagram demonstrating the relations between classes used for job posting logic.
Requirements	2.5.1, 2.5.2, 2.5.3, 2.5.4, 2.5.5, 3.3.4
Elements	**Job List:** A list of all job postings for a company.
	Job Form: A form for creating new job postings.
	Job Post: A single job posting with all necessary information for applicants.
Referenced By	2.2.28, 2.2.29, 2.2.30, 2.3.15, 2.3.73, 2.5
Viewpoint	Class Diagram

2.3.17. Companies Job Postings: Sequence Diagram



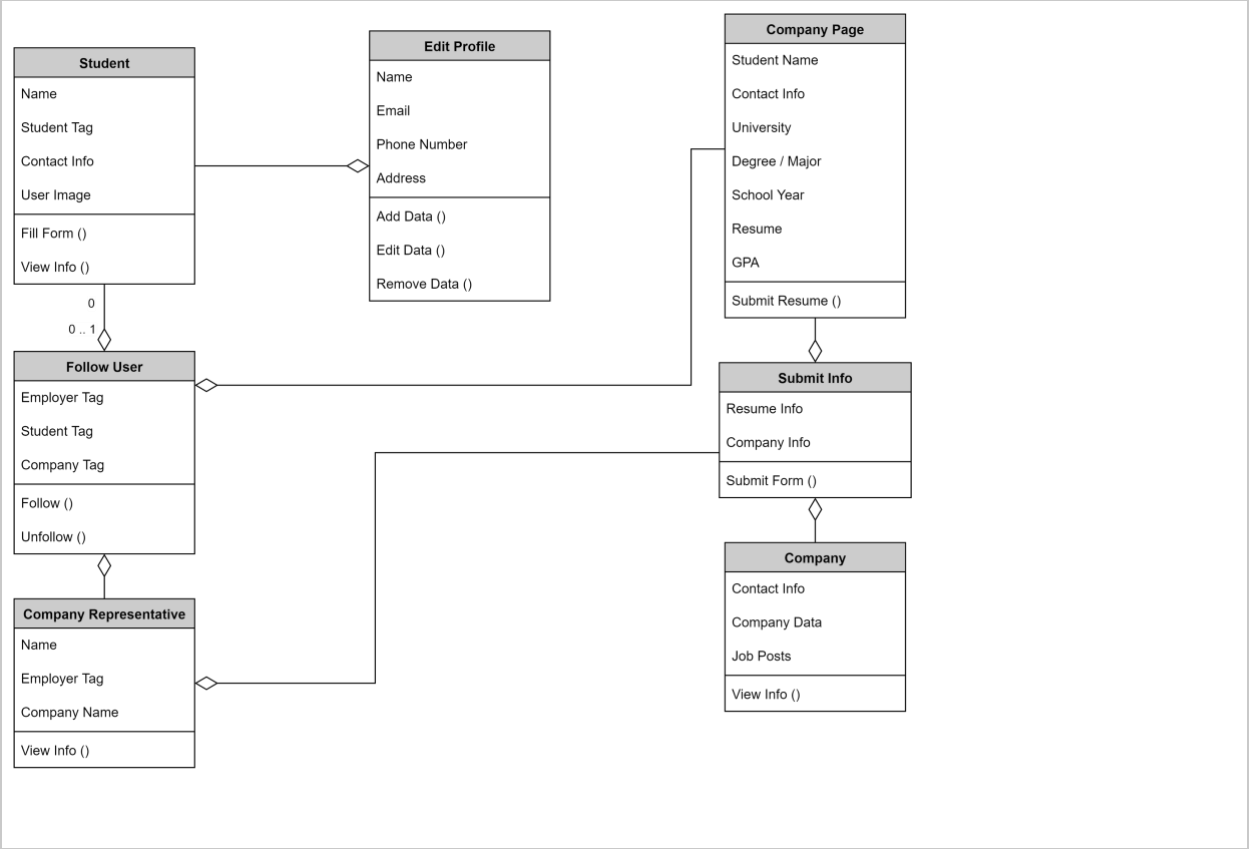
Name	**Student Read Job Postings Sequence Diagram**
Purpose	Displays sequence of how students will view job postings.
Description	Students will use the frontend UI to communicate a desire to view job postings, and the UI will communicate to the backend to receive data to populate the UI. Students will also apply to job postings and post resumes back to a company job posting.
Requirements	2.5.1, 2.5.2, 2.5.4
Elements	**Student:** A user in the role of student.
	Web Page/UI: The user will visually see and interact with the interface.
	Backend: The interface with which the software will interact but not the user.
Referenced By	2.3.73, 2.5
Viewpoint	Sequence Diagram

2.3.18. Students Profile Page: UI Components



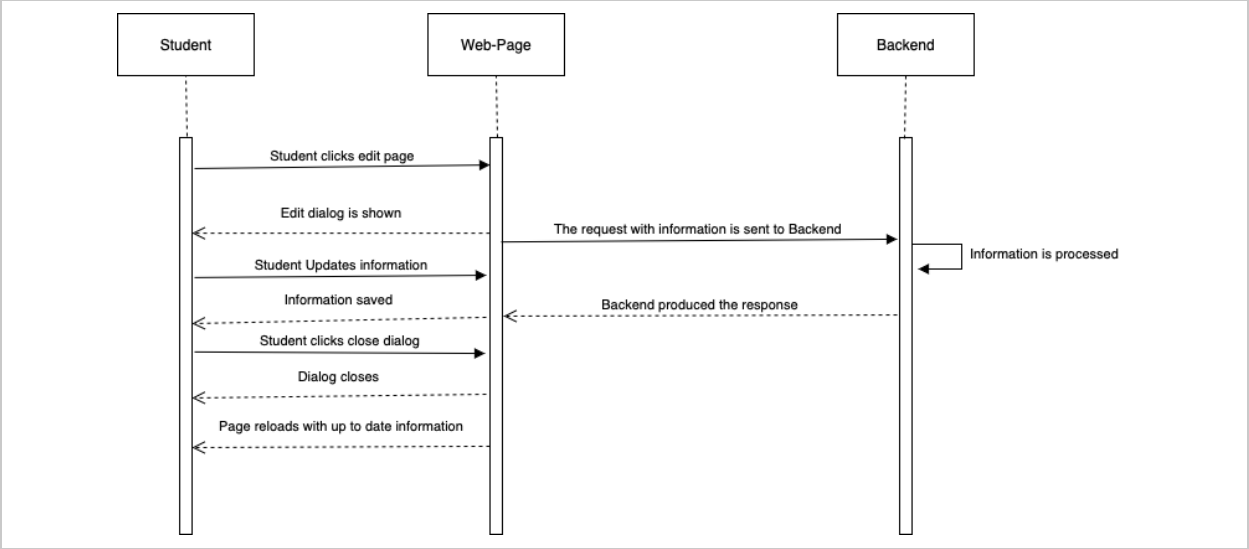
Name	Students Profile Page: UI Components
Purpose	Outline the Interactable elements of a student profile.
Description	These are interactable elements that only the student will be able to interact with. There will be no interactable elements for any visitors to the page other than the student.
Requirements	3.1
Elements	**Page Edit: Edit the information on a student profile page.**
	Education/GPA: A field to put information about Education/GPA.
	Edit Contact Info: A field to edit contact info, including email, phone number, socials such as LinkedIn, and mailing address.
	Awards: Any awards or achievement recognition the student wishes to have displayed.
	Edit Skills: A place for the student to list their skills.
	Upload Resume: A button that will allow the student to upload a resume.
	Choose File: A button that uses the OS's file selection interface.
	Submit: Submits the selected file to the student's page.
	Follow Student: A button that lets a company follow a student.
Referenced By	2.3
Viewpoint	Component Diagram

2.3.19. Students Profile Page: UML Class



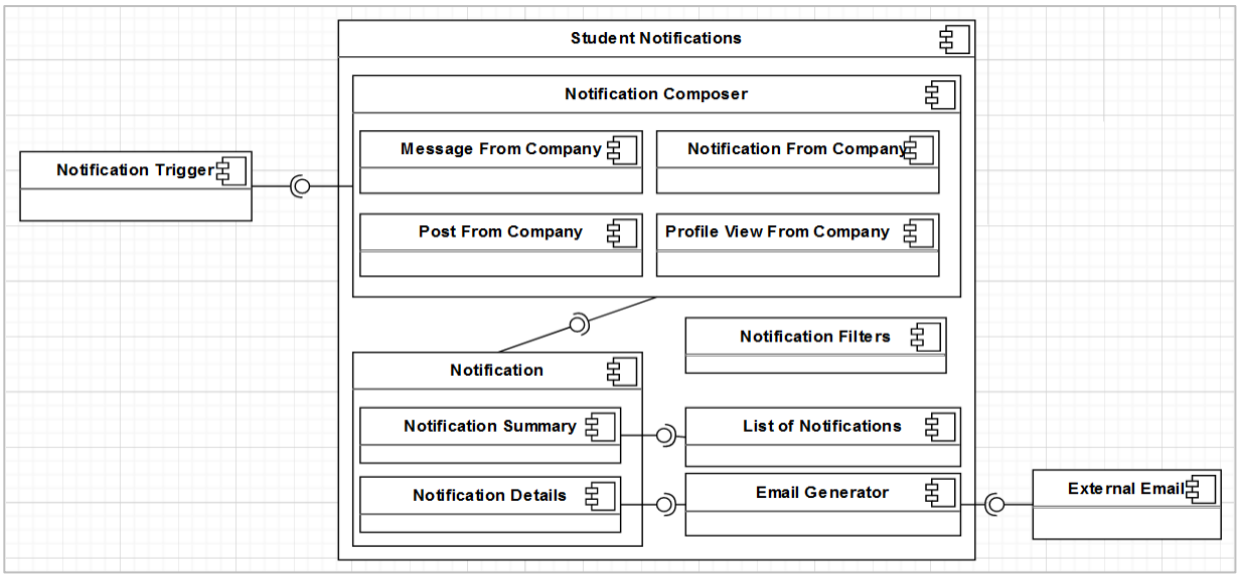
Name	**Students Profile Page (Frontend): UML Class Diagram**
Purpose	List the UI Components of the Student Profile Page.
Description	These components will represent the objects containing all the information on the student profile page.
Requirements	3.1.3.1 - 3.1.3.4
Elements	**Student:** Information about the user (student).
	Company Page: information about the student.
	Edit Profile: interface allows the user to edit the student profile information.
	Follow User: interface which enables the user to follow another student.
	Company: information about a company.
	Submit Info: interface allows the user to submit their info or resume to the company.
	Company Representative: interface which enables the user to view the information for a company representative.
Referenced By	3.1
Viewpoint	UML Class Diagram

2.3.20. Students Profile Page: Sequence Diagram



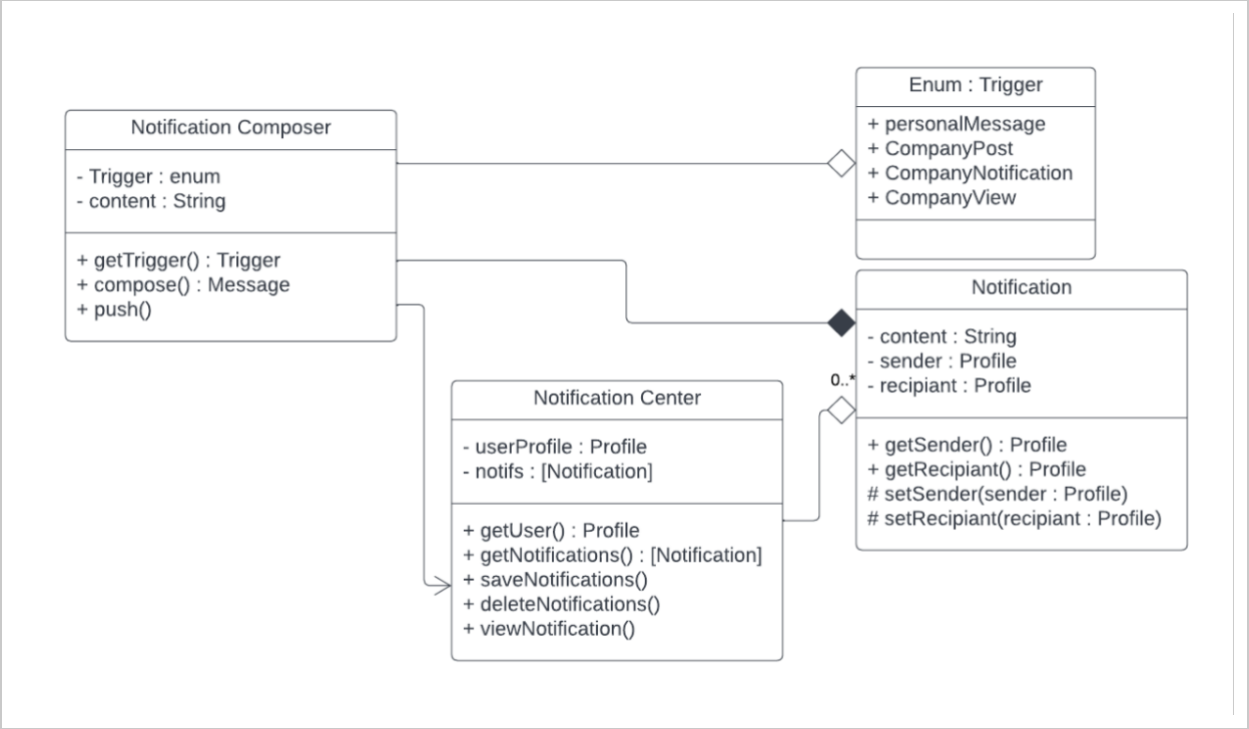
Name	**Students Profile Page: Sequence Diagram**
Purpose	Demonstrate a sequence the profile page might go through.
Description	The student updates their contact information.
Requirements	3.1.3.1 - 3.1.3.4
Elements	**Student:** The owner of the profile page.
	Web Page: The student's profile page.
	Back End: Anything the user doesn't see.
	Student Clicks Edit Page: The edit page button is clicked.
	Edit Dialogue is shown: A dialogue is shown containing editable fields corresponding to the sections of the student's profile page. These fields begin with the current contents of the page.
	Student edits contact information: The student changes the "Contact Information" field.
	Student clicks save: The student clicks a button labeled save at the bottom of the dialogue.
	Information is sent to the back end: The contents of each field are sent to the back end to be saved.
	Dialogue closes: The dialog window closes, showing the page with the unedited information.
	Information is saved: Saves the information to the database.
	Back end confirms save: The database wrapper function returns after the information is saved. (most likely with # of rows affected, but this information is irrelevant).
	Page reloads with up-to-date information: The page reloads, populating its fields with current information from the database as usual (except the information has just been changed so that the web page will show the newly edited information).
Referenced By	3.1
Viewpoint	Sequence Diagram

2.3.21. Students Notifications: UI Components



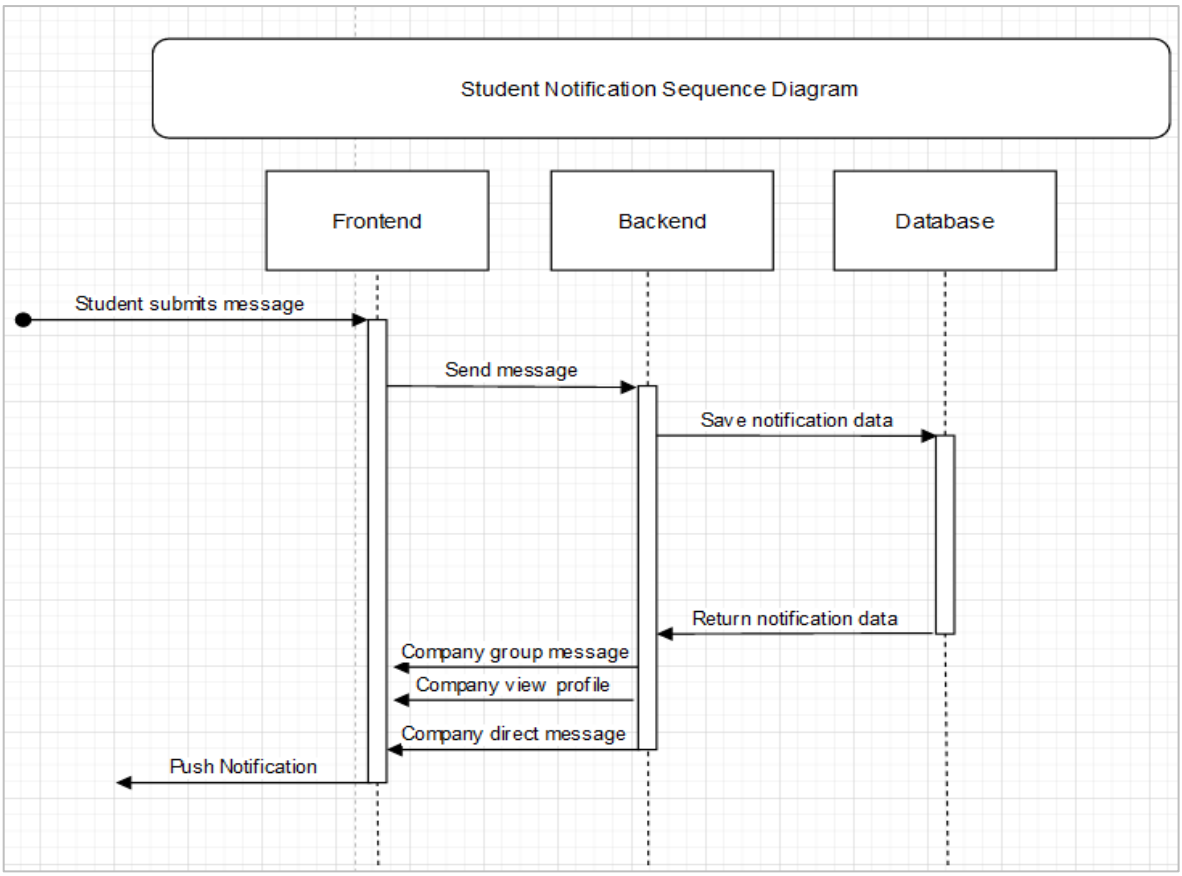
Name	**Student Notification UI Components**
Purpose	To show recommended components for student notifications
Description	Simple component diagram to describe the CRUD commands for a student's notification
Requirements	3.2.1 - 3.2.7
Elements	**Notification Trigger** - An event creating a notification for a student.
	Student Notifications - All the components related to sending and viewing Student Notifications.
	Message From Company - Notification for a student telling them they have a new message from a company.
	Post From Company - Notification for a student telling them that a company they follow has a new post.
	Notification From Company - Notification for a student telling them that a company has pushed a notification to them.
	Profile View From Company - Notification for a student telling them that a company viewed their profile.
	Notification Filters - Filters that a student can apply to their notifications to determine which ones to receive.
	Notification - The actual notification the student receives.
	Notification Summary - A brief summary of the notification.
	Notification Details - The full details of the notification.
	List of Notifications - All notifications for a given student.
	Email Generator - Adaptor for creating an email to send to an external email.
	External Email - Sent email notifications go to the external email service.
Referenced By	2.3.22, 2.3.23,
Viewpoint	Component Diagram

2.3.22. Students Notifications: UML Class Diagram



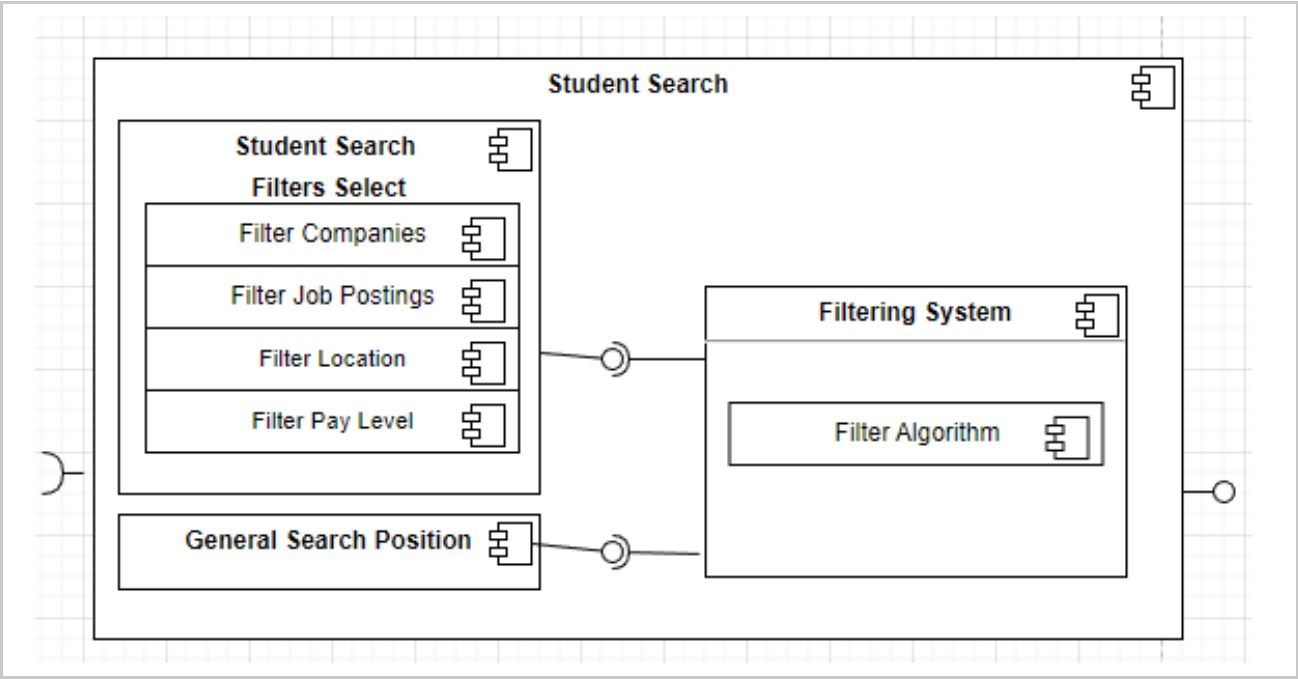
Name	**Front End Student Notification Class Diagram**
Purpose	Show classes required for the student-side implementation of notifications.
Description	Multi-class UML diagram for student notifications.
Requirements	3.2.1, 3.2.2, 3.2.3, 3.2.5, 3.2.7
Elements	**Notification:** The notification stores the sender, recipient, and content data.
	Notification Composer: Composes the content of the notification based on the **Trigger** and pushes it to the **Notification Center**.
	Trigger: Enum of possible notification triggers passed to the **Notification Composer**.
	Notification Center: Interface for interacting with notifications, storing, and allowing the user to view, save, or delete each one.
Referenced By	2.3.21, 2.3.23
Viewpoint	UML Class Diagram

2.3.23. Students Notifications: Sequence Diagram



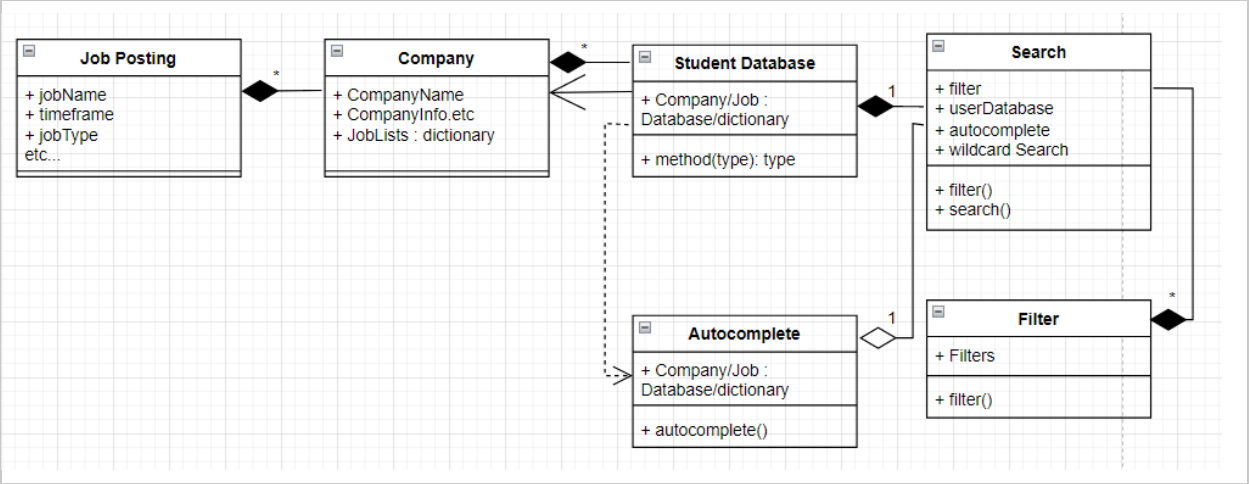
Name	**Student Notification Sequence Diagram**
Purpose	Show the possible sequences of events between student users and the notification system.
Description	This sequence diagram describes the flow of interaction between different application layers based on user input and stored data.
Requirements	3.2.1–3, 3.2.5
Elements	**Frontend:** A representation of the user interface for the program through which all requests are made and all notifications delivered.
	Backend: A representation of the program's backend, responsible for interfacing with the database and handling the different notification triggers.
	Database: A representation of the database stores all **notification data** for use.
	Notification data: Data contained in internal messages and shared through the notification system.
	Push Notification: internal or externally forwarded email notification of messages received.
Referenced By	2.2.5, 2.2.17–18, 2.2.27, 2.3.21, 2.3.22
Viewpoint	Sequence Diagram

2.3.24. Students Search: UI Components



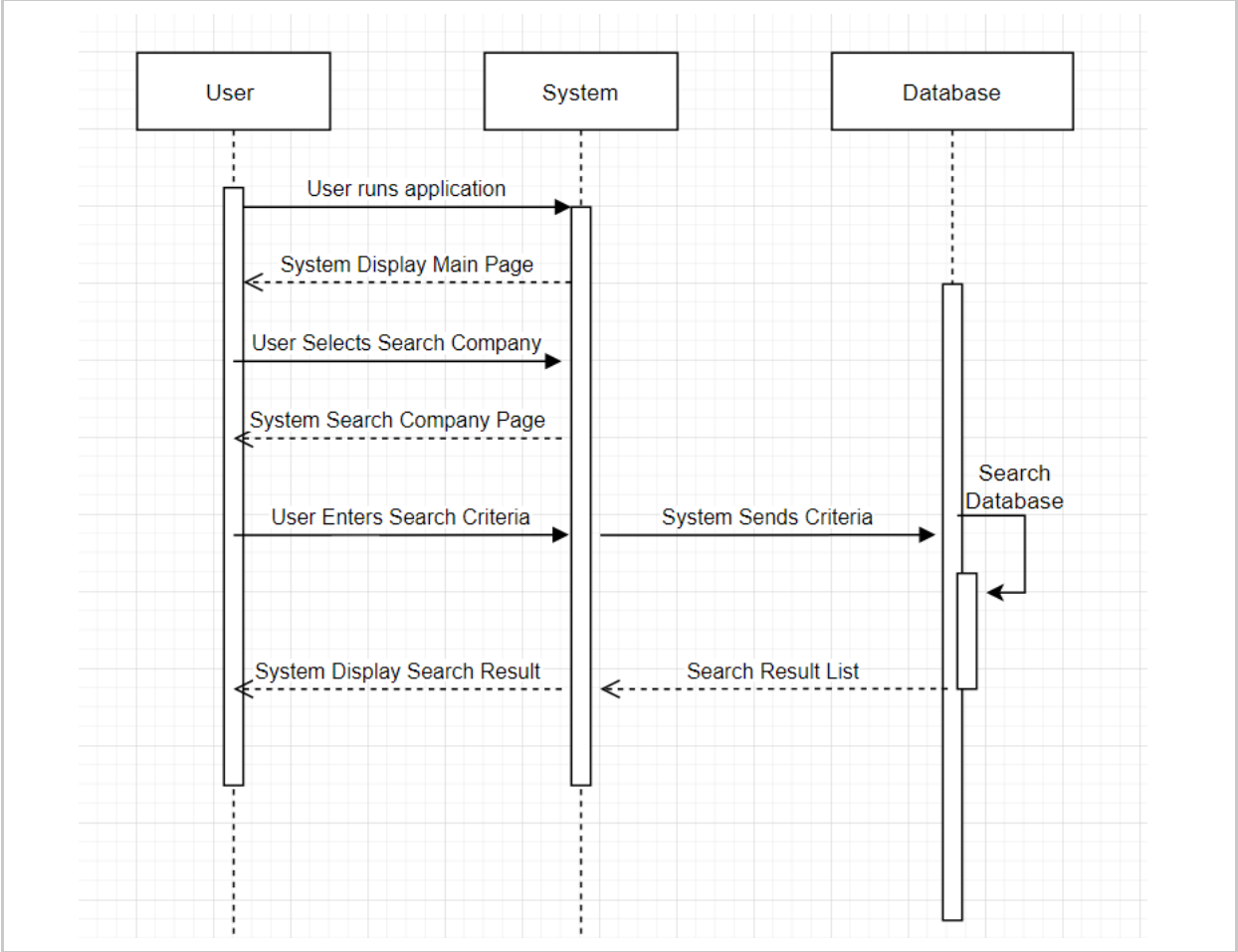
Name	**Student Search UI Components**
Purpose	Display components for UI system
Description	The logic for student search input and resulting filtered results
Requirements	2.1.3.6, 3.3.1 - 3.3.4
Elements	**Filter Companies:** Filter companies a user may or may not want to hire with.
	Filter Job Postings: Filter postings based on date.
	General Search description: Job type and general titles.
	Filter Location: Preferred state, location, area.
	Filter pay Level: Preferred salary dictated by the user.
Referenced By	2.2.57, 2.3.40, 2.3.41, 2.3.25,
Viewpoint	Component Diagram

2.3.25. Students Search: UML Class Diagram



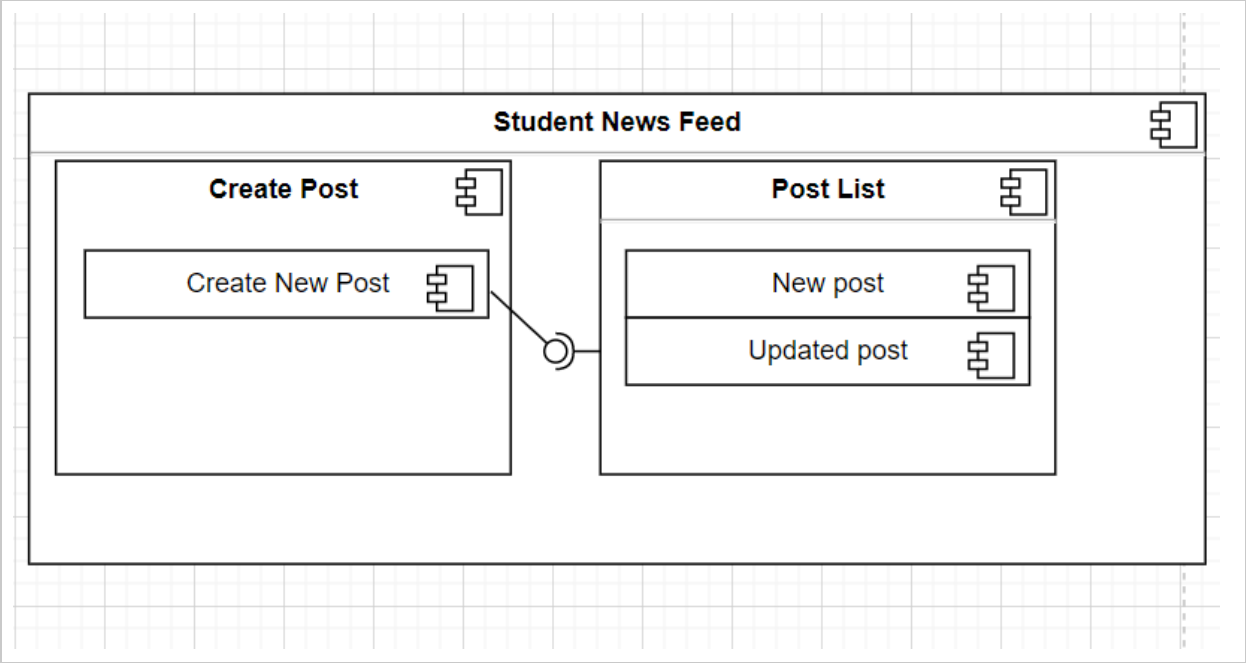
Name	**Student Search UML Class Diagram**
Purpose	Lay out the basic classes and inheritance between said classes for the Student side of the Search Page.
Description	The diagram describes a structured class diagram of the necessary classes and a supposed structure to make the Student side Search page work.
Requirements	3.3.1 - 3.3.4
Elements	**Search**: the search function that runs everything.
	Filter: allows the user to filter the results according to the different specifications taken from the company/job database and other information.
	Student Database: Where the users are stored.
	Autocomplete: completes the user's search text taking data from the database.
	Company: company information and job listings.
	Job Posting: contains the information specific to the job; this is what the student is trying to find.
Referenced By	2.3.24, 2.3.26
Viewpoint	UML Class Diagram

2.3.26. Students Search: Sequence Diagram



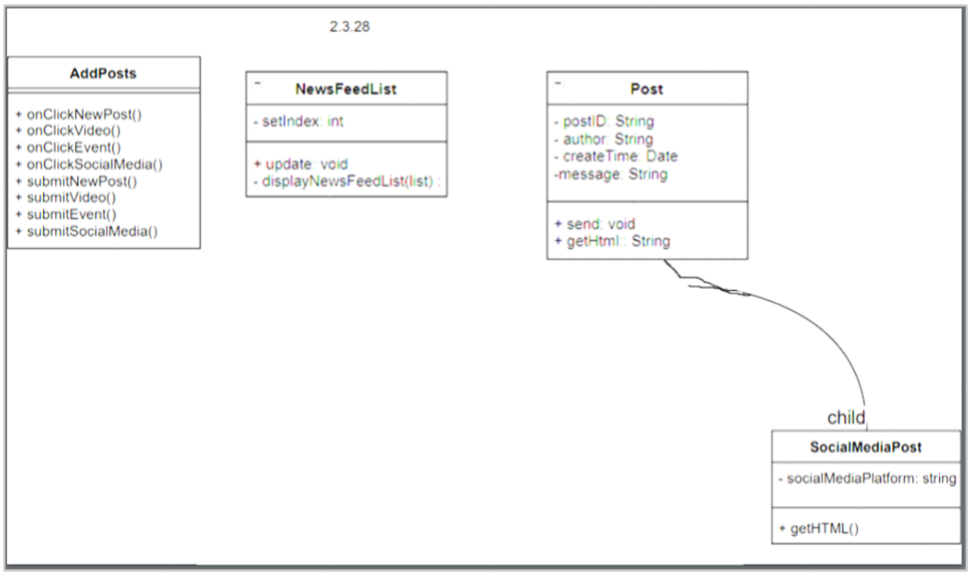
Name	**Student Search Sequence Diagram**
Purpose	To illustrate the sequencing of events from the student's perspective during a search.
Description	A diagram showing the sequence of events for a search from a student.
Requirements	3.3.1 - 3.3.4
Elements	**User:** A representation of the student performing actions against the system.
	System: A representation of the logic that communicates between the user and the database.
	Database: A representation of the collection of data in need of searching.
	Events: Actions performed by any of the three entities throughout the session.
	Results: Returned search results from the database to the system, then displayed to the user.
Referenced By	2.3.33
Viewpoint	Sequence Diagram

2.3.27. Student News Feed: UI Components



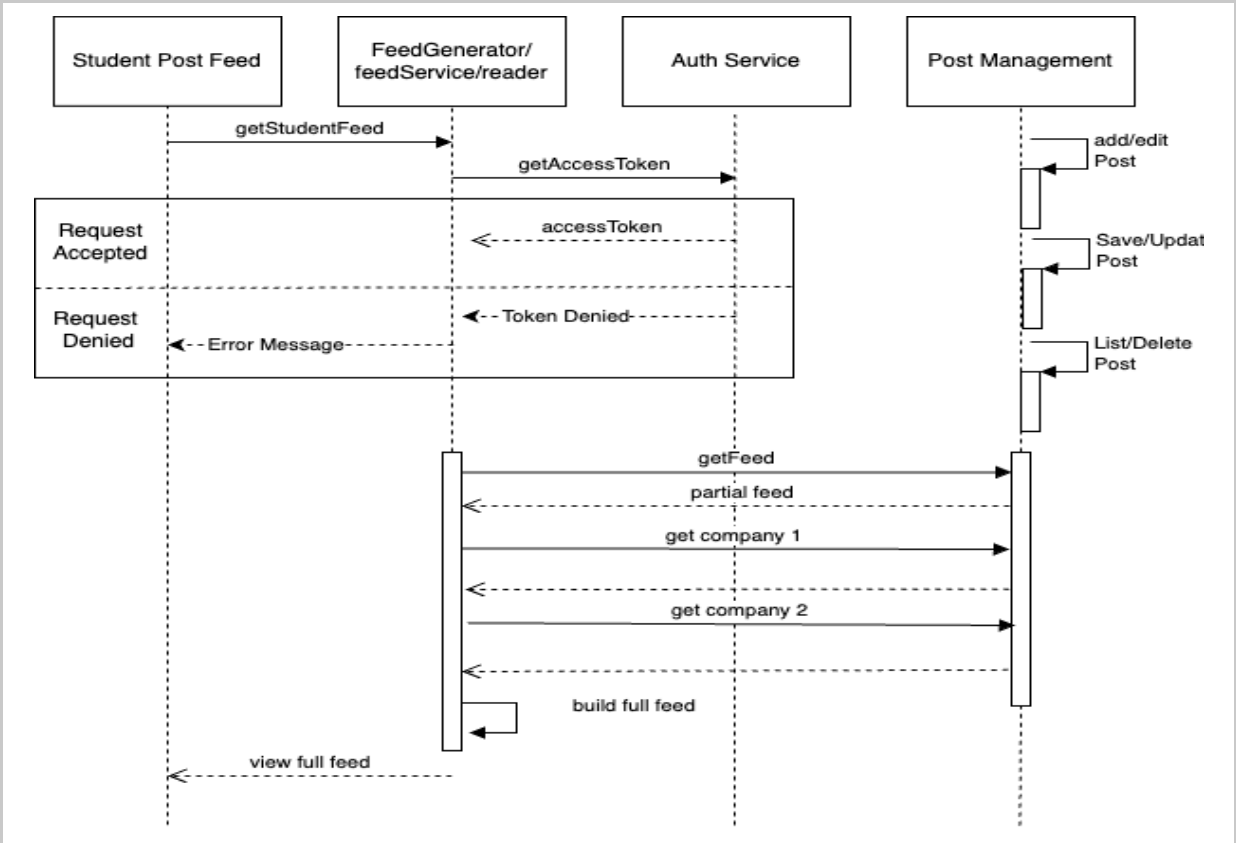
Name	**Student News Feed: UI Components**
Purpose	Display the components of the student news feed and their interactions
Description	Create Post allows us to add a post to the list, which is populated and updated with a new post from followed companies
Requirements	3.4.1. , 3.4.2
Elements	**Create Post:** Text editor for adding a post.
	Post List: A list of all posts followed by the user.
	Create New Post: Submits a new post.
	New Post: Displays a post submitted by the user.
	Updated Post: Refreshes the new posts by followed companies
Referenced By	2.3.28
Viewpoint	Student News UI Components

2.3.28. Students News Feed: UML Class Diagram



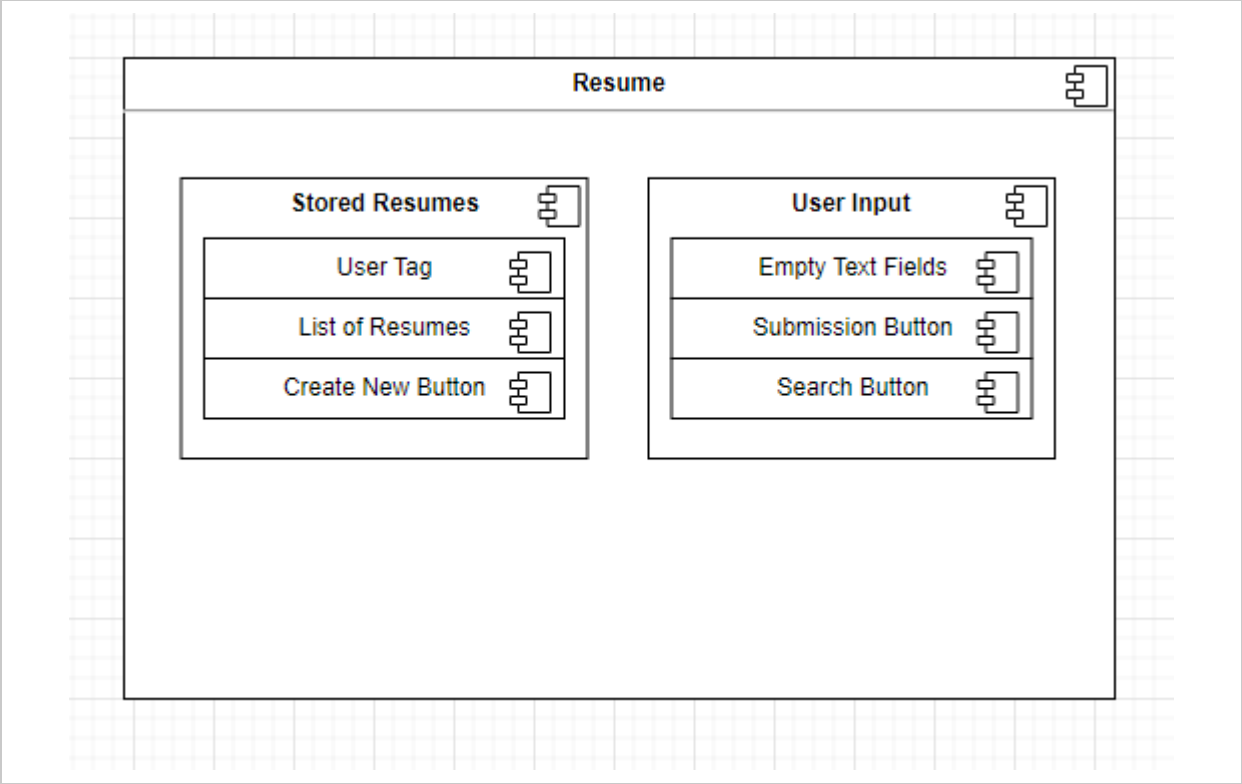
Name	**Students News Feed: UML Class Diagram (Frontend)**
Purpose	Describes the front-end aspect of the Student News Feed
Description	Features include creating Student posts and displaying company posts on the Student's News Feed
Requirements	2.4.1., 2.4.6., 3.4.1., 3.4.2.
Elements	**AddPosts**: Provides methods for creating different types of posts and then submitting them
	NewsFeedList: A list of Post objects retrieved from the backend. This class displays the list, and the list can be updated when the user scrolls down to see more company posts
	Post: A parent class for all types of Company Posts that the Student user might see. This base class can display the text in a company's post and allows the Student user to respond to them. Child classes add to this functionality, although it can work independently for text-only posts.
	SocialMediaPost: Extends the Post class to allow a Post to "link back to a company's social media." (Quoted from requirement 2.4.6)
Referenced By	2.3
Viewpoint	UML Class Diagram

2.3.29. Students News Feed: Sequence Diagram



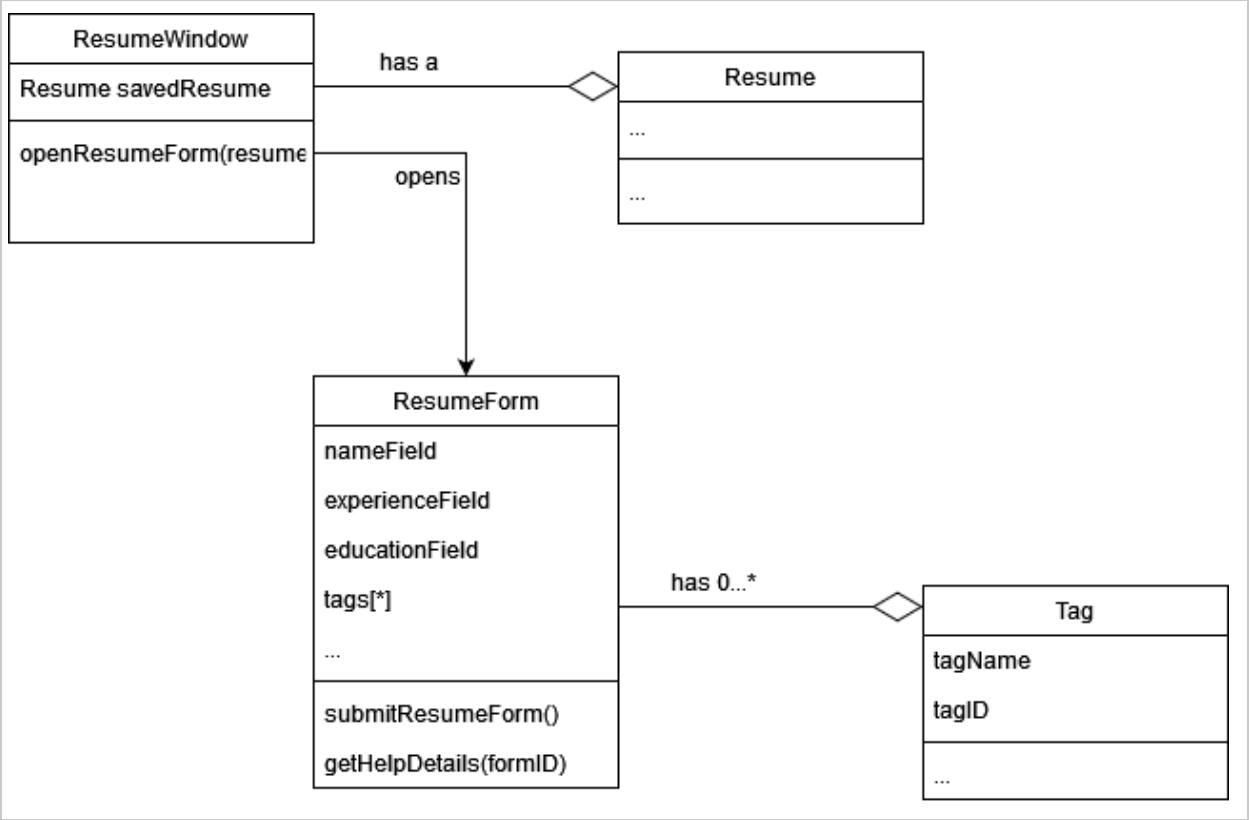
Name	**Student Newsfeed Sequence Diagram**
Purpose	To demonstrate the sequence to view student news feeds.
Description	Post feed is populated with posts. These posts interact with Post Management, giving the ability to add, delete, remove, and update posts that a user owns.
Requirements	2.4.1
Elements	**Post Feed:** List of all posts for viewing.
	Feed Generator: Generates feed to display based on followed users and universities.
	Authentication Service: Confirms user authentication information.
	Post Management: Allows for creating, editing, and deleting users' posts. The system to control updates on posts integrates with ACL
	Delete Post: Removes posts from feeds.
	Edit Post: Changes content of Post in Feeds.
	Add Post: Adds a post into Feeds.
Referenced By	1.3.1, 1.3.4, 2.4
Viewpoint	Sequence Diagram

2.3.30. Students Resume: UI Components



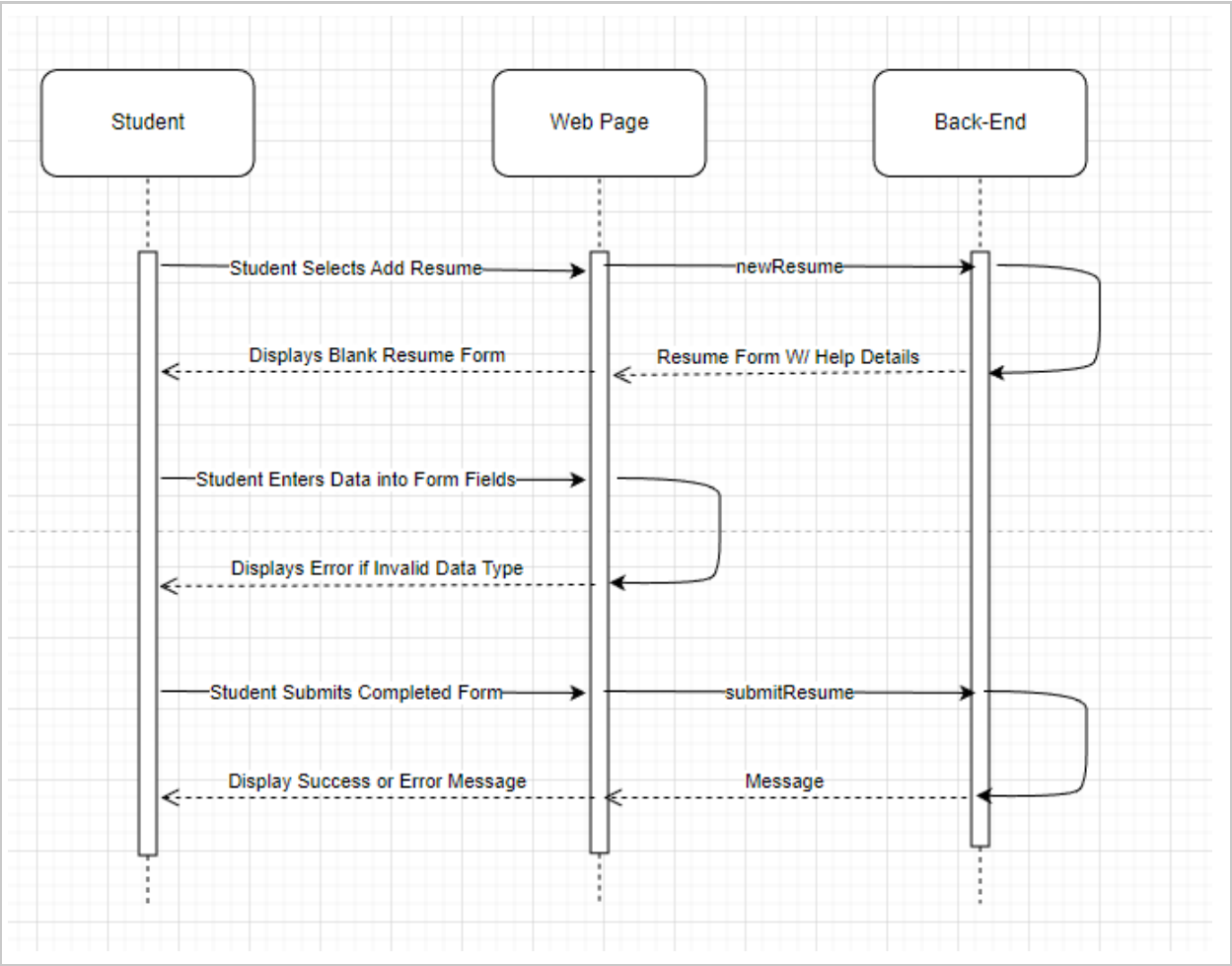
Name	**Resume UI Components**
Purpose	Describes the UI aspects of the Resume Requirements
Description	Describes the User Interface according to the Resume Requirements documentation.
Requirements	3.5
Elements	**Stored Resume:** Resumes are stored in a stacked list by user tag assortment.
	User Input: User Interface aspects that allow for input from the user.
	Create New Button: Gives the user the ability to create a new resume.
	Submission Button: Gives the user the ability to submit a finished resume.
	Search Button: Allows the user to search for other resumes.
Referenced By	2.3.32
Viewpoint	UI Components

2.3.31. Students Resume: UML Class Diagram



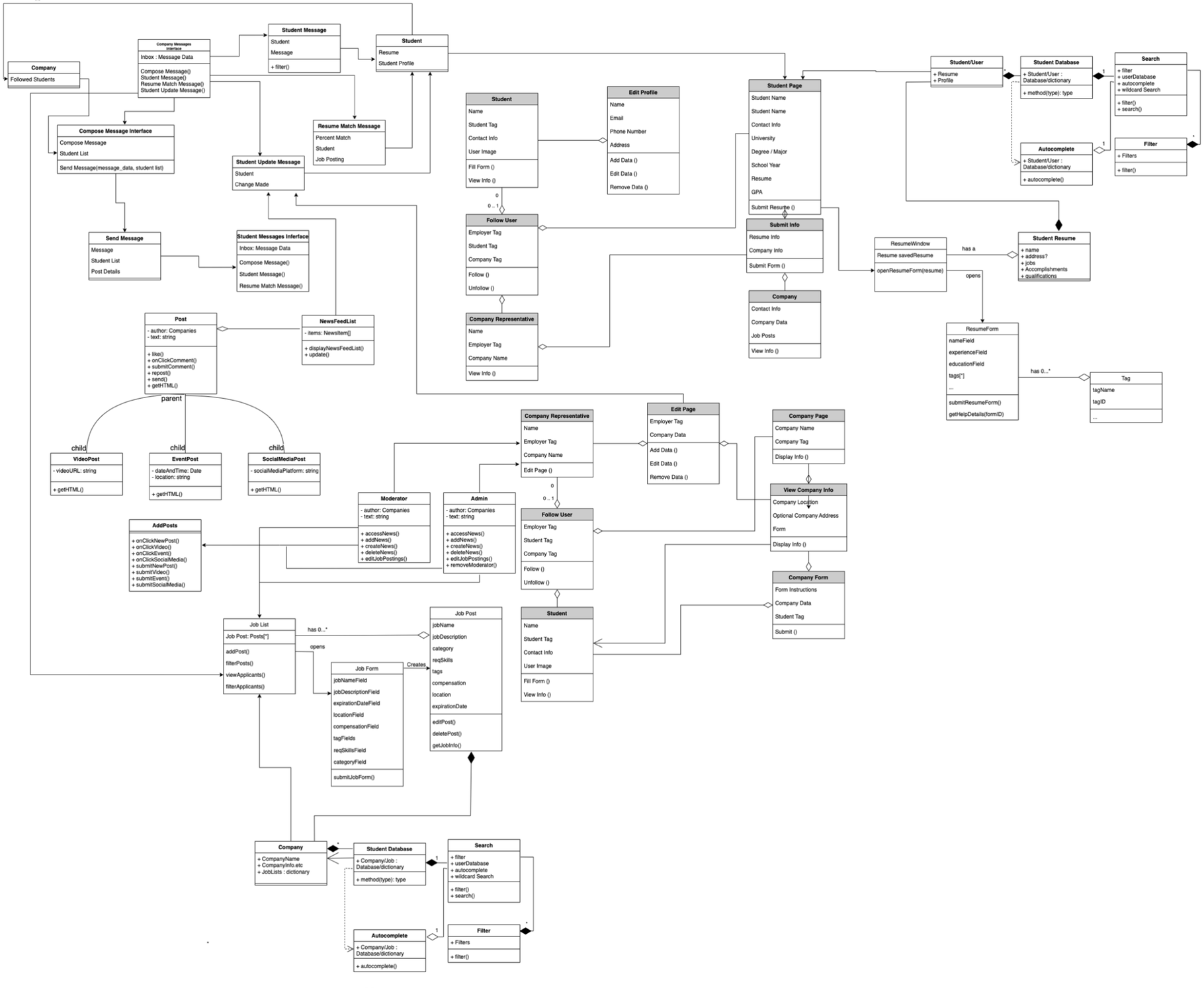
Name	**Student Resume Diagram**
Purpose	To demonstrate the logic behind a Student's resume view.
Description	A diagram displaying the classes used to view or edit resumes and how they interact.
Requirements	3.5.1, 3.5.2, 3.5.3
Elements	**Resume Window:** The window where a student user can view their current resume or navigate to the resume form.
	Resume: A document outlining a student's qualifications, accomplishments, or qualities desirable for employment.
	Resume Form: A form that allows a student to fill out details for their resume.
	Tag: A short descriptive identifier that can be used to filter resumes.
Referenced By	2.3
Viewpoint	Class Diagram

2.3.32. Students Resume: Sequence Diagram



Name	**Student Create Resume Sequence Diagram**
Purpose	To show the sequence of a student creating a resume.
Description	Students will use the front-end UI to communicate their desire to create their resume in the web app. Next, the UI will communicate to the backend to receive the UI form with its help details. Students will then need to submit their resume to be saved in the web app's DB; the UI will forward this to the back end to complete the action.
Requirements	3.5.1, 3.5.2, 3.5.3
Elements	**Student:** A user in the role of student.
	Web Page/UI: The user will visually see and interact with the interface.
	Backend: The interface that the software will interact with but not the user.
Referenced By	3.5
Viewpoint	Sequence Diagram

2.3.33. All Frontend UML Class Diagrams



Name	**All Frontend UML Class Diagrams**
Purpose	Describes the interfaces among all Frontend classes.
Description	Contains the relationship of all classes in the logic layer.
Requirements	2.1.1 – 3.4.2
Elements	**Add Posts:** The action that the UI performs to add posts to the database (through API endpoints)
	News Feed List: The list of elements to be displayed on a given user's news feed
	Post: An element to be displayed on a news feed
	Video Post: A post containing a video element
	Event Post: A post containing a gathering/event element
	Social Media Post: A post containing some form of social media, assumably from an external social media site
	Company Employee: A token/object representing a company employee
	Moderator: A Company Employee with Moderator rights
	Admin: A Company Employee with Admin rights
	Company Page: A page displaying all the information for a company
	View Company Info: A link or action getting and possibly displaying all the information for a company
	Company Form: A form, either used to edit or otherwise interact with a company
	Follow User: The action that sets the user to follow another user
	Edit Page: The action allowing the user to modify the page
	Edit Profile: The action allowing the user to modify their profile
	Student: The token/object representing a student
	Search: The action allowing the user to search a database
	Filter: The action allowing the user to filter the results of a search action
	Autocomplete: The functionality for the program to provide suggestions on what the user is typing in a text input field

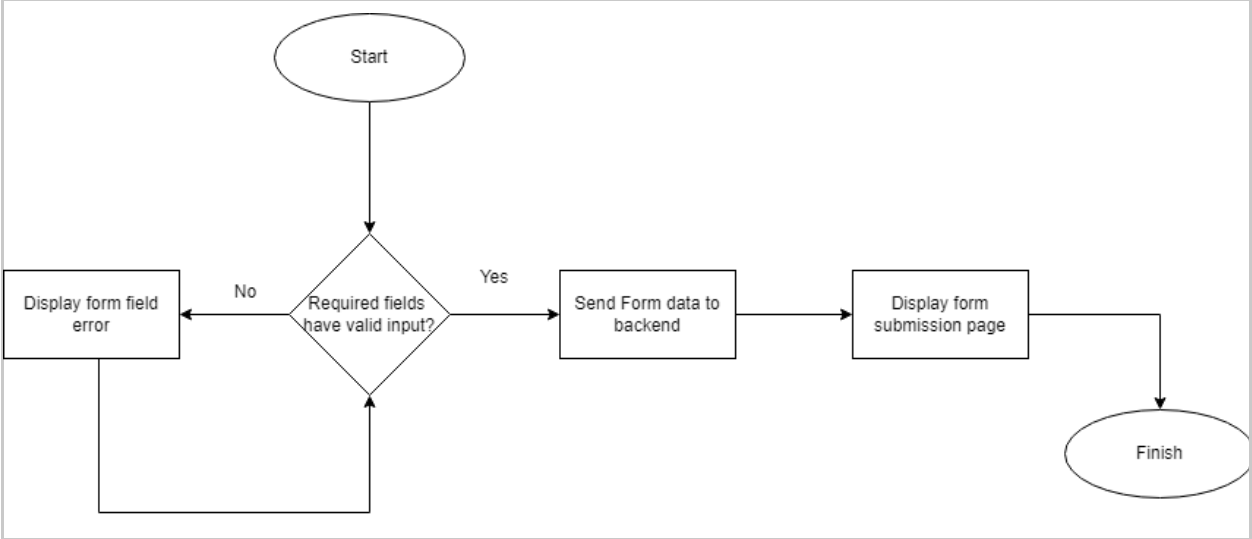
	Student Database: The backend storage/database for the students
	Company Database: The backend storage/database for the companies
	Student Resume: The resume document of a student
	Resume Window: The page/element displaying the student resume (document)
	Resume Form: A form allowing the user to create/modify their resume
	Company Message: An element which contains a message to be sent from a company
	Student Message: An element which contains a message to be sent from a company
	Company Message Interface: The UI and/or abstract data type for a company message
	Student Message Interface: The UI and/or abstract data type for a student message
	Send Message: The action allowing the user to send a message (see "student message", "company message")
	Resume Match Message: The message containing the information for a "resume match" database event notification
Referenced By	2.1.1 – 3.4.2
Viewpoint	UML Class Diagram

2.3.34. Company.viewInfo()

```
viewInfo()
   PUT company info on screen
```

Name	**Company.viewInfo()**
Purpose	Display information.
Description	Displays company information to the user
Requirements	2.1.3.1
Inputs	None
Outputs	None
Referenced By	2.1.3.10
Viewpoint	Pseudocode

2.3.35. SubmitInfo.submitForm()



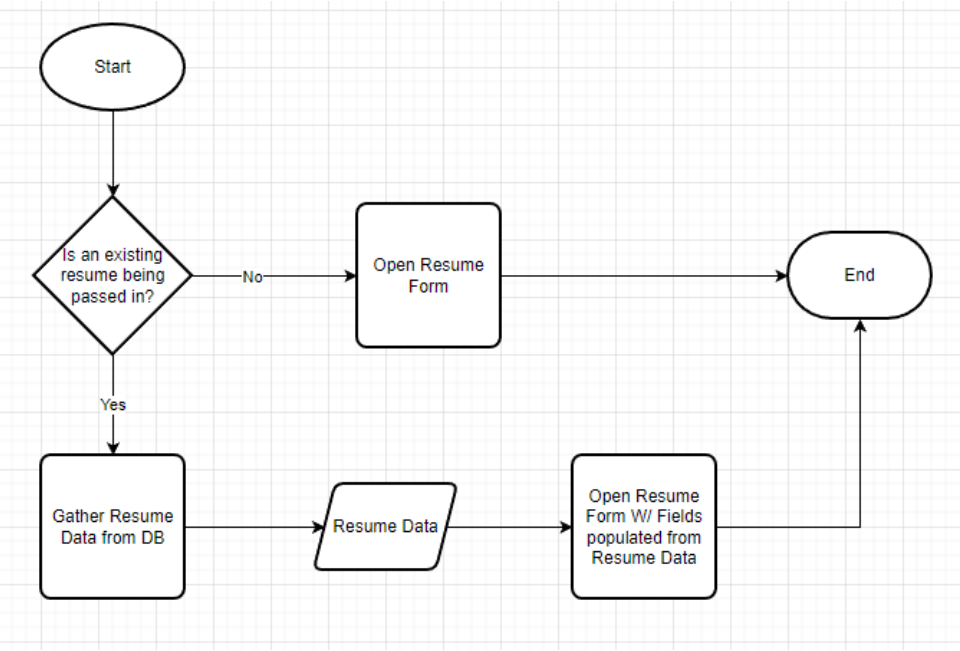
Name	**SubmitInfo.submitForm() (Related to StudentPage)**
Purpose	Submits a populated form to the backend for storage.
Description	Validates form field entries. Sends form data to backend. Displays appropriate pages based upon form submission success/fail.
Requirements	3.1.2, 3.1.3.1, 3.1.3.2
Inputs	Form field data
Outputs	None
Elements	**Field:** a user input text box, check box, etc.
	Form Field Error: a helpful message which indicates to the user invalid input data.
	Submission Page: a page that indicates that the form was submitted successfully.
	Form Data: first name, last name, address, education experience, bio, etc.
Referenced By	2.3.33
Viewpoint	Flowchart

2.3.36. StudentPage.submitResume()

```
StudentPage::submitResume()
    submitInfo.submitForm()
```

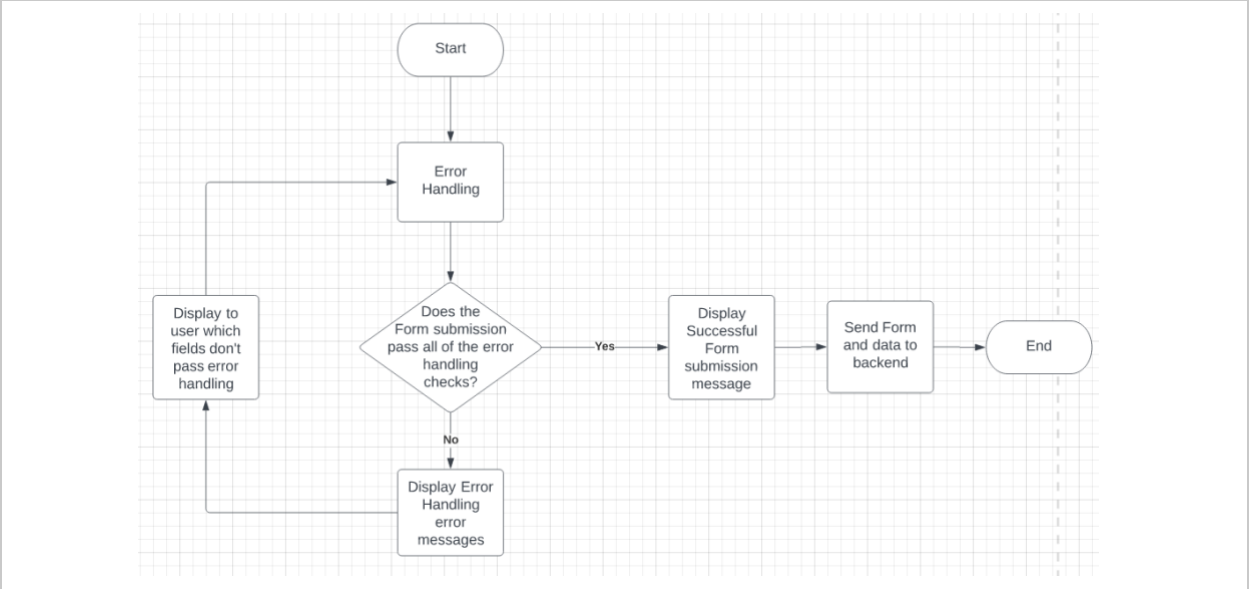
Name	**StudentPage.submitResume()**
Purpose	Submits a resume for the student.
Description	Submits the resume using the Resume Form.
Requirements	3.5.1, 3.5.2, 3.5.3
Inputs	None
Outputs	None
Elements	**SubmitInfo:** Class used to submit a resume.
Referenced By	2.3.52
Viewpoint	Pseudocode

2.3.37. ResumeWindow.openResumeForm()



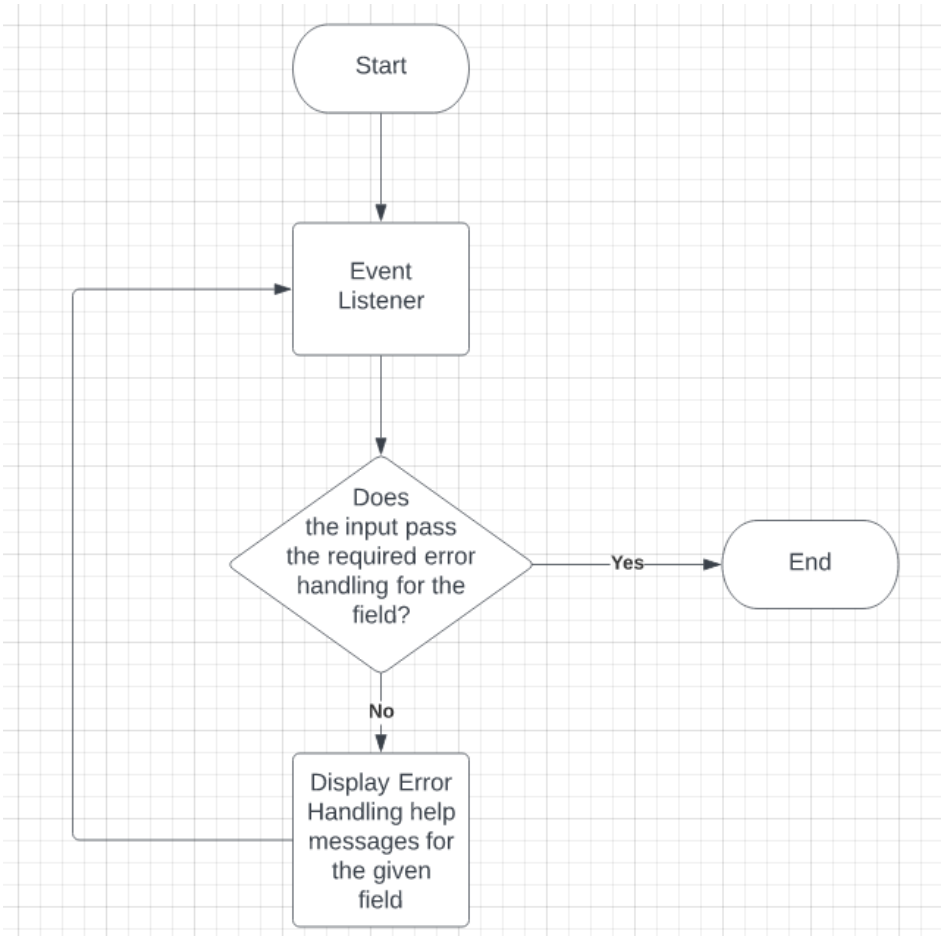
Name	**ResumeWindow.openResumeForm()**
Purpose	Opens Resume Form for user
Description	Opens Resume Form for user to be able to populate form with their resume info
Requirements	3.5.1, 3.5.2, 3.5.3
Inputs	A resume
Outputs	Resume form
Elements	**ResumeForm:** Class used to create/edit resumes
Referenced By	2.2.13, 2.2.40, 2.3, 2.3.30, 2.3.31
Viewpoint	Flowchart

2.3.38. ResumeForm.submitResumeForm()



Name	ResumeForm.submitResumeForm()
Purpose	To submit the requested resume form
Description	This function is meant to submit the resume form and the data embedded within.
Requirements	3.1.3.2, 3.5.5
Inputs	Filled out Form and event listener for button press
Outputs	Returns error handling or success and form data
Elements	**Error Handling:** Checks for discrepancies in the data provided before submission.
	Boolean Logic: Ensures proper Error Handles are met before submission.
	Success Display: Displays a successful submission.
	Return Form Data: Returns form data to be sent to the back end.
	Error Handling Display: Displays errors located in the form when found.
Referenced By	2.3.37, 2.3.36, 2.3.35, 2.3.33
Viewpoint	Flowchart

2.3.39. ResumeForm.getHelpDetails()



Name	**ResumeForm.getHelpDetails()**
Purpose	To call an individual field error handling check
Description	This function calls individual field error handling checks to display the correct input needed for a given field.
Requirements	3.5.4.2
Inputs	Event Listening
Outputs	Display Error Handling Help Messages
Elements	**Event Listener –** These listen for incorrect input through error handling or for a help button click.
	Error Handling Check – A Boolean checking to see if the input passes the error handling?
	Display Error help Message – This displays the error with each given field.
Referenced By	2.3.33
Viewpoint	Flowchart

2.3.40. Search.filter()

```
filter(filterValue)
   SET index <- 0
   SET filteredList <- []

   WHILE (index < number of items in the list)
      IF (list[index].contains(filterValue))
         filteredList.append(list[index])
      INCREMENT index by 1
   RETURN filteredList
```

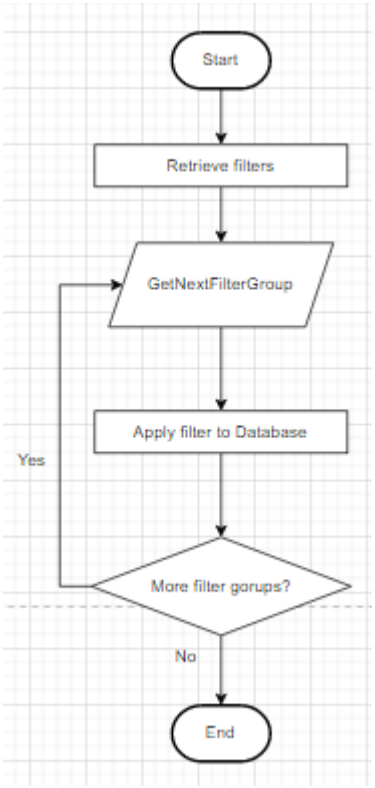
Name	**Search.filter()**
Purpose	Filter for a specific resume, job posting, company, or student
Description	Filter will filter through the resume, job posting, company, or student with the string in the filter bar.
Requirements	2.3.2-2.3.5, 2.3.8, 2.5.8, 3.2.7, 3.3.3-3.3.4
Inputs	Filter String
Outputs	Filter Results
Elements	Filter
Referenced By	2.3
Viewpoint	Pseudocode

2.3.41. Search.search()

```
Search()
INITIALIZE index = 0
WHILE (index < number of items in the list)
             IF (list[index] == target element)
                    RETURN list[index]
             INCREMENT index by 1
RETURN -1
```

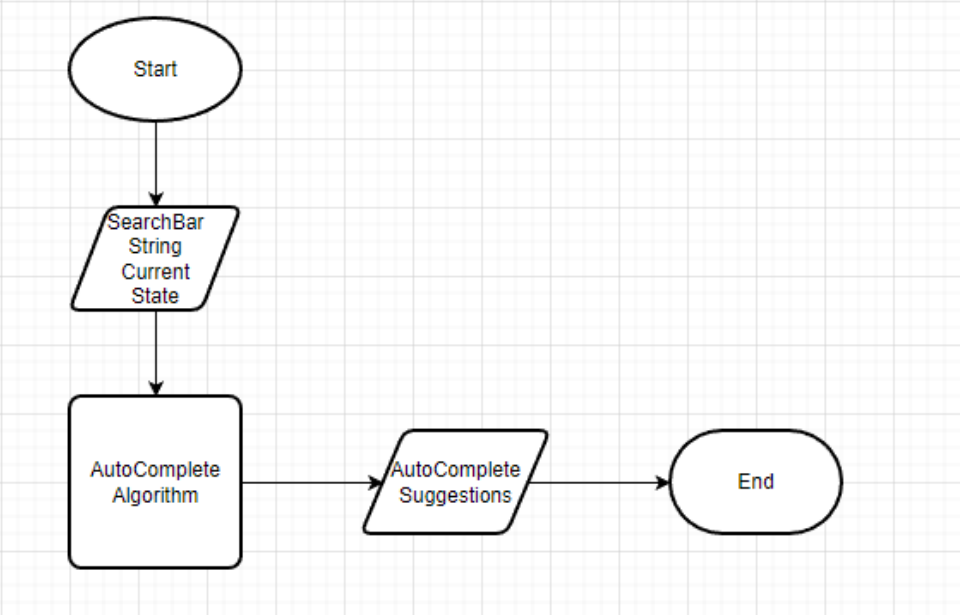
Name	**Search.search()**
Purpose	Search for a resume, job posting, company profile, or student profile page.
Description	Search will search for the resume, job posting, company, or student with the string in the search bar.
Requirements	2.3.1 - 2.3.8, 3.1.3.1 – 3.1.3.4
Inputs	**Search String**
Outputs	**Search Result**
Elements	**Search:** The search method.
Referenced By	2.3, 2.3.10, 2.3.25, 2.3.33
Viewpoint	Pseudocode

2.3.42. Filter.filter()



Name	**Filter.filter()**
Purpose	Filters Data retrieved from the Database
Description	Accepts input from user for chosen filters, and tells
Requirements	2.3.1, 2.3.2, 2.3.3, 3.3.1
Inputs	**User-chosen Filters**
Outputs	None
Elements	**Filter Groups**: A collection of users chosen filters, not including the actual search string
	Filter: Selection from the user to restrict search results, often fitting into prespecified categories, that are chosen when searching. This doesn't not include the search string entered by the user.
	Database: The online database where all the Jobs Listings, Companies, Student Profiles and Resumes are stored.
Referenced By	2.2.3
Viewpoint	Flowchart

2.3.43. Autocomplete.autocomplete()



Name	**Autocomplete.autocomplete()**
Purpose	Auto Completion Suggestions for Search Bar
Description	An AutoCompletion algorithm will run on the current state of the string in the search bar returning suggestions for auto completion of the string
Requirements	2.3.6
Inputs	Search String
Outputs	String Suggestions
Elements	Search Bar
	Auto Complete
Referenced By	2.3.12
Viewpoint	Flowchart

2.3.44. CompanyRepresentative.viewInfo()

```
viewInfo()
Name <- get(name)
Company <- get(Company Name)
tag <- get(Company Tag)
displayInfo()
```

Name	**CompanyRepresentative.viewInfo()**
Purpose	Request the Company representative’s info to be displayed
Description	Get the variables of the company representative and display them
Requirements	2.3.1.8
Inputs	Name: a string that is used to pull a person’s company info Company Name: a string that is used to pull the company's information.
Outputs	displayInfo(): a UI box that shows formatted information of the company.
Elements	Name: get a name of representative
	Company: get a name of a company
	Tag: get the company's tag
Referenced By	2.3.44
Viewpoint	Pseudocode

2.3.45. CompanyRepresentative.editPage()

```
editPage(Edit)
onclick(data)
       onclick -> Choice
IF (Choice == Edit)
       GET -> New_Data
       Send(Data, Choice, New_Data)
ELSE IF(Choice == Remove)
Send (data, Choice)
onClick(AddNew)
GET -> New_Data
       Send(Choice, New_Data)
```

Name	**CompanyRepresentative.editPage()**
Purpose	Allow the company representative to edit a page
Description	The company representative will click what and the type of change they desire to make, and then input the new data which will be sent along with the type of change to a function in the backend.
Requirements	2.1.3.10
Inputs	Data: a button that selects that Data one desires to change Onclick / Choice: a button that will select the desired change NewData: an input field to enter the altered data
Outputs	New data sent with a choice
Elements	**Data:** Get the users data
	Choice: The users' choices for editing the data
	Cin: having the ability to input the data
	Send: Send the new data along with its choice
Referenced By	2.2.1,2.2.2, 2.2.3, 2.3.44
Viewpoint	Pseudocode

2.3.46. Student.fillForm()

```
FillForm()
   SET <- name
   SET <- age
   SET <- education
   SET <- address
   GET -> name
   GET -> age
   GET -> education
   GET -> address
   RETURN name, age, education, address
```

Name	**Student.fillForm()**
Purpose	Allow the student to fill in information
Description	Allow the student to fill in the information so it can be added to the student’s profile page
Requirements	3.1
Inputs	Data from the user
Outputs	User’s information
Elements	Name: gets user’s name
	Age: gets user’s age
	Education-gets user’s education
	Address-gets user's address
Referenced By	2.3
Viewpoint	Pseudocode

2.3.47. EditProfile.addData()

```
addData()
   currentUser <-- GET PageData.currentUser
   PUT user --> data[currentUser].userProfile
   INSERT users <-- data change where currentUser info is
          PageData.currentUser
```

Name	**EditPage.addData()**
Purpose	Method for getting the data from the EditPage to be added to the user's profile.
Description	This Method will take the data from the EditPage and add the information on the user's profile page of the current user.
Requirements	2.1.3, 3.1.2
Inputs	The data from the form of the user's page
Outputs	Added data viewable on the profile page
Elements	PageData: Data on the user's page to be altered.
	CurrentUser: The current user logged in
	Users: the appropriate users database (referring to either the students or companies' databases)
	UserProfile: The user's profile page
Referenced By	1.3.2, 1.3.5,1.3.8, 1.3.9
Viewpoint	Pseudocode

2.3.48. EditProfile.editData()

```
editData()
   currentUser <-- GET PageData.currentUser
   PUT user --> data[currentUser].user Profile
   UPDATE users <-- data change where currentUser info is
          PageData.currentUser
```

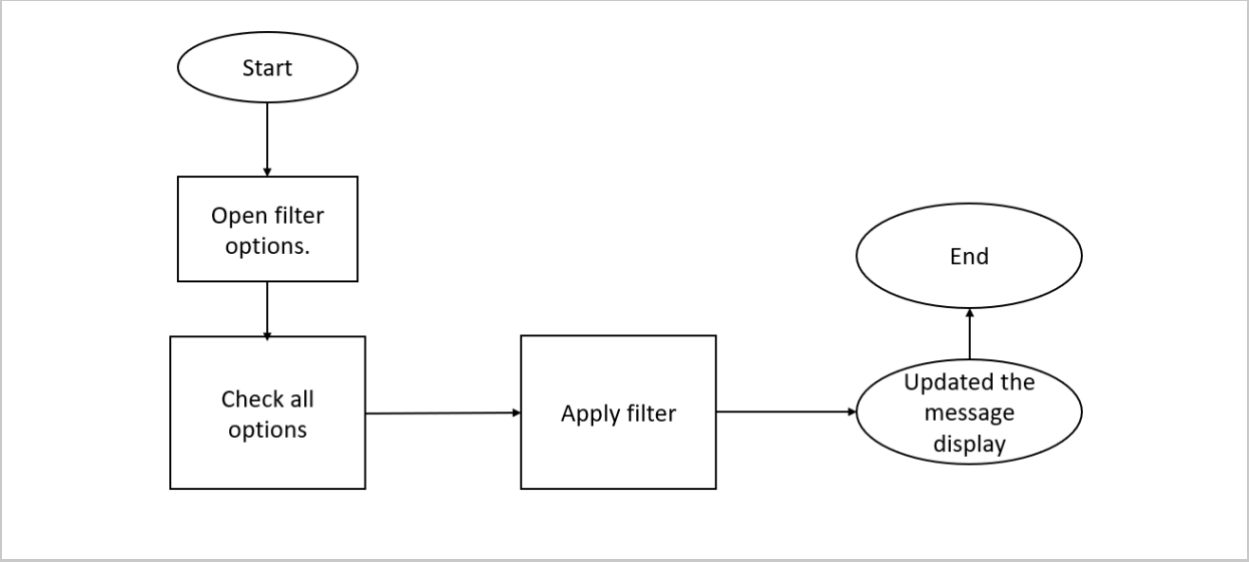
Name	**EditPage.editData()**
Purpose	Method for getting the data from the EditPage to be updated to the user's profile.
Description	This Method will take the data from the EditPage and update the information on the user's profile page of the current user.
Requirements	3.1.2
Inputs	The data from the form of the user's page
Outputs	Updated data viewable on the profile page
Elements	PageData: Data on the user's page to be altered.
	CurrentUser: The current user logged in
	Users: the appropriate user's database (referring to either the student's or companies' databases)
	UserProfile: The user's profile page
Referenced By	1.3.2, 1.3.5,1.3.8, 1.3.9
Viewpoint	Pseudocode

2.3.49. EditProfile.removeData()

```
removeData()
   DELETE FROM data change where currentUser info is PageData.currentUser
```

Name	**EditPage.removeData()**
Purpose	Method for removing selected data from the user's profile.
Description	This Method will remove the information/data on the user's profile page of the current user.
Requirements	3.1.2
Inputs	The data from the form of the user's page
Outputs	Data deleted is no longer viewable on the profile page
Elements	PageData – Data on the user's page to be altered.
	CurrentUser – The current user logged in
	Users – the appropriate users database (referring to either the students or companies' databases)
	UserProfile – The user's profile page
Referenced By	1.3.2, 1.3.5,1.3.8, 1.3.9
Viewpoint	Pseudocode

2.3.50. StudentMessage.filter()



Name	StudentMessage.filter()
Purpose	Display options to filter student user's messages
Description	Allow filtering of different messages based on user's preferences.
Requirements	3.2.7
Inputs	List of messages
Outputs	List of messages limited by filters
Elements	**Filter options:** List of fields that can be filtered
	Apply filters: Submit button
Referenced By	2.2.1, 1.3.9, 1.3.9, 1.3.12
Viewpoint	Flowchart

2.3.51. CompanyMessagesInterface.composeMessage()

```
composeMessage(Recipient)
   PROMPT for message body
   SEND message to Recipient
```

Name	**CompanyMessagesInterface.composeMessage()**
Purpose	Allow the user to compose a message
Description	The UI component of message composition
Requirements	2.2.2
Inputs	Recipient: the message's intended recipient
Outputs	None
Referenced By	2.2.3
Viewpoint	Pseudocode

2.3.52. CompanyMessagesInterface.studentMessage()

```
CompanyMessages:StudentMessage()
    FOR student IN students
        composeMessage()
```

Name	**CompanyMessagesInterface.studentMessage()**
Purpose	Compose a message for all the students.
Description	Creates a message to send to all the students.
Requirements	2.2.2
Inputs	None
Outputs	None
Elements	ComposeMessage is another method in the same class.
Referenced By	2.3.33; 2.3.51
Viewpoint	Pseudocode

2.3.53. CompanyMessagesInterface.resumeMatchMessage()

```
ResumeMatchMessage(api, student, company, msgTemplate)
  companyRepName <- inbox.user.name
  CONCATENATE companyRepName, student.firstName, student.lastName TO msgTemplate
  POST msgTemplate WITH “notification/resumeMatch”
```

Name	**CompanyMessagesInterface.resumeMatchMessage()**
Purpose	Function to send a message in the event of a resume that matches the criteria
Description	Will send a message to the Company informing them that a resume that matches their job posting criteria has been found.
Requirements	2.2.5
Inputs	CompanyID: the id number of the company receiving the message. StudentID: the id number of the student whose resume matches the job criteria. MsgTemplate: A template for the notification to be sent to the company representative.
Outputs	None
Elements	**Inbox:** A member variable that contains a reference to the company rep’s inbox.
	Endpoint: “notification/resumeMatch” tells the backend a resume has matched and triggers a notification and message to be sent.
Referenced By	2.3.33
Viewpoint	Pseudocode

2.3.54. CompanyMessagesInterface.studentUpdateMessage()

```
studentUpdateMessage( api, followingUsers, msgTemplate )
    // "inbox" is a member variable of the Company Messages Interface

    SET companyId ← inbox.user.id
    SET companyRepName ← inbox.user.name

    FOR student IN followingUsers
        SET studentId ← student.id
        SET lastUpdatedProfileServer ← GET "profileLastUpdated" from
          "/student/{id}" endpoint
        IF lastUpdatedProfileServer NOT == student.lastUpdated
            // update the lastUpdated attribute in User class
            // for the student to mirror value from student profile
            SET student.lastUpdated ← lastUpdatedProfileServer

            // Address representative, then list who updated profile

            CONCATENATE companyRepName, student.firstName,
              student.lastName to msgTemplate
            POST msgTemplate with "/notification/profileUpdate"
                endpoint

END
```

Name	CompanyMessagesInterface.studentUpdateMessage()
Purpose	Describe the algorithm of studentUpdateMessage()
Description	studentUpdateMessage() takes the list of students the company is following
Requirements	2.2.4
Inputs	API, followingStudents, msgTemplate
Outputs	None
Elements	Inbox – An attribute of the CompanyMessagesInterface
	api – an instance of the API used for interacting with the data layer. Endpoints used are "/student/{id}" and "notification/profileUpdate".
	followingStudents – the array of students that are being followed by the company to be iterated through.
	msgTemplate – the message template that is available from the NotificationTemplates object.
Referenced By	3.2.1
Viewpoint	Pseudocode

2.3.55. ComposeMessageInterface.sendMessage()

```
ComposeMessageInterface::sendMessage(messageData, studentList, companyID)
    IF messageData IS empty:
        RETURN
    IF studentList IS empty:
        RETURN
    POST (companyID, studentList) to "/notification/company/message"
    POST (companyID, messageData, studentList) to "/message"
```

Name	**ComposeMessageInterface.sendMessage()**
Purpose	Sends a message from a company to all the students following it.
Description	What happens when the user hits the “Send” button on a message. Will not do anything if there is no message to send, or if there are no students to send to. Otherwise, sends a notification to all recipients saying they have a new message from a company they follow, and sends the message to their inbox.
Requirements	3.2.1 - 3.2.3
Inputs	**messageData:** The contents of the message, as a string. **studentList:** The list of students to whom this message will be sent, as integers. **companyID:** The id of the company sending the message, as an integer.
Outputs	None
Elements	**Endpoint**: “/notification/company/message” Sends a notification to each of the students in studentList telling them they have received a new message from a company they follow.
	Endpoint: “/message” Sends a message from the company to the web app inbox of each of the students in studentList.
Referenced By	2.2.1
Viewpoint	Pseudocode

2.3.56. StudentMessagesInterface.studentMessage()

```
StudentMessagesInterface::StudentMessage(companyId, studentList)
    IF studentList IS empty:
        RETURN
    IF companyId IS null:
        RETURN
    POST (companyId, recipients) to "/notification/company/post"
```

Name	**StudentMessagesInterface.studentMessage()**
Purpose	Send a notification to all following students when a company creates a post.
Description	When a company crafts a post, then it dispatches a notification to any students following the company.
Requirements	3.2.1, 3.2.2
Inputs	**companyId:** the ID of the company who created the post, as an integer.
	studentList: a list of students following the company, each as an integer ID.
Outputs	None
Elements	**Endpoint:** “/notification/company/post” used to dispatch a notification to a list of students and tell them that a new post has been created by a company they follow.
Referenced By	2.3.33
Viewpoint	Pseudocode

2.3.57. NewsFeedList.displayNewsFeedList()

```
displayNewsFeedList(list)
    FOR post of list
        PUT post.getHTML()
```

Name	**NewsFeedList.displayNewsFeedList()**
Purpose	Display each *Post* or news feed item in *list* on the student news feed.
Description	Loops through all *Post*s in *list* and displays them using their generated HTML.
Requirements	3.4.1
Inputs	list
Outputs	HTML written to the DOM.
Dependencies	2.3.62
Execution Time	O(n)
Elements	**list:** The list of Post objects.
	post: An individual Post object in list.
Referenced By	2.3.28, 2.3.60
Viewpoint	Pseudocode

2.3.58. NewsFeedList.update()

```
update()
   SET list <- GET from “/companyFeed/{setIndex}” endpoint
   displayNewsFeedList(list)
```

Name	**NewsFeedList.update()**
Purpose	To retrieve a list of company Posts from the backend and display them.
Description	Retrieves a list of Posts from the backend using a GET request. The setIndex determines which set of Posts to retrieve. For example: if setIndex is 0, get the first, say, 100 Posts. If setIndex is 3, get the 4th set of 100 Posts. Finally, display the list using displayNewsFeedList.
Requirements	3.4.1
Inputs	setIndex
Outputs	list
Dependencies	2.3.59
Execution Time	O(n)
Elements	**setIndex:** The index (a field/property of NewsFeedList) of which list of Posts to retrieve from the backend.
	list: The list of Posts retrieved from the backend.
Referenced By	2.3.28, 2.3.59
Viewpoint	Pseudocode

2.3.59. Post.send()

```
send()
    jsonString <-ßSON.stringify(this)
    POST jsonString to “/studentFeed/{this.postID}/” endpoint
```

Name	**Post.send()**
Purpose	To send this Post to the student news feed backend.
Description	This Post is first converted in JSON format, then is sent to the student news feed backend using POST.
Requirements	3.4.2
Inputs	this
Outputs	jsonString
Dependencies	N/A
Execution Time	O(1)
Elements	**jsonString:** The JSON representation of this Post object.
	postID: The postID property/field of this Post object.
	this: The current Post object
Referenced By	2.3.28, 2.3.60
Viewpoint	Pseudocode

2.3.60. Post.getHtml()

```
getHtml()
    RETURN author + createTime + message
```

Name	**Post.getHtml()**
Purpose	To generate HTML for the purpose of displaying the Post.
Description	Produces HTML that incorporates the author, Post creation date and time, and the Post message.
Requirements	3.4.1
Inputs	The Post object's author, createTime, and message properties/fields.
Outputs	The HTML returned.
Dependencies	N/A
Execution Time	O(1)
Elements	**author:** The author field/property of this Post object in HTML form.
	createTime: The createTime field/property of this Post object in HTML form.
	message: The message field/property of this Post object in HTML form.
Referenced By	2.3.28, 2.3.59
Viewpoint	Pseudocode

2.3.61. EventPost.getHtml()

```
getHtml()
    RETURN Author + Time + Event Post Date + Event Location
```

Name	**EventPost.getHtml()**
Purpose	To generate HTML to display the Event.
Description	Produces HTML that incorporates the author, post creation date and/or time, and the event location.
Requirements	2.4.2, 2.4.4
Inputs	NONE
Outputs	The HTML returned.
Elements	**author:** The author field/property of this Post Event in HTML form.
	createTime: The createTime field/property of this Event object in HTML form.
	Event Date: The Date and Time field/property of this Event object in HTML form.
	Event Location: The location field/property of this Event object in HTML form.
Referenced By	1.3.4, 1.3.5, 3.1, 2.3.28, 2.3.33, 2.3.59
Viewpoint	Pseudocode

2.3.62. EditPage.addData()

```
EditPage:addData(dataType)
IF dataType = profile
 EditProfile:addData
 IF dataType = jobList
 JobList:addPost
```

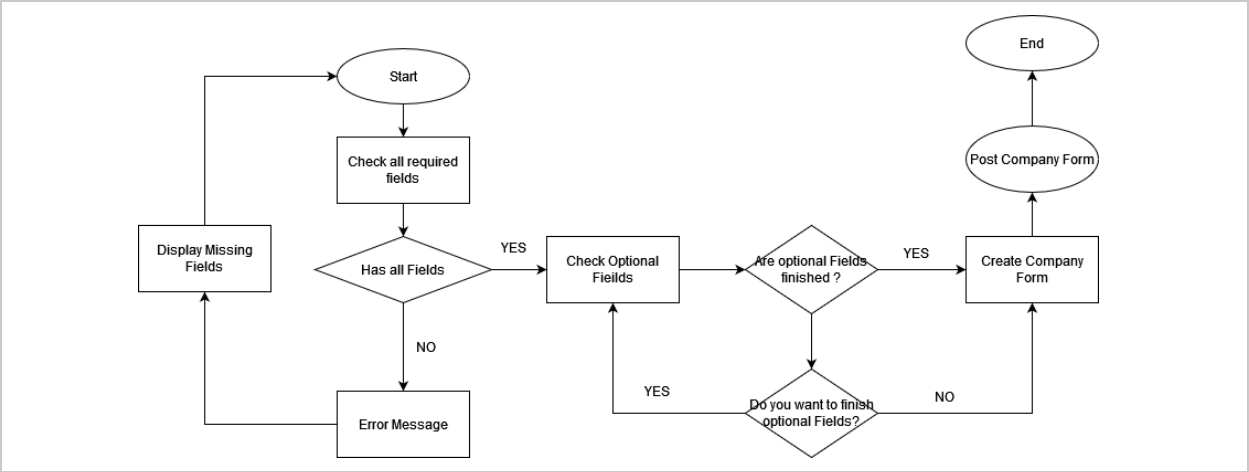
Name	**EditPage.addData()**
Purpose	Page used to add new data.
Description	This page will be given what will be type of element to be added then create the page
Requirements	2.1.3.2, 2.1.3.10, 3.1.2
Inputs	Type of Data to add.
Outputs	Retrieves appropriate page information
Elements	**dataType**: The data type either profile data or job list data
Referenced By	1.3.2, 1.3.5, 1.3.8, 1.3.9, 2.2.1, 2.2.2, 2.2.3, 2.2.4
Viewpoint	Pseudocode

2.3.63. EditPage.editData()

```
EditPage:editData(id)
DataType <-- getDataType(id)
 IF dataType = profile
 EditProfile:editData
 IF dataType = jobList
JobList:editPost
```

Name	**EditPage.editData()**
Purpose	Page used to edit data.
Description	This page will use an ID to create the page to edit already existing data.
Requirements	2.1.3, 2.1.3.10, 3.1.2
Inputs	Data ID
Outputs	Page for editing Data
Elements	**dataType:** The data type either profile data or job list data
	id: provided id for data to be entered
Referenced By	1.3.2, 1.3.5, 1.3.8, 1.3.9, 2.2.1, 2.2.2, 2.2.3, 2.2.4
Viewpoint	Pseudocode

2.3.64. CompanyForm.submit()



Name	**CompanyForm.submit()**
Purpose	Allows a company to submit a Company Form
Description	Checks all the required and optional fields of a company form, if all required fields are filled allow the posting of the form
Requirements	2
Inputs	Input Fields
Outputs	Company Form
Elements	**Required Fields:** Fields necessary to allow searching for this post, usually including post ID, Author, time, c.
	Optional Fields: Fields that would be used to be more accurate for the search functions, some examples might be Class, features, average pay … etc.
Referenced By	2.2.15
Viewpoint	Flowchart

2.3.65. Admin.editJobPostings()

```
editJobPostings()
SET user <- Admin
onclick(job_posting_data)
onclick -> Choice

IF (Choice == Edit)
GET -> new_job_posting_data
Send(job_posting_data, Choice, new_job_posting_data)
ELSEIF (Choice == Remove)
Send (job_posting_data,Choice)
onClick(AddNew)
GET -> new_job_posting_data
Send(Choice, new_job_posting_data)
```

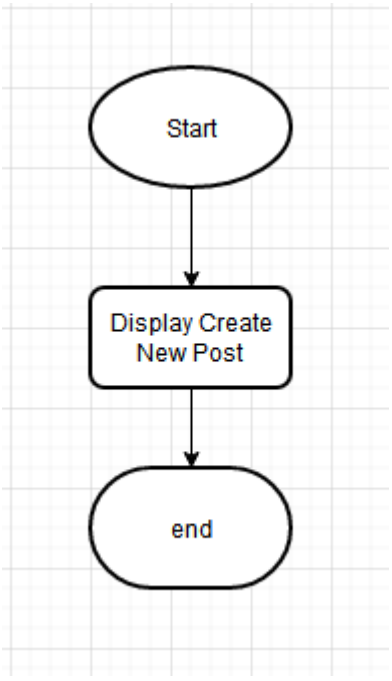
Name	Admin.editJobPostings()
Purpose	Administrator access to update, alter or remove job postings
Description	Administrator access is required to change and edit the job postings content and format.
Requirements	2.5.1
Inputs	The job post - company name, job title, required skills, job description, hours, position type, compensation, location, contact information, and links to apply for the job.
Outputs	Changes in the post
Elements	**User:** The level of access of the user
	OnClick(): User interaction
	Choice: Type of information's to change (company name, job title, required skills, job description, hours, position type, compensation, location, contact information, and links to apply for the job)
	job_posting_data: Data to be changed
	new_job_posting_data: Changed data.
Referenced By	3.3.1
Viewpoint	Pseudocode

2.3.66. Moderator.editJobPostings()

```
EditJobPosting(id)
JobPostData <- getJobPostData(id)
JobPost:editJobPost(JobPostData)
```

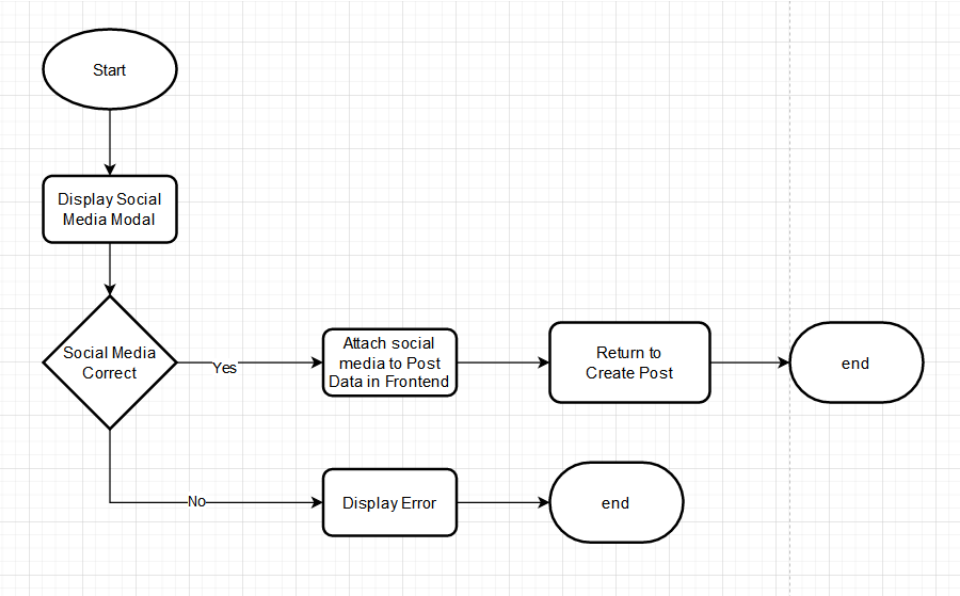
Name	**Moderator.editJobPostions()**
Purpose	Page will provide a way for editing job postings
Description	This page provides a way for a moderator to edit a job posting.
Requirements	2.5, 2.5.1,
Inputs	Job Post ID
Outputs	Page for modifying job posting
Elements	**JobPostData:** Retrieved Job posting data
	Id: ID for the job posting
Referenced By	1.3.3, 1.3.5
Viewpoint	Pseudocode

2.3.67. AddPosts.onClickNewPost()



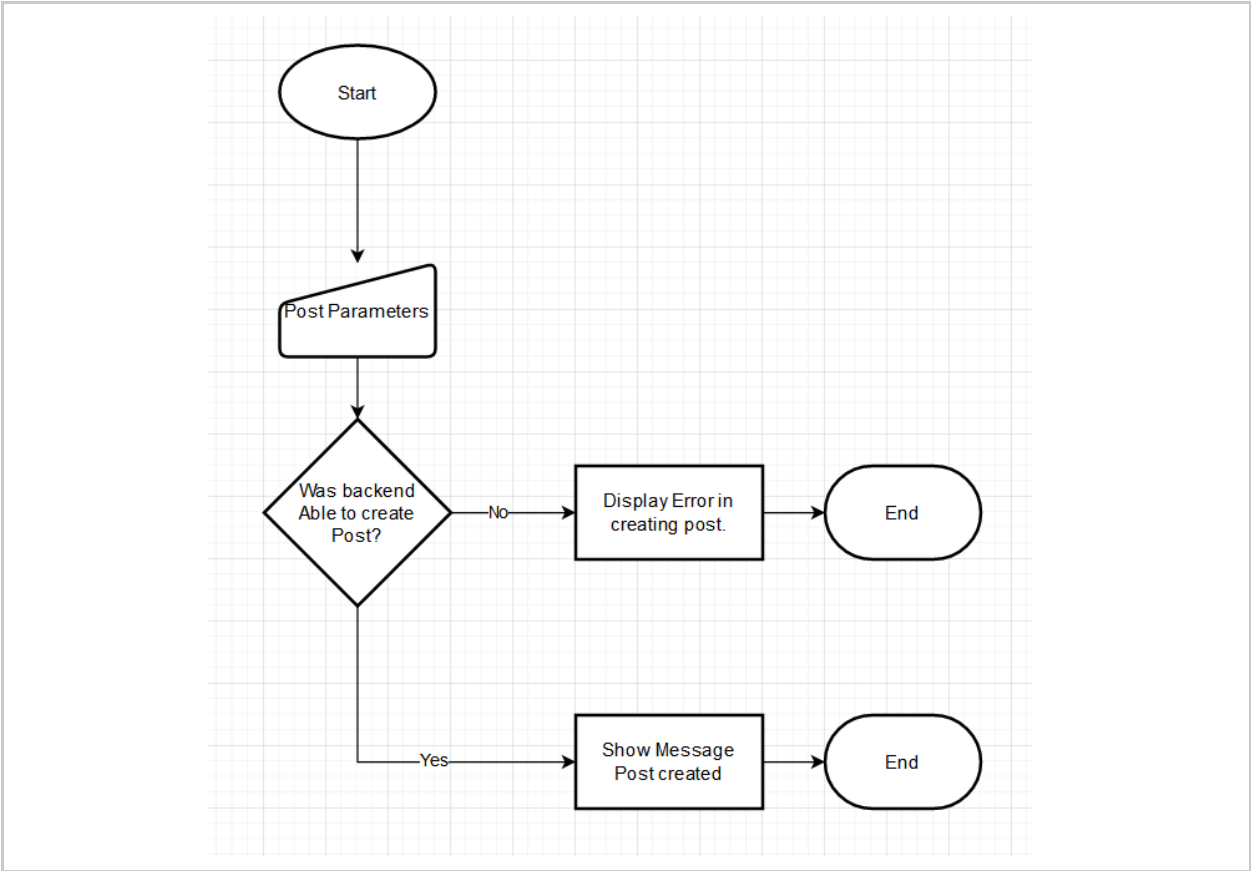
Name	**AddPosts.onClickNewPost()**
Purpose	To show the create new post page
Description	The function that detects when a user wants to create a new post
Requirements	2.4, 2.4.1
Inputs	The user click on button
Outputs	None
Elements	Display Create New Post Page
Referenced By	2.3.1
Viewpoint	Flowchart

2.3.68. AddPosts.onClickSocialMedia()



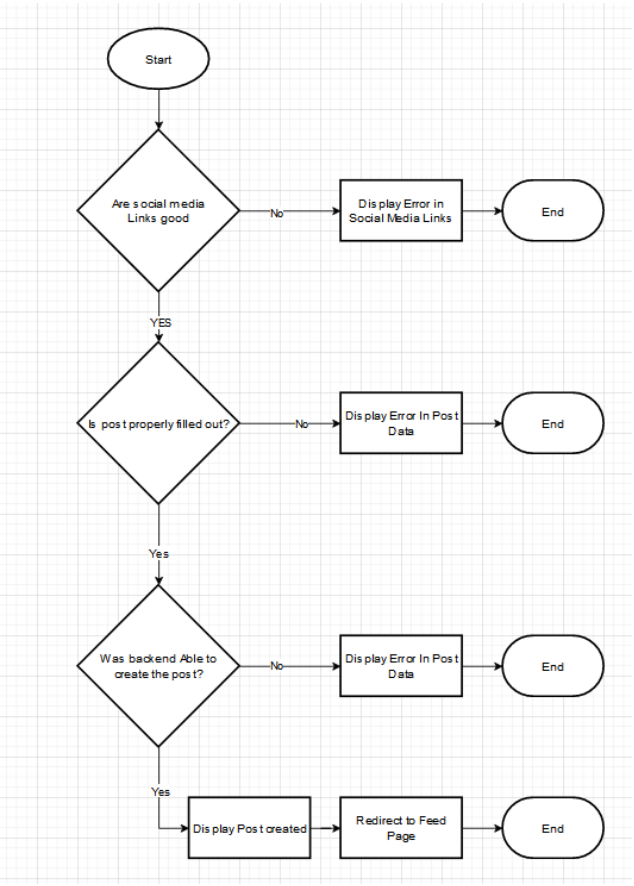
Name	**AddPosts.onClickSocialMedia()**
Purpose	Adds social media to a post
Description	Allows a company to create a post that has social media connection links added to it
Requirements	2.4, 2.4.6
Inputs	Post object, Social Media Links
Outputs	Adds links to post object
Elements	Social Media Modal
	Links
Referenced By	1.1.1, 2.3.28
Viewpoint	Flowchart

2.3.69. AddPosts.submitNewPost()



Name	AddPosts.submitNewPost()
Purpose	Create a post for the currently signed in company
Description	When a company wants to create a post, they will need to use this function to create it
Requirements	2.4 , 2.4.1
Inputs	Post Title, Post Body, Social Media link, Attachments
Outputs	Returns Error if could not create, or Success if the post was created
Elements	**Post Parameters:** Gathered from user input
	Backend Call: Backend will use parameters and create a new post for a company
	Display: Error or Post created
Referenced By	2.3.1
Viewpoint	Flowchart

2.3.70. AddPosts.submitSocialMedia()



Name	**AddPosts.submitSocialMedia()**
Purpose	Create a social media Post
Description	Create a post with social media links
Requirements	1.1.1, 2.4.6
Inputs	Social media links, Post form Data
Outputs	Newly created Post
Elements	Social Media Check, Display Errors, Post data Check, Backend Creation Check, Display Success Message, Redirect Page
Referenced By	2.3.28
Viewpoint	Flowchart

2.3.71. JobList.addPost()

JobList.addPost()
```
JobPostDetails <- GET JobPostDetails
```
Submit jobPostDetails

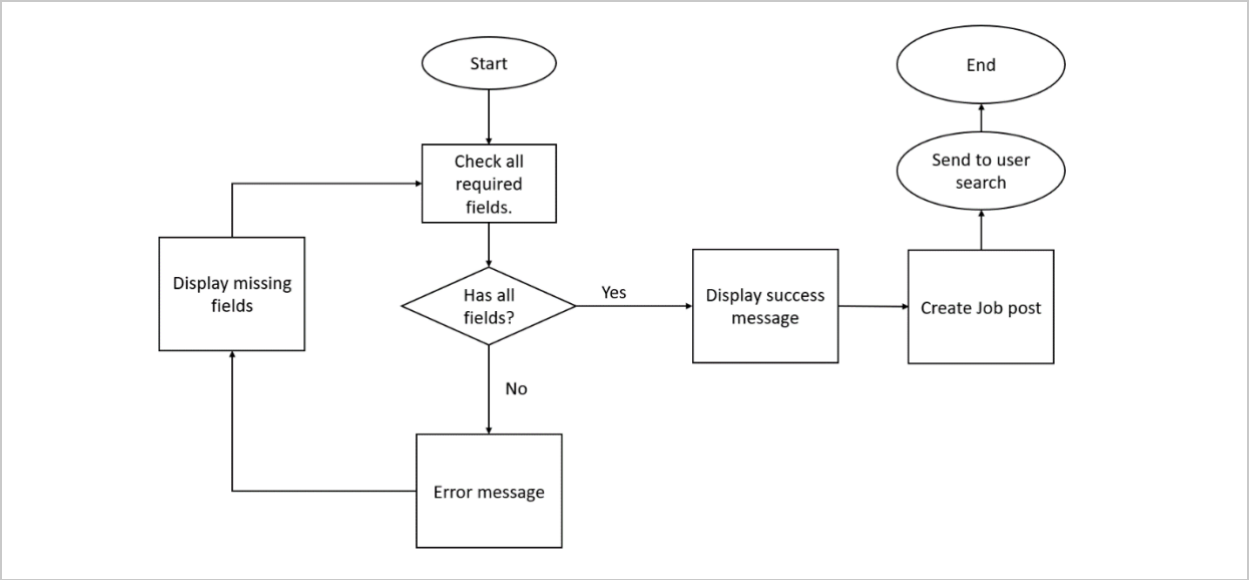
Name	**JobList.AddPost()**
Purpose	Method for creating a job post.
Description	This method will add a job post.
Requirements	2.5, 2.5.1, 1.3.4, 1.3.5
Inputs	Job post details
Outputs	Job post
Elements	**Job post details:** this information will contain all the data the user will submit.
Referenced By	2.3.67 - 2.3.70
Viewpoint	Pseudocode

2.3.72. JobList.viewApplicants()

```
JobList::ViewApplicants()
    For applicants in jobPost.getApplicants()
        For applicant in applicants
            DISPLAY applicant
```

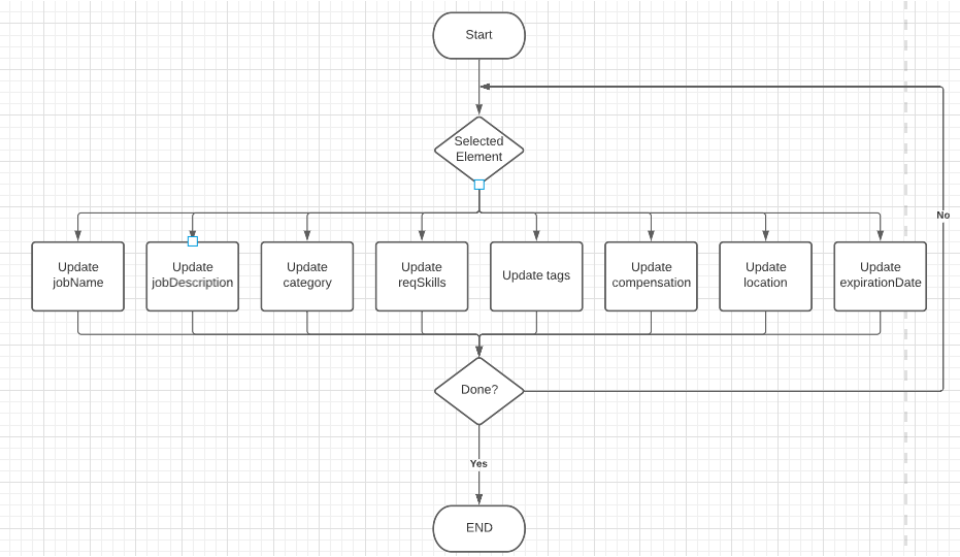
Name	**JobList.viewApplicants**
Purpose	Display the applicants for a list of jobs.
Description	Loops through the applicants and displays them.
Requirements	2.1.3.1. , 2.5 , 2.5.1
Inputs	None
Outputs	Applicants output via display.
Elements	**JobPost:** Backend class with information on job posts.
	Display: Displays data on the screen.
Referenced By	2.3.52
Viewpoint	Pseudocode

2.3.73. JobForm.submitJobForm()



Name	**JobForm.submitJobForm()**
Purpose	Submit the job form information to create the job post.
Description	Validate all required fields and submit the information to create the job post.
Requirements	2.5.2, 2.5.7 , 2.3.3, 1.3.1
Inputs	Company name, job title, required skills, job description, hours, position type, compensation, location, contact information, and links to apply for the job
Outputs	Job Post
Elements	**Error handlining:** Will check all required fields before sending to JobPostings()
	Create job post: Will send the form information to JobPostings()
Referenced By	2.3.15-2.3.17
Viewpoint	Flowchart

2.3.74. JobPost.editPost()



Name	**JobPost.editPost()**
Purpose	Describes how this function will make the decision to send the change notification.
Description	Checks to see if a post has been changed, if it has then it will send a notification to all students who are following that company to inform them of the change.
Requirements	2.5.4.
Inputs	Incoming changed to the post
Outputs	Notifications to the users
Elements	jobName
	jobDescription
	category
	reqSkills
	tags
	compensation
	location
	expirationDate
Referenced By	2.2.29
Viewpoint	Flow Chart

2.3.75. JobPost.deletePost()

```
JobPost::deletePost()
    backendJobPost.deletePost()
```

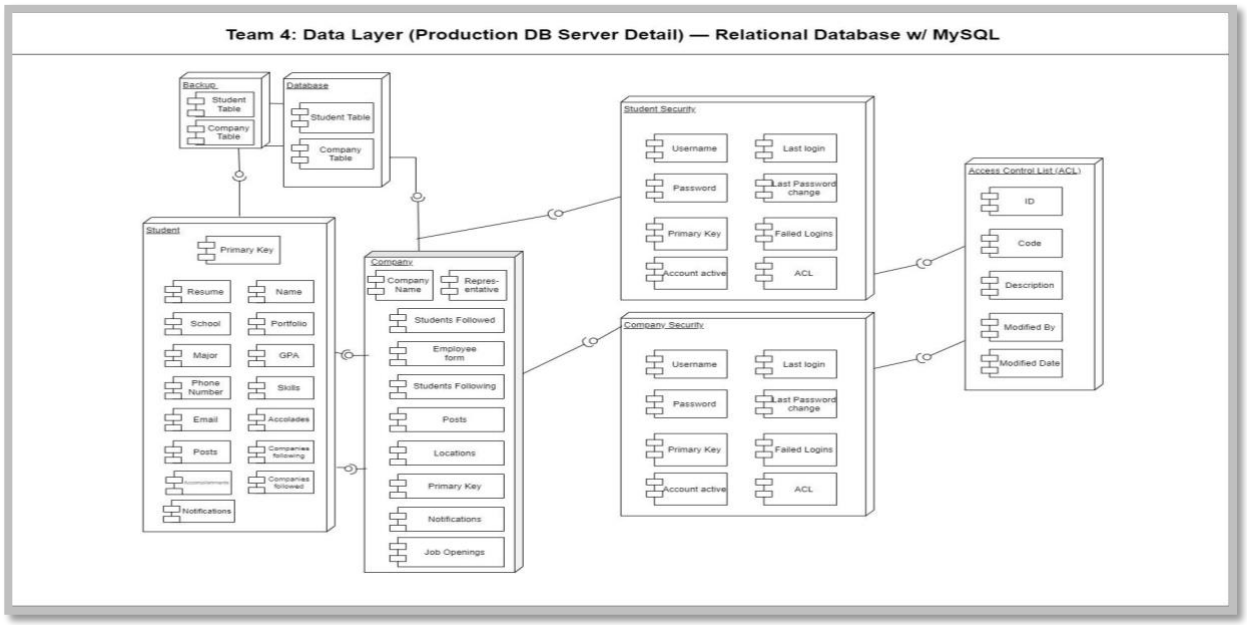
Name	**JobPost.deletePost()**
Purpose	Delete a job post.
Description	Deletes the job post by calling the backend function to delete the backend.
Requirements	2.5.1
Inputs	None
Outputs	None
Elements	**Job Post (Front End):** The front-end class for job posts.
	Job Post (Back End): Gets data from the database to create the front-end job post.
	DeletePost – Deletes the backend version of JobPost.
Referenced By	2.3.15, 2.3.16, 2.3.17
Viewpoint	Pseudocode

2.3.76. JobPost.getJobInfo()

```
getJobInfo()
  SET jobData ⇐ backendJobPost.getJobInfo(jobPosting)
  PUT jobData to screen
```

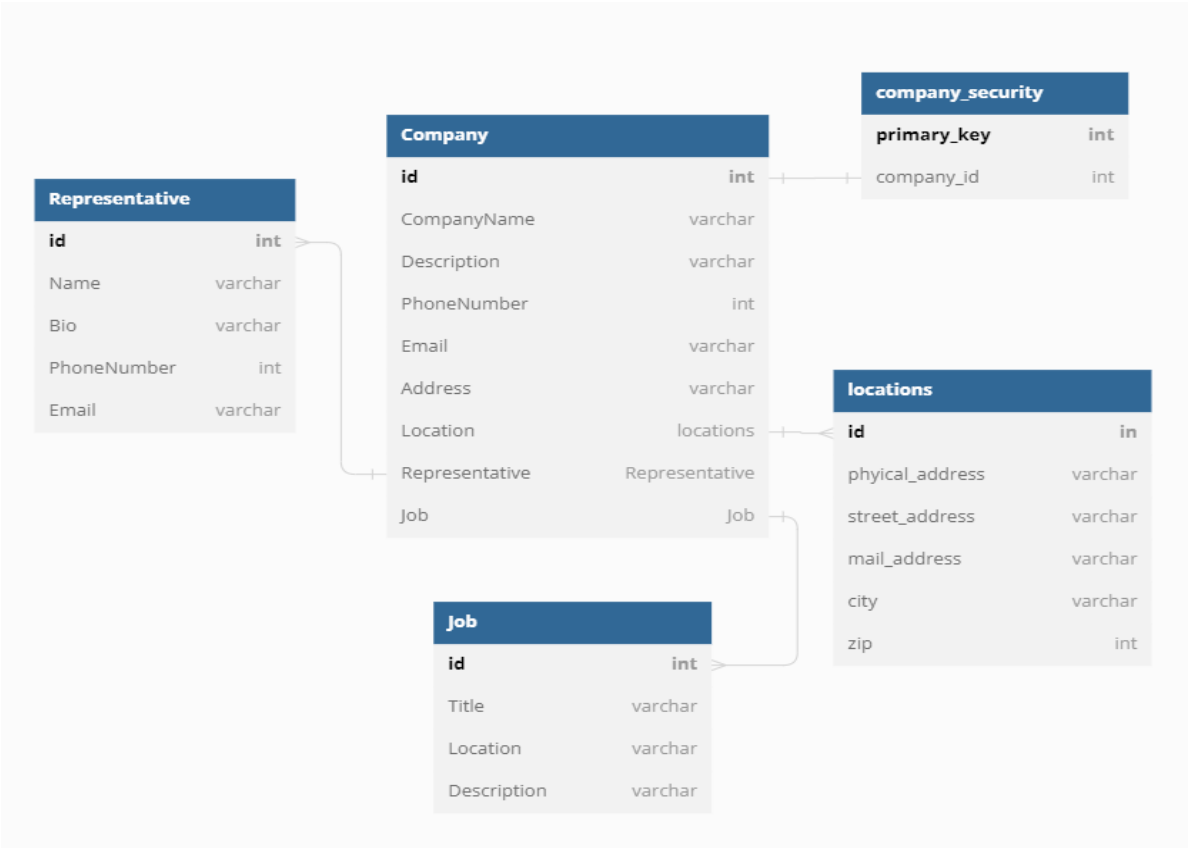
Name	**JobPost.getJobInfo()**
Purpose	Display information
Description	Calls the backend getJobInfo and then displays the data to the screen
Requirements	2.5.1, 2.5.9
Inputs	**None**
Outputs	**None**
Elements	**JobData:** all info related to a job posting
	JobPosting: the id of the requested job info
Referenced By	2.3.33
Viewpoint	Pseudocode

2.4. Data Tier - Database Overview



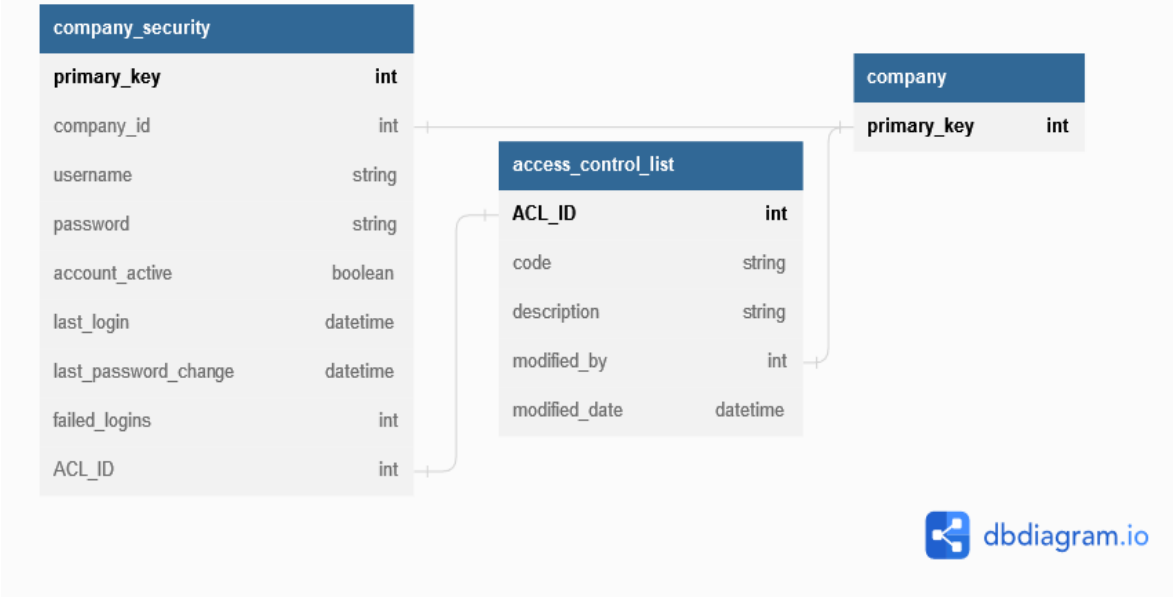
Name	**Overall Data Tier System Diagram**
Purpose	Describe the overall database architecture.
Description	The diagram depicts the two main tables used to hold the student and company models and their dependencies on their security table governed by ACL rules.
Requirements	2-3
Elements	**Company Table:** A listing of the various data points required by company profiles.
	Student Table: A listing of the various data points required by student profiles.
	Company Security Table: A list of credentials needed to initiate an active sign-in session for a company representative.
	Student Security Table: A list of required credentials to initiate an active sign-in session for a student.
	ACL: A list of rules denoting which clients are allowed certain privileges.
Referenced By	1-3
Viewpoint	Component Diagram

2.4.1. Company Entity Relation Diagram



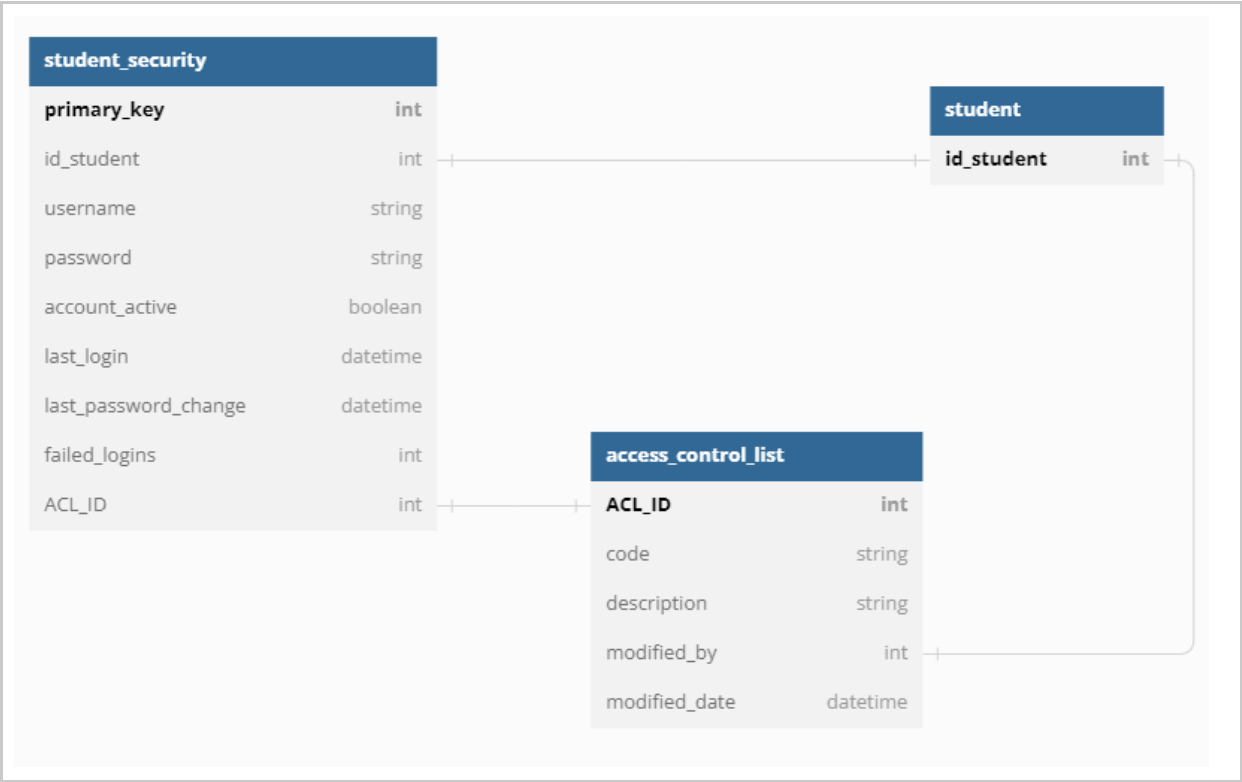
Name	**Company Entity Relation Diagram**
Purpose	Describes the relations between the company table and other entities.
Description	The diagram displays the company table's relations with representatives, locations, job postings, and company security tables.
Requirements	2.1.3
Elements	**Company Table:** Database table with information about the company.
	Locations Table: Database Table with information about the locations of the company.
	Representative Table: Database table with information about the representative of the company and their contact information.
	Job Table: Database table containing relevant information about a single job posting.
	Company Security Table: Database table that contains information about what a company can access within the system (more details are in the next view).
Referenced By	2.4
Viewpoint	Entity Relationship Diagram (ERD)

2.4.2. Company Security Entity Relationship Diagram



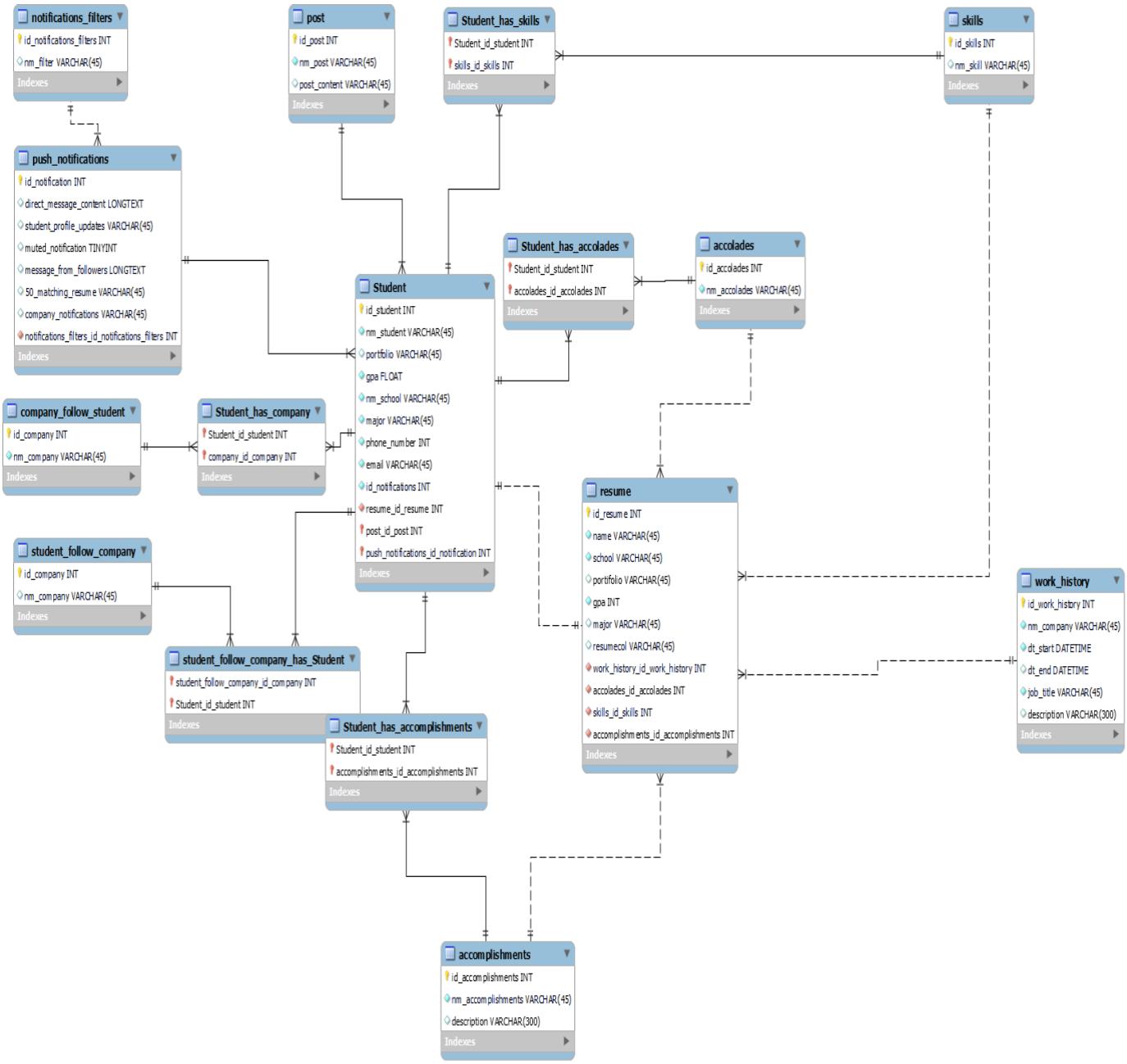
Name	**Company Security**
Purpose	Show relationships between the Company Security table and other tables in the database.
Description	The company_security table connects to the access_control_list via the ACL_ID attribute, with the foreign key "company_id" being derived from the company table's primary key. All relationships are one-to-one.
Requirements	2.1.3.3, 2.1.3.10, 2.4.3, 2.5.1
Elements	**Company Security Table:** Database table containing information about a company's account and authorization for that account.
	Access Control List Table: Database table for the sets of permissions attached to company profile privileges whose availability is administered to the appropriate company representative once satisfied. **Code:** a string denoting what permissions an account has.
	Company table: Database table containing one single attribute for the company profile's id.
Referenced By	2.4
Viewpoint	Entity Relationship Diagram (ERD)

2.4.3. Student Security Relationship Diagram



Name	**Student Security**
Purpose	Show the relationship between the Student Security table and other tables in the database.
Description	The student_security table connects to the access_control_list via the ACL_ID attribute, with the foreign key “id_student” derived from the student table’s primary key. All relationships are one-to-one.
Requirements	3.1.3 - 3.1.3.4
Elements	**Student Security Table:** Database table containing information about a student’s account and authorization for that account.
	Access Control List (ACL) Table: Database table for the sets of permissions attached to student profile privileges whose availability is administered to the student once satisfied.
	Student Table: A database table containing one attribute for the student profile’s ID.
Referenced By	2.4
Viewpoint	Entity Relationship Diagram (ERD)

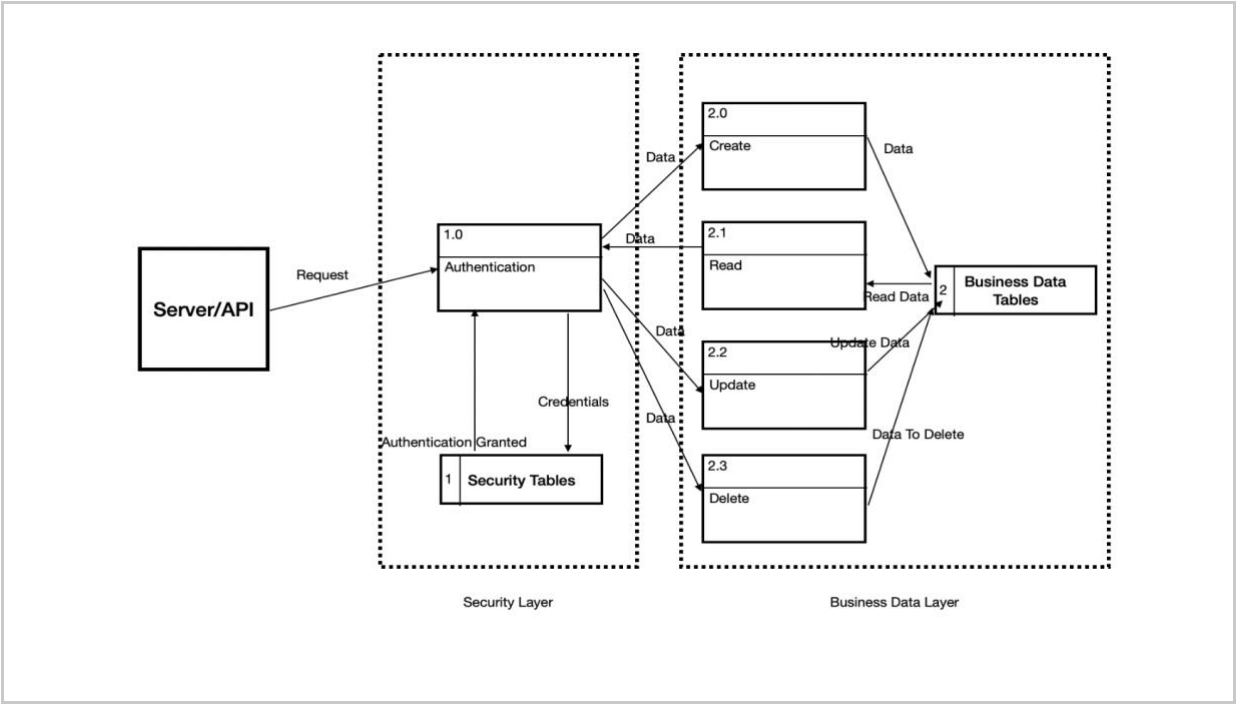
2.4.4. Student table review diagram.



Name	**Student Table**
Purpose	Store all student-related information for the student user.
Description	The student table is required to store the related information regarding points of contact, accomplishments, and experience as shown on student user profile page. These data points are subject to filters in candidate searches performed by a company user representative.
Requirements	3.1.1 - 3.2.7, 3.4.2 - 3.5.3
Elements	**Notifications_filters:** Allow filtering of different notification types based on user's preference with a one-to-many relationship to the push_notification.
	push_notifications: A table to hold the different types of notifications, including direct messages, student profile updates for companies, and messages from followers.
	***company_follow_student:** A table to hold the different companies following a student's profile.
	***Student_follow_company:** A table to hold the different students a company profile is following.
	***Accomplishments:** A table to hold the different accomplishments of a student.
	Resume, work_history: Including a separate table, "work_history," a table to hold data from resume forms, including contact information, user work experience, degree information, and skills.
	***student_has_accolades**: A table to hold students' different accolades (awards).
	Posts: A list of posts that a student makes.
	***Skills, student_has_skills:** A table to hold students' different skills.
Referenced By	2.4
Viewpoint	Entity Relationship Diagram (ERD)

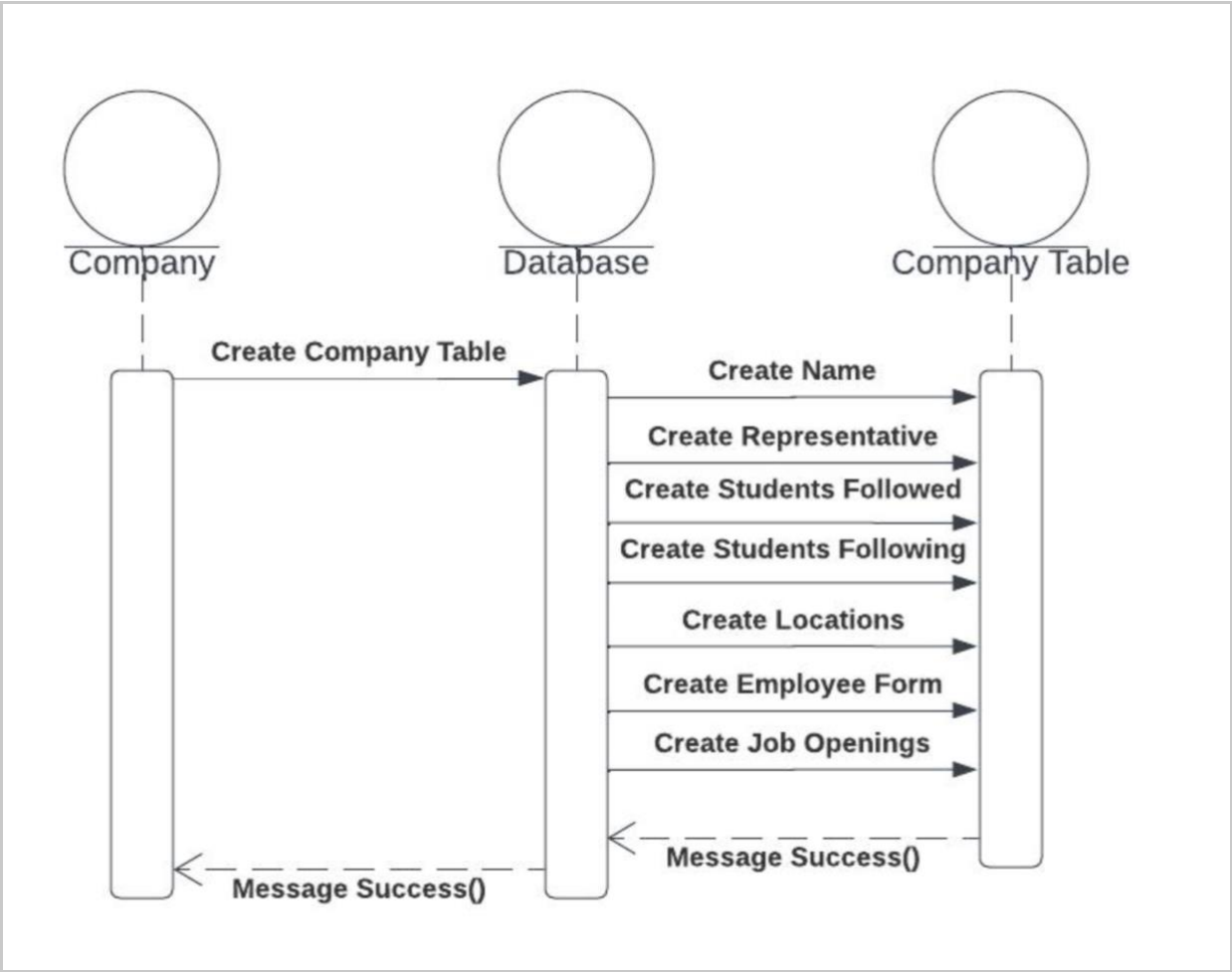
*Bridge tables are included to properly connect the many-to-many relationships (i.e., one skill can have many students, and many students can have one skill).

2.4.5. Data Flow Diagram for Interacting with the Database



Name	**Server interaction with Database DFD**
Purpose	Illustrate the flow of CRUD operations between server-side API requests and the Company Profile Page Data tables in relation to authentication.
Description	The server makes a request authenticated via the security tables to read, update, and delete the requested data.
Requirements	2.1.3.10, 2.4.1, 3.4.2
Elements	**Server/API**: The source of all requests to the database as initiated by client interactions with the UI.
	Security Layer: For data confidentiality and integrity, all communication to and from the database is required to pass through here to be verified by the security tables.
	Authentication: Ensures that all requests are fulfilled with valid credentials.
	Security Tables: Database for storing credentials and other security data.
	Create, Read, Update, Delete: The main task of the database API is to modify data in the database.
	Company Data Tables: Stores all of the company data, such as student and employer information.
	Company Data Layer: Where all the company-related tasks' changes are performed and reflected.
Referenced By	2.4
Viewpoint	Data Flow Diagram (DFD)

2.4.6. Sequence Diagram: Perspective of Company Representative



Name	**Company interaction Sequential Diagram**
Purpose	Illustrate the timely sequence of creating tables in the database.
Description	A company representative interacts with the UI to create their company profile. After the initial profile is created, further data can be added, such as company locations and a job interest form.
Requirements	2.1.3.1, 2.1.3.3, 2.1.3.10
Elements	**Company**: The client initializes their company's profile as its representative.
	Database: The structure of data meant to hold data models used in the application.
	Company Table: The table to preserve the additions and modifications made by the client to their company profile.
Referenced By	2.4
Viewpoint	Sequence Diagram

Appendices - Back Matter

2.5. Traceability Matrix

SRS Sections	1.1.1	1.2.1	1.3.1	1.3.2	2.1.1	2.1.2	2.1.3.1	2.1.3.2	2.1.3.3	2.1.3.4	2.1.3.5	2.1.3.6	2.1.3.7	2.1.3.8	2.1.3.9	2.1.3.10	2.2.1	2.2.2	2.2.3	2.2.4	2.2.5	2.3.1	2.3.2	2.3.3	2.3.4	2.3.5	2.3.6	2.3.7	2.3.8	2.4.1	2.4.2	2.4.3	2.4.4	2.4.5	2.4.6	2.5.1	2.5.2	2.5.3	2.5.4	2.5.5	2.5.6	2.5.7	2.5.8	2.5.9	3.1.1	3.1.2	3.1.3.1	3.1.3.2	3.1.3.3	3.1.3.4	3.2.1	3.2.2	3.2.3	3.2.4	3.2.5	3.2.6	3.2.7	3.3.1	3.3.2	3.3.3	3.3.4	3.4.1	3.4.2	3.5.1	3.5.2	3.5.3	3.5.4.1	3.5.4.2	3.5.5.1	3.5.5.2	3.5.5.3	4.1	4.2	4.3	4.4
2.0 System View	X																																																																										
2.1 DevOps Overview	X																																																																								X	X	X	
2.1.1 DevOps DFD	X																																																																										
2.1.2 Server Cluster	X																																																																										
2.1.3 Sequence Diagram	X																																																																								X	X	X	
2.2 Backend Overview					X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
2.2.1 Backend DFD																																												X	X	X	X	X	X	X								X	X	X	X	X	X												
2.2.2 Authentication Flowchart									X							X																														X																													
2.2.3 Web Server DFD																																																																										
2.2.4 Companies Profile Page: UML Class Diagram					X	X	X	X	X	X	X	X	X	X	X	X	X																																																											
2.2.5 Companies Notifications: UML Class Diagram																	X		X	X	X																																X																							
2.2.6 Companies Search: UML Class Diagram																		X				X		X	X				X																																															
2.2.7 Companies News Feed: UML Class Diagram																														X	X	X	X	X	X																																								
2.2.8 Companies Job Postings: UML Class Diagram																																				X	X	X	X	X	X	X																																			
2.2.9 Students Profile Page: UML Class Diagram																																													X	X	X	X	X	X																									
2.2.10 Students Notifications: UML Class Diagram																																																			X	X	X	X	X	X	X																		
2.2.11 Students Search: UML Class Diagram																																																										X		X	X															
2.2.12 Students News Feed: UML Class Diagram																														X	X	X	X	X	X																																								
2.2.13 Students Resume: UML Class Diagram																																																																										
2.2.14 All Backend UML Class Diagrams					X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
2.2.15 REST API Endpoints					X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
2.2.16 Profile Update Notification Flowchart																				X																															X	X	X		X																				
2.2.17 Profile View Notification Flowchart																																																							X																				
2.2.18 Company Notification to Followers Flowchart																	|X|																																																									
2.2.19 Company Notification when Post Created Flowchart																		X																																	X	X					X																		
2.2.20 Resume Match Company Notification																	X	X	X	X	X						X	X	X																																															
2.2.21. CompanyNewsFeed.createPost()																													|X|																																														
2.2.22. CompanyNewsFeed.editPost()																													|X|																																														
2.2.23. StudentNewsFeed.createPost()																																																															X												
2.2.24. NotificationListener.fetchProfiles()																	X					X																																																					
2.2.25. NotificationListener.fetchUsername()																			X																																																								
2.2.26. NotificationListener.fetchResumes()																					X																																																						
2.2.27. INotification.send()																		X																																																									
2.2.28. JobPost.editPost()																																				X								X																															
2.2.29. JobPost.deletePost()																																			|X|																																							
2.2.30. JobPost.getJobInfo()																																			|X|																																							
2.2.31. Application.apply()					|X|																																																																					
2.2.32. Application.withdrawApplication()					|X|																																																																					
2.2.33. Application.deleteApplication()					|X|																																																																					
2.2.34. Application.saveApplication()					|X|																																																																					
2.2.35. CompanyRepresentative.editPage()																X																																																											
2.2.36. LoginPage.login()									X							X																																						X																				
2.2.37. LoginPage.forgotPassword()									X							X																																						X																				
2.2.38. Student.updateInfo()																																														X																													
2.2.39. Resume.addResume()																																																																X	X										
2.2.40. Resume.updateResume()																																																																X	X										
2.2.41. Education.addEducation()																																														X																													
2.2.42. Education.deleteEducation()																																														X																													
2.2.43. Education.updateEducation()																																														X																													
2.2.44. JobExperience.addExperience()																																														X																													
2.2.45. JobExperience.deleteExperience()																																														X																													
2.2.46. JobExperience.updateExperience()																																														X																													
2.2.47. MandatoryFields.addMandatoryField()																																																																						X	X				
2.2.48. HelpfulHint.addHint()																																																																				X							
2.2.49. CompanySearch.filterBy()																							X		X	X			X																																														
2.2.50. CompanySearch.searchResumes()																												|X|																																														
2.2.51. CompanySearch.searchProfiles()																						X	X	X	X	X	X	X	X																																															
2.2.52. CompanySearch.wildcardSearch()																											|X|												|X|																																			
2.2.53. CompanySearch.displayResults()																						X	X	X	X	X	X	X	X																																															
2.2.54. SearchBar.autocomplete()																						X	X	X	X	X	X	X	X																																															
2.2.55. SearchBar.displaySearchHistory()																							X	X																																																				
2.2.56. UserSearch.filterBy()												X																																																X	X														
2.2.57. UserSearch.search()												X																																																X	X														

SRS Sections	1.1.1	1.2.1	1.3.1	1.3.2	2.1.1	2.1.2	2.1.3.1	2.1.3.2	2.1.3.3	2.1.3.4	2.1.3.5	2.1.3.6	2.1.3.7	2.1.3.8	2.1.3.9	2.1.3.10	2.2.1	2.2.2	2.2.3	2.2.4	2.2.5
2.2.58. UserSearch.displayResults()																					
2.2.59. UserSearch.searchJobs()																					
2.2.60. UserSearch.searchCompanies()																					
2.2.61. AuthenticationList.isAuthorized()									X							X					
2.2.62. EditPage.addData()									X												
2.2.63. EditPage.editData()									X												
2.2.64. EditPage.RemoveData()									X												
2.3 Frontend Overview					X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
2.3.1 Frontend Class Diagram					X	X															
2.3.2 Frontend Structure							X	X	X	X	X	X	X	X	X	X					
2.3.3. Companies Profile Page: UI Components					X	X	X	X	X	X	X	X	X	X	X	X					
2.3.4. Companies Profile Page: UML Class Diagram					X	X	X	X	X	X	X	X	X	X	X	X					
2.3.5. Companies Profile Page: Sequence Diagram					X	X	X	X	X	X	X	X	X	X	X	X					
2.3.6. Companies Notifications: UI Components																	X	X	X	X	X
2.3.7. Companies Notifications: UML Class Diagram																	X	X	X	X	X
2.3.8. Companies Notifications: Sequence Diagram																	X	X	X	X	
2.3.9. Companies Search: UI Components																					
2.3.10. Companies Search: UML Class Diagram																					
2.3.11. Companies Search: Sequence Diagram																					
2.3.12. Companies News Feed: UI Components																					
2.3.13. Companies News Feed: UML Class Diagram																					
2.3.14. Companies News Feed: Sequence Diagram																					
2.3.15. Companies Job Postings: UI Components																					
2.3.16. Companies Job Postings: UML Class Diagram																					
2.3.17. Companies Job Postings: Sequence Diagram																					
2.3.18. Students Profile Page: UI Components																					
2.3.19. Students Profile Page: UML Class Diagram																					
2.3.20. Students Profile Page: Sequence Diagram																					
2.3.21. Students Notifications: UI Components																					
2.3.22. Students Notifications: UML Class Diagram																					
2.3.23. Students Notifications: Sequence Diagram																					
2.3.24. Students Search: UI Components												X									
2.3.25. Students Search: UML Class Diagram																					
2.3.26. Students Search: Sequence Diagram																					
2.3.27. Students News Feed: UI Components																					
2.3.28. Students News Feed: UML Class Diagram																					
2.3.29. Students News Feed: Sequence Diagram																					
2.3.30. Students Resume: UI Components																					
2.3.31. Students Resume: UML Class Diagram																					
2.3.32. Students Resume: Sequence Diagram			X																		
2.3.33 All Frontend UML Class Diagrams					X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
2.3.34. Company.viewInfo()							X														
2.3.35. SubmitInfo.submitForm()																					
2.3.36. StudentPage.submitResume()																					
2.3.37. ResumeWindow.openResumeForm()																					
2.3.38. ResumeForm.submitResumeForm()																					
2.3.39. ResumeForm.getHelpDetails()																					
2.3.40. Search.filter()																					
2.3.41. Search.search()																					
2.3.42. Filter.filter()																					
2.3.43. Autocomplete.autocomplete()																					
2.3.44. CompanyRepresentative.viewInfo()														X							
2.3.45. CompanyRepresentative.editPage()																X					
2.3.46. Student.fillForm()																					
2.3.47. EditProfile.addData()							X	X	X	X	X	X	X	X	X	X					
2.3.48. EditProfile.editData()																					
2.3.49. EditProfile.removeData()																					
2.3.50. StudentMessage.filter()																					
2.3.51. CompanyMessagesInterface.composeMessage()																		X			
2.3.52. CompanyMessagesInterface.studentMessage()																		X			
2.3.53. CompanyMessagesInterface. ResumeMatchMessage()																					X
2.3.54. CompanyMessagesInterface.studentUpdateMessage()																				X	
2.3.55. ComposeMessageInterface.sendMessage()																					
2.3.56. StudentMessagesInterface.studentMessage()																					
2.3.57. NewsFeedList.displayNewsFeedList()																					
2.3.58. NewsFeedList.update()																					
2.3.59. Post.send()																					
2.3.60. Post.getHtml()																					
2.3.61. EventPost.getHtml()																					

SRS Sections	2.3.1	2.3.2	2.3.3	2.3.4	2.3.5	2.3.6	2.3.7	2.3.8	2.4.1	2.4.2	2.4.3	2.4.4	2.4.5	2.4.6	2.5.1	2.5.2	2.5.3	2.5.4	2.5.5	2.5.6	2.5.7	2.5.8	2.5.9
2.2.58. UserSearch.displayResults()																							
2.2.59. UserSearch.searchJobs()																							
2.2.60. UserSearch.searchCompanies()																							
2.2.61. AuthenticationList.isAuthorized()																							
2.2.62. EditPage.addData()																							
2.2.63. EditPage.editData()																							
2.2.64. EditPage.RemoveData()																							
2.3 Frontend Overview	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
2.3.1 Frontend Class Diagram																							
2.3.2 Frontend Structure																							
2.3.3. Companies Profile Page: UI Components																							
2.3.4. Companies Profile Page: UML Class Diagram																							
2.3.5. Companies Profile Page: Sequence Diagram																							
2.3.6. Companies Notifications: UI Components																							
2.3.7. Companies Notifications: UML Class Diagram																							
2.3.8. Companies Notifications: Sequence Diagram																							
2.3.9. Companies Search: UI Components	X	X	X	X	X	X	X	X															
2.3.10. Companies Search: UML Class Diagram	X	X	X	X	X	X	X	X															
2.3.11. Companies Search: Sequence Diagram	X	X	X	X	X	X	X	X															
2.3.12. Companies News Feed: UI Components									X		X												
2.3.13. Companies News Feed: UML Class Diagram											X												
2.3.14. Companies News Feed: Sequence Diagram									X	X	X	X	X	X									
2.3.15. Companies Job Postings: UI Components															X	X	X	X	X	X	X	X	X
2.3.16. Companies Job Postings: UML Class Diagram															X	X	X	X	X				
2.3.17. Companies Job Postings: Sequence Diagram															X	X		X					
2.3.18. Students Profile Page: UI Components																							
2.3.19. Students Profile Page: UML Class Diagram																							
2.3.20. Students Profile Page: Sequence Diagram																							
2.3.21. Students Notifications: UI Components																							
2.3.22. Students Notifications: UML Class Diagram																							
2.3.23. Students Notifications: Sequence Diagram																							
2.3.24. Students Search: UI Components																							
2.3.25. Students Search: UML Class Diagram																							
2.3.26. Students Search: Sequence Diagram																							
2.3.27. Students News Feed: UI Components																							
2.3.28. Students News Feed: UML Class Diagram																							
2.3.29. Students News Feed: Sequence Diagram																							
2.3.30. Students Resume: UI Components																							
2.3.31. Students Resume: UML Class Diagram																							
2.3.32. Students Resume: Sequence Diagram												X											
2.3.33 All Frontend UML Class Diagrams	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
2.3.34. Company.viewInfo()																							
2.3.35. SubmitInfo.submitForm()																							
2.3.36. StudentPage.submitResume()																							
2.3.37. ResumeWindow.openResumeForm()																							
2.3.38. ResumeForm.submitResumeForm()																							
2.3.39. ResumeForm.getHelpDetails()																							
2.3.40. Search.filter()		X	X	X	X			X														X	
2.3.41. Search.search()	X	X	X	X	X	X	X	X															
2.3.42. Filter.filter()	X	X	X																				
2.3.43. Autocomplete.autocomplete()						X																	
2.3.44. CompanyRepresentative.viewInfo()																							
2.3.45. CompanyRepresentative.editPage()																							
2.3.46. Student.fillForm()																							
2.3.47. EditProfile.addData()																							
2.3.48. EditProfile.editData()																							
2.3.49. EditProfile.removeData()																							
2.3.50. StudentMessage.filter()																							
2.3.51. CompanyMessagesInterface.composeMessage()																							
2.3.52. CompanyMessagesInterface.studentMessage()																							
2.3.53. CompanyMessagesInterface. ResumeMatchMessage()																							
2.3.54. CompanyMessagesInterface.studentUpdateMessage()																							
2.3.55. ComposeMessageInterface.sendMessage()																							
2.3.56. StudentMessagesInterface.studentMessage()																							
2.3.57. NewsFeedList.displayNewsFeedList()						X																	
2.3.58. NewsFeedList.update()																							
2.3.59. Post.send()																							
2.3.60. Post.getHtml()																							
2.3.61. EventPost.getHtml()										X		X											

SRS Sections	3.1.1	3.1.2	3.1.3.1	3.1.3.2	3.1.3.3	3.1.3.4	3.2.1	3.2.2	3.2.3	3.2.4	3.2.5	3.2.6	3.2.7	3.3.1	3.3.2	3.3.3	3.3.4	3.4.1	3.4.2	3.5.1	3.5.2	3.5.3	3.5.4.1	3.5.4.2	3.5.5.1	3.5.5.2	3.5.5.3	4.1	4.2	4.3	4.4
2.2.58. UserSearch.displayResults()														X	X	X	X														
2.2.59. UserSearch.searchJobs()														X																	
2.2.60. UserSearch.searchCompanies()														X																	
2.2.61. AuthenticationList.isAuthorized()											X																				
2.2.62. EditPage.addData()																															
2.2.63. EditPage.editData()																															
2.2.64. EditPage.RemoveData()																															
2.3 Frontend Overview	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
2.3.1 Frontend Class Diagram	X	X	X															X	X												
2.3.2 Frontend Structure			X	X	X	X												X	X												
2.3.3. Companies Profile Page: UI Components																															
2.3.4. Companies Profile Page: UML Class Diagram																															
2.3.5. Companies Profile Page: Sequence Diagram																															
2.3.6. Companies Notifications: UI Components																															
2.3.7. Companies Notifications: UML Class Diagram																															
2.3.8. Companies Notifications: Sequence Diagram																															
2.3.9. Companies Search: UI Components																															
2.3.10. Companies Search: UML Class Diagram																															
2.3.11. Companies Search: Sequence Diagram																															
2.3.12. Companies News Feed: UI Components																															
2.3.13. Companies News Feed: UML Class Diagram																				X	X										
2.3.14. Companies News Feed: Sequence Diagram																															
2.3.15. Companies Job Postings: UI Components																X	X														
2.3.16. Companies Job Postings: UML Class Diagram																	X														
2.3.17. Companies Job Postings: Sequence Diagram																															
2.3.18. Students Profile Page: UI Components	X	X	X	X	X	X																									
2.3.19. Students Profile Page: UML Class Diagram	X	X	X	X	X	X																									
2.3.20. Students Profile Page: Sequence Diagram	X	X	X	X	X	X																									
2.3.21. Students Notifications: UI Components							X	X	X	X	X	X	X																		
2.3.22. Students Notifications: UML Class Diagram							X	X		X		X																			
2.3.23. Students Notifications: Sequence Diagram							X	X	X		X																				
2.3.24. Students Search: UI Components																															
2.3.25. Students Search: UML Class Diagram														X	X	X	X														
2.3.26. Students Search: Sequence Diagram														X	X	X	X														
2.3.27. Students News Feed: UI Components																		X	X												
2.3.28. Students News Feed: UML Class Diagram																		X	X												
2.3.29. Students News Feed: Sequence Diagram																		X	X												
2.3.30. Students Resume: UI Components																				X	X	X	X	X	X	X	X				
2.3.31. Students Resume: UML Class Diagram																				X	X	X	X	X	X	X	X				
2.3.32. Students Resume: Sequence Diagram																															
2.3.33 All Frontend UML Class Diagrams	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
2.3.34. Company.viewInfo()																															
2.3.35. SubmitInfo.submitForm()		X	X	X																											
2.3.36. StudentPage.submitResume()																									X	X	X				
2.3.37. ResumeWindow.openResumeForm()																				X	X	X			X	X	X				
2.3.38. ResumeForm.submitResumeForm()				X																					X	X	X				
2.3.39. ResumeForm.getHelpDetails()																								X							
2.3.40. Search.filter()													X			X	X														
2.3.41. Search.search()			X	X	X	X																									
2.3.42. Filter.filter()			X																												
2.3.43. Autocomplete.autocomplete()																															
2.3.44. CompanyRepresentative.viewInfo()																															
2.3.45. CompanyRepresentative.editPage()																															
2.3.46. Student.fillForm()																							X								
2.3.47. EditProfile.addData()		X																													
2.3.48. EditProfile.editData()		X																													
2.3.49. EditProfile.removeData()		X																													
2.3.50. StudentMessage.filter()													X																		
2.3.51. CompanyMessagesInterface.composeMessage()																															
2.3.52. CompanyMessagesInterface.studentMessage()																															
2.3.53. CompanyMessagesInterface. ResumeMatchMessage()																															
2.3.54. CompanyMessagesInterface.studentUpdateMessage()																															
2.3.55. ComposeMessageInterface.sendMessage()							X	X	X																						
2.3.56. StudentMessagesInterface.studentMessage()							X	X																							
2.3.57. NewsFeedList.displayNewsFeedList()																		X													
2.3.58. NewsFeedList.update()																		X													
2.3.59. Post.send()																			X												
2.3.60. Post.getHtml()																		X													
2.3.61. EventPost.getHtml()																															

SRS Sections	1.1.1	1.2.1	1.3.1	1.3.2	2.1.1	2.1.2	2.1.3.1	2.1.3.2	2.1.3.3	2.1.3.4	2.1.3.5	2.1.3.6	2.1.3.7	2.1.3.8	2.1.3.9	2.1.3.10	2.2.1	2.2.2	2.2.3	2.2.4	2.2.5	2.3.1	2.3.2	2.3.3	2.3.4	2.3.5	2.3.6	2.3.7	2.3.8	2.4.1	2.4.2	2.4.3	2.4.4	2.4.5	2.4.6	2.5.1	2.5.2	2.5.3	2.5.4	2.5.5	2.5.6	2.5.7	2.5.8	2.5.9	3.1.1	3.1.2	3.1.3.1	3.1.3.2	3.1.3.3	3.1.3.4	3.2.1	3.2.2	3.2.3	3.2.4	3.2.5	3.2.6	3.2.7	3.3.1	3.3.2	3.3.3	3.3.4	3.4.1	3.4.2	3.5.1	3.5.2	3.5.3	3.5.4.1	3.5.4.2	3.5.5.1	3.5.5.2	3.5.5.3	4.1	4.2	4.3	4.4
2.3.62. EditPage.addData()							■									■																														■																													
2.3.63. EditPage.editData()							■									■																														■																													
2.3.64. CompanyForm.submit()																																																																							■				
2.3.65. Admin.editJobPostings()																																				■																																							
2.3.66. Moderator.editJobPostings()																																			■																																								
2.3.67. AddPosts.onClickNewPost()																														■																																														
2.3.68. AddPosts.onClickSocialMedia()																																			■																																								
2.3.69. AddPosts.submitNewPost()																														■																																														
2.3.70. AddPosts.submitSocialMedia()																																			■																																								
2.3.71. JobList.addPost()				■																																■																																							
2.3.72. JobList.viewApplicants():							■																													■																																							
2.3.73. JobForm.submitJobForm()			■																					■													■					■																																			
2.3.74. JobPost.editPost()																																							■																																				
2.3.75. JobPost.deletePost()																																				■																																							
2.3.76. JobPost.getJobInfo()																																				■								■																															
2.4 Database Overview					■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■				
2.4.1 Company ERD									■							■																■				■																																							
2.4.2 Company Security ERD									■							■																■				■																																							
2.4.3 Student Security ERD																																															■	■	■	■																									
2.4.4 Student ERD																																													■	■	■	■	■	■	■	■	■	■	■	■	■						■												
2.4.5 Database Access DFD																■														■																																	■												
2.4.6 Database Sequence Diagram							■		■							■																																																												

3. Glossary

Word	Definition
Access Control List	A list of permissions associated with an object in a system.
Accolades	A list of awards.
Authentication	The authentication component receives user data and passes the entered password through the same hashing algorithm used on account creation to compare with the database's data if the entered password and stored password match.
Authorization	When a user entity (**company**, school, **student**, etc.) is in a session, each will have access to database resources (resumes, applications, notifications, etc.) in the web app unique to their ID and entity type.
Application Programming Interface	An API is a set of subroutines, definitions, protocols, and tools for building application software. ("Secure Logic App endpoint with API Management")
Application	A form used by a **company** to gather information on prospective employees.
Attachment	An extra part or extension that can be attached to something to perform a function, usually consisting of a file.
Autocomplete	A function that anticipates a **user's** input by offering several potential options based on the **user's** input.
Certification	An official document providing a certain level of achievement.
Certification Form	A **form** for the user to input currently held **certifications.**
Company	An organization consisting of employees seeking to hire additional employees.
Contact Information	Contact details about an entity include but are not limited to name, phone number, email, and address.
Company Candidate Search	A feature allowing **companies** to search **student users** based on desired **company candidate search filters.**
Company Candidate Search Filters	**Filters** including **GPA, project experience, skills,** and location.
Company Data	Information about the company, including **company name**, **company location**, **mission statement**, **social media,** and **company URL.**
Company Location	Place where business or production is conducted.
Company Name	The name that represents the **company.**

Word	Definition
Company Profile Page	A **page** with the collection of information associated with a given **company** that can contain the **company**'s **data.**
Company Representative	An authorized **user** that manages a **company profile page.**
Company Search Parameters	Searchable company **tags** such as **company locations**, **company names**, job titles, **pay scales**, **GPA requirements**, **skill requirements**, **full-time employment**, **internships**, and **internship dates/season**s.
Company URL	A **URL** associated with a **company.**
Contact Information	An individual's private or personal information, which another person, business, or entity can use to reach the individual, includes email, phone number, and current address.
Data	A collection of text, pictures, or documents.
Degree information	Information pertaining to a **user's** specific degree, including what type of degree, **GPA**, and expected graduation date.
Education Experience	Any experience a person gains while attending a collegiate or trade school.
Education Form	A **form** which allows users to input information on past and current **educational experiences**.
Employer	A person or organization that employs people; a recruiter or CEO can make a **company profile page.**
Filter	Criteria are applied to **search results to** reduce the number of returned **results** based on the same criteria.
Follow	When one user subscribes to the posts of another, allowing for notifications to be sent to the user and posts to be displayed on the user's news feed.
Form	A hypertext document with blank input fields for information to be inserted.
Free-Form Text Area	A text box in an entry form where the user can type several lines of text, possibly even line breaks, tabs, etc.
Full Time	A non-**student** position with no set duration limit.
GPA	Stands for Grade Point Average. GPAs are determined by a combination of course credits, individual grades, and semester hours spent in the class.
GPA requirements	Minimum **GPA** desired for employment.
Graduation Rate	The percentage of **students** completing their undergraduate degree within 150% of the time for the program.
Help Feature	Information is available to the user to help indicate what should be entered into a text field.

Word	Definition
Internship	A temporary position for a **student** employee to work within a **company** to gain experience and training.
Internship dates/seasons	Yearly times when hiring occurs.
Job Experience Form	A form for the user to input past and current **job experience** information.
Job Experience	Any experience that a person gains while working in a field or occupation. [5]
Job Posting	A **post** consisting of **job posting data** made by an **employer** about a job that needs to be filled. Viewable in the **news feed** of **student users.**
Job Posting Data	Information included in the job posting: **company** name, job title, **required skills**, job description, hours, position type, compensation, location, **contact information**, and links to apply for the job.
Mandatory Field	A text field that requires **user** input before the submission is possible.
Message	"A message is a discrete unit of communication intended by the source for consumption by some recipient or group of recipients." [6]
Miscellaneous Form	A form for the **user** to input all information that is not included in any other form.
Mission Statement	A statement of a **company** or individual's objectives or goals.
Modern Web Browser	A web browser that supports HTML5, CSS3, and ES6.
Mute	To turn off or cease to receive **notifications**.
News Feed	A vertically scrollable window containing a collection of relevant information or news about **students**, universities, and **employers**.
Notification	A **message** sent to a device, visually displaying a specified event to the user, and containing **notification data**.
Notification Data	Contains a picture, text, or other **attachments**.
Notification Push	The delivery of information from a software application to a computing device without a specific request from the client.
Notification Trigger	To initiate, actuate, or set off a **notification**.
Page	A hypertext document connected to the system.
Pay Scale	Amount of compensation for services rendered.

Word	Definition
Picture	A formatted image embedded within a body of text according to size and format.
Portfolio	A collection of a **user's** work (such as programs, **projects**, etc.) compiled over a period of time and used for exhibiting the **user's** talents, abilities, or progress.
Post	Text, images, or other content items published on a social media profile.
Profile	A visual display of personal data associated with a **user**.
Project	A piece of planned work or activity intended to achieve a particular goal. [7]
Project Experience	Any experience a person gains by working on **projects** related to the industry in which they are seeking a job.
Project Form	A form used by the **user** to input past and current **student projects** they have completed.
Required Skills	Skills the **employer** is looking for candidates to have.
Required Skills List	A text box on a **job posting** contains the **required skills** desired by the **company**.
Results	Items returned from a **search**.
Resume	A plain text document that provides an **employer** with a **student user's** information.
Resume Data	Information about the user consists of **contact information**, **user work experience**, **degree information**, and **skills**.
Resume Form	Includes a **Job Experience Form**, a **Skills Form**, a **Volunteer Experience Form**, a **Project Form**, an **Education Form**, a **Certification Form**, and a **Miscellaneous Form**.
Resume Form Submission	When the followers of the **student user** are permitted to view the **student user's resume form**.
Search	The process of identifying and displaying the desired **result** by applying **filters** to all the system data.
Search Bar	A **text input field** specifically used to **search**.
Settings	A collection of controls that enables the **user** to configure the appearance or actions of the system.
Site	A location connected to the Internet that maintains one or more pages on the World Wide Web. [8]
Skill Requirements	Desired attributes and strengths for employment.
Skills Form	A **form** for the **user** to input **soft skills** or **required skills**.

Word	Definition
Social Media	An online platform in which a **user** can communicate with other **users** on a large-scale basis.
Soft Skills	Skills and traits that enable effective and harmonious interpersonal communication.
Standard Page Loading Time	The page shall load in under 3 seconds.
Student	A person who is studying at a school or college. [9]
Student Account	A collection of data associated with a **student** of a social media system comprised of a username, password, and a **student profile**.
Student Profile	A collection of settings and data associated with a **student** which can contain personal data.
Student Profile Page	A visual display of a **student's** data associated with a specific **student** referring to the explicit digital representation of the **student's** identity.
Student User	A **user** that manages a **student profile page**.
Student User Followers	**Students** that **follow** a **company**.
System	Collection of software, including **web servers**, backend databases, mobile applications, and web applications.
Tag	A one-to-three-word long keyword that summarizes the outcomes of a **resume**, **portfolio**, comment, endorsement, or other items on a **user's** profile.
Text Field	A location where the user can input text on the interface.
University Statistics	National ranking, overall **student GPA**, and **graduation rates**.
URL	Acronym for Uniform Resource Locator. URL refers to a specific location of web resources on a computer network.
User	Individual or group that uses a system.
Volunteer Experience Form	A **form** for the **user** to input past and current information regarding **volunteer experience**.
Volunteer Experience	Any experience that a person gains while volunteering in a field or occupation. [5]
Web Server	Program that uses the Hypertext Transfer Protocol to serve files that makeup web pages.
Wildcard Searches	A **search** based on matching character patterns.

Word	Definition
Work Experience	Contains information about the **user's** current and past jobs, includes the following information: **company** name, **employer's** name, **employer's** phone number, and a short description detailing what the job entailed.

4. Abbreviations

Abbreviation	Word
ACL	Access Control List
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DFD	Data Flow Diagram
ERD	Entity Relationship Diagram
UI	User Interface
UML	Unified Modeling Language
UX	User Experience