SYSTEM DESIGN DOCUMENT

LINK UP

Ashtian Dela Cruz
Bahar Chidem
Keerthiha Baskaran
Matthew Wu
Vaibhav Lakshmi Santhanam

Table of Contents

FRONTEND

Sign Up Page **4**
Anonymous user name generation **5**
Resume zoom **5**
Resume upload **5**
UseZoomModal **6**
Deletion of resumes from profile page **7**
Direct Messaging **9**
Rerouting from root **10**
Landing Page

BACKEND **10**
Schemas **12**
Delete Resumes API **17**
Display Resumes API **17**
LoginUser API **19**
Get user API **19**
Get User Resumes API **20**
newUser API **20**
Get Preferences **21**
Update preferences API **21**
Get User by ID API **22**
Upload Resumes API **22**
Get Public Resumes API **23**
Cache **23**
Get Trending Comments API **24**
Add Trending Comments API **24**
Update Votes API **25**
Get Messages API **25**
Send Message API **25**
Mark Messages As Read API **26**
Add Swipe API **26**
Check Match API **26**
Get Swiping Resumes API **27**
Delete Conversation API **27**
Delete Message API **27**
Block Users API **29**
updateDMStatus API **29**
getDMStatus API **29**

getUsername API **30**
getComments API **30**
addComments API **30**
Dependencies: **31**
System Interaction: **33**
Software Architecture: **34**
LinkUp system decomposition: **35**

Front-end

Login Page

Class Name: Login.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities:	Collaborators:
Render a form that allows input for user emails and password for log in	React, Login.css
Checks if a user's credentials are valid for logging in by making an API call to the backend	server.js (backend router), loginUser.js (where the request is handled)
Display error messages and styling based on the API call response	Login.css
Reroutes already authenticated users to the profile page	react-auth-kit/hooks/useIsAuthenticated

Sign Up Page

Class Name: SignUp.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Render a form that allows input for user emails and password for sign up	React, SignUp.css
Validate user input to ensure required fields are non-empty and/or follow a specific format	
Adds a user to the database by making an API call to the backend	server.js (backend router), newUser.js (where the request is handled)
Display error messages and styling based on the API call response	SignUp.css
Reroutes already authenticated users to the profile page	react-auth-kit/hooks/useIsAuthenticated

Anonymous user name generation

Class Name: newUser.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Using a random number generator to randomly select an (animal, number) pair for the username	Math, Math.random()
Checks if there already exists a user with the randomly generated username before returning	Mongoose, User Model

Resume zoom

Class Name : ResumeZoom.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Render a file upload section with a progress bar that visually states the upload status.	React, ResumeZoom.css
Manage public/private status of uploaded resume	React, ResumeUpload.css, UploadPopUp.css

Resume upload

Class Name : ResumeUpload.js, UploadPopUp.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Render a zoomable and draggable PDF	React, ResumeUpload.css

viewer	UploadPopUp.css
Handle touch and mouse events	React, ResumeUpload.css
Close the modal when the close button is clicked	React, ResumeUpload.css

UseZoomModal

Class Name : UseZoomModal.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Manage the state for the selected resume	React, ResumeZoom.js
Provide functions to open and close the zoom modal:	React, ResumeZoom.js
Render the ResumeZoom component when a resume is selected	React, ResumeZoom.js

Deletion of resumes from profile page

Class Name : ProfilePage.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Fetch and display resumes	Axios, server.js
Handle resume deletion with the confirmation popup	React, Axios, Modal component
Adding resumes with a modal interface (interface has same responsibilities as ResumeUpload.jss)	ResumeUploadModal from UploadPopUp.js, Axios
Fetches user preferences	React, react-auth-kit/hooks/useAuthUser, react-auth-kit/hooks/useIsAuthenticated
Edit user preferences	React, axios
Allow users to choose and update their profile pic from a selection of profile pics	setProfilePic.js
Display the current user’s profile pic	getProfilePic.js, extract-colors

Preferences Page

Class Name : PreferencesPage.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Render a form for setting user preferences	React
Handle user input for preferences such as field of interest, work experience level, education, and geographic location	Select (from 'react-select')

Update user preferences on form submission by making an API call to the backend	useAuthUser (from 'react-auth-kit/hooks/useAuthUser')
Display a success or error message based on the API call response	'./Preferences.css' (custom CSS file)
get the current user’s preferences	axios, React

Direct Messaging

Class Name: DirectMessages.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Render a form that allows the current user choose another user to message and shows which messages are unread	React, DirectMessages.css
Retrieves messages from other users (through the database)	getMessages.js, message.js
Sends new messages to other users (through the database)	sendMessage.js, message.js
Ensures list of other users to choose from are matched with the current user	checkMatch.js
Listens for and retrieves new messages from others in real-time	server.js (uses changeStream and mongoose collection.watcher())
Marks another user's messages as read (through the database)	markMessagesAsRead.js, message.js
Displays pdf's of resumes that a user has commented on	
Deletes a single message from the user's side (through database)	deleteMessage.js
Deletes all messages from the user's side (ie, delete conversation) (through database)	deleteConversation.js
Block User , Allows users to block other users if they don't want to communicate with them any longer	Blocking User.js BlockUser.js(modal code)
Reroutes already authenticated users to the profile page	react-auth-kit/hooks/useIsAuthenticated
Display profile pics of each user that the current user can message	getProfilePic.js, extract-colors

Rerouting from root

Class Name : CheckIfUserLoggedIn.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Redirect user to logged in page, if the user is not logged in on the machine, or to the landing page, otherwise	React, useAuthUser, navigate

Landing Page

Class Name: LandingPage.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Display resumes from other users that are public and that match the user preferences	React, Axios, Worker, Viewer
Link/Unlink buttons	onClick(), Axios
Check if user has uploaded a resume already	Axios
Check if match was made	Axios

Trending Resumes Page

Class Name : TrendingResumes.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:

Manages component states for displaying resumes and user interactions, fetches and renders public resumes from the API, implements UI elements for comments and resume impressions, controls horizontal scrolling.	Axios, react-pdf-viewer/core, Sidebar.js, VisibilityIcon, AddIcon, React hooks
Renders the navigation/sidebar component for user interaction within the main application layout.	TrendingResumes.css
Loads and manages the PDF.js worker script for processing PDF files.	react-pdf-viewer/core
Allow users to post comments on trending resumes and retrieve comments related to them.	postTrendingComment.js, onClick(), Axios, getTrendingComment.js
Manage voting on comments.	getTrendingComment.js, onClick(), Axios, getTrendingComment.js
-Filters hateful comments and prevents user from entering them	FilterMsg.js FilterComment.js
Manages votes for comments, fetches the vote status and also updates the votes	TrendingResumes.css, onClick(), Axios, votes.js
Manages the replies for comments, fetches the comments and posts them.	TrendingResumes.css, onClick(), Axios, gettReplies.js

Resume Comment Page

Class Name : ResumeComment.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Manages adding comments and highlights to resumes.	React, Axios, React-pdf-viewer, React-router-dom, react-auth-kit/hooks/useIsAuthenticated, react-auth-kit/hooks/useAuthUser,

	@react-pdf-viewer/core/lib/styles/index.css,
Handles user interactions, text selections, and submission of comments.	React, Axios, React-pdf-viewer, React-router-dom, @react-pdf-viewer/core/lib/styles/index.css
Fetches and displays existing comments and highlights from the database.	react-pdf-viewer/core
-Sends a direct message notification with the comment details and resume after submission.	React, Axios, React-pdf-viewer, React-router-dom, ResumeComment.js
-Filters hateful comments and prevents user from entering them	FilterMsg.js FilterComment.js

View Resume Comments Page

Class Name : ViewResumeComments.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Manages the display of comments and highlights for a specific resume and commenter and loads the comments by the commenter on the PDF.	Axios, react-pdf-viewer/core, useParams, ViewResumeComments.css

Reviewed Resumes

Class Name: ReviewedResumes.js
Parent Class(if any): n/a

Subclass (if any): n/a	
Responsibilities:	Collaborators:
Manage state related to resumes, comments, selected commenters, and highlights.	React, Axios, Worker, Viewer
Fetch resumes and their associated comments upon component load or user actions.	onClick(), Axios
Navigate through different resumes and comments.	ReviewedResumes.css, onClick(), Axios
Display resumes in a PDF viewer and highlight comments within the document.	@react-pdf-viewer/core

Back-end

Schemas

Class Name : user.js (User Schema)	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Store and define the structure of a user in the database including :	mongoose
• `anon_username: String` • `email: String` • `password: String` • `field_of_interest: String` • `work_experience_level: String` • `education: String` • `location: String` • `avatar: String` • `salt: String` • `verified: Boolean`	

• `verificationToken:String`	

Class Name: resume.js (Resume Schema)	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Represent the structure of a resume document in the database which includes the following fields :	mongoose
uploader_id: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }, // Reference to User schema file_path: String,Store the file path of the resume document. public: Boolean,Track visibility (public or private). num_swipes: Number (Counts the number of right swipes on the document)	User (referenced by `uploader_id`)

Class Name: swipes.js (User Schema)	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Store and define the structure of a swipe in the database including:	mongoose
• `user_id: mongoose.Schema.Types.ObjectId, references user schema` • `Resume_id: mongoose.Schema.Types.ObjectId, references resume schema` • `Uploader_id: mongoose.Schema.Types.ObjectId, references user schema`	

• `accept: Boolean`	

Class Name : message.js (User Schema)	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Store and define the structure of a message in the database including :	mongoose
• `to: String,` • `from: String,` • `timestamp: String,` • `message: String,` • `read_by_to: Boolean`	

Class Name : comments.js (Comments Schema)	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Store and define the structure of a comment on a resume in the database including :	mongoose
• `resumeId: Resume Object Type,` • `user: String,` • `comment: String,` • `highlightedText: String,` • `position: {top: Number, left: Number, width: Number, height: Number}`	

Class Name : connections.js (Connections Schema)	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Store and define the structure of a connection between two users when they accept each other on DMs in the database including :	mongoose
• `user1: String,` • `user2: String,` • `accepted: Boolean,` • `createdAt: {type: Date, default: Date.now}`	

Class Name : blockedUsers.js (Blocked User Schema)	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Store and define the structure of a message in the database including :	mongoose
• `_id:mongoose.Schema.Types.ObjectId,` • **`username:`** `String` • **`blockedUsernames:`** `Array of String`	

Class Name : trendingComments.js (Blocked User Schema)	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Store and define the structure of a trending page comment in the database including :	mongoose
• **resumeId**:ObjectId • **text**:String • **username**:String • **createdAt**:Date • **votes**:Number • **parentId**:ObjectId • Default: null	

Class Name : votes.js (Votes Schema)	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Store and define the structure of a trending page comment in the database including :	mongoose
• **commentId**:ObjectId • **userId**:ObjectId • voteType: Number	

Class Name: server.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Connects to MongoDB Atlas database	Mongoose
Handles incoming GET and POST requests from front end by rerouting requests to specific modules responsible for handling the request	'./API/getUser' './API/loginUser' './API/newUser' './API/EmailVerification' './API/UpdatePreferences' './API/getPreferences' ‘ ./API/uploadResumes’ './API/getUserResumes' './API/displayResumes' './API/deleteResumes'

Delete Resumes API

Class Name : deleteResumes.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities:	Collaborators:
- Receive resumes to be deleted from user -Delete resumes from MongoDB resumes collection -Delete resumes from GridFs bucket -Send appropriate HTTP response based on the outcome(File found/deleted/did not delete)	Resume Model (to fetch and update resume data)

Display Resumes API

Class Name : displayResumes.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities:	Collaborators:
-Receive filename to display - Find file in the GridFs storage -Check if it is in PDF format - Stream PDF from bucket -Send appropriate HTTP response based on the outcome (File valid/invalid/failed to display)	

Email Verification API

Class Name : EmailVerification.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities:	Collaborators:
-Receive HTTP requests with verification tokens. -Validate the tokens against the User model. -Update the verification status of a user. -Send appropriate HTTP response based on the outcome (token valid, invalid, or already verified).	User Model (to fetch and update user data) Express.js (to handle HTTP requests and responses)

LoginUser API

Class Name : loginUser.js	
Parent Class(if any): N/A Subclass (if any): N/A	
Responsibilities: -Validate user credentials (email and password). -Check if user is verified (via email verification) -Generate access tokens using JWT. -Return user details and token, or error messages.	Collaborators: -User Model (to fetch user data and verify credentials) -bcryptjs (to compare hashed passwords) -jwt (to generate JWT tokens) -dotenv (to manage environment variables

Get user API

Class Name : getUser.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities: -Retrieve all users from the database. -Return user data or a "No user found" message. -Handle errors and return the appropriate HTTP status code and message.	Collaborators: User Model (to query user data)

Get User Resumes API

Class Name : getUserResumes.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities:	Collaborators:
-Receive the user's id - Find all resumes with the same uploader_id -Convert the resumes to json format -Send appropriate HTTP response based on the outcome (fetched resumes/ failed to fetch).	Resume Model (to fetch and update resume data)

newUser API

Class Name : newUser.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities: -Create and configure an email transporter using OAuth2 and nodemailer for sending emails. -Send a verification email to new users using Mailgen to generate the email content. -Generate unique anonymous usernames to ensure privacy. -Handle new user registration including data validation, password hashing, and database storage. -Respond to API requests with appropriate	Collaborators: -User Model: Interacts with the database to check for existing users and save new users. -bcryptjs: Used for hashing and salting passwords securely. -crypto: Generates random tokens for email verification links. nodemailer and Google APIs (OAuth2): Manage email sending functionalities. -Mailgen: Generates human-readable HTML for emails. dotenv: Manages environment variables for secure access to API keys and secrets.

success or error messages.	-Express: Framework used to handle HTTP requests and middleware setup.

Get Preferences

Class Name : getPreference.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities: -Retrieve user preferences based on email from the database. -Handle HTTP GET requests to fetch user-specific preferences. -Respond with the relevant user preferences in JSON format. -Handle errors during the fetch operation and respond accordingly	Collaborators: -Mongoose -User Model: Interacts with the database to check for preferences of users and retrieve them.

Update preferences API

Class Name : updatePreferences.js	
Parent Class(if any): N/A Subclass (if any): N/A	
Responsibilities:	Collaborators:
Handle the update of user preferences based on the provided email and preference data.	`User` model (from `../schema/user`)
Find the user by email.	

Update the user's preferences using the user ID if the user is found.	
Log the updated user information.	
Handle and log errors if they occur during the process.	

Get User by ID API

Class Name: getUserById.js	
Parent Class(if any): N/A Subclass (if any): N/A	
Responsibilities:	Collaborators:
get User by the ID in the parameter	`User` model (from `../schema/user`)

Upload Resumes API

Class Name : uploadResumes.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities:	Collaborators:
-Create the storage engine for files to upload in storage -Create the new resume based on parameters passed in -Save resume and find uploader_id for resume -Send appropriate HTTP response based on the outcome (uploaded/error uploading).	User Model (to fetch and update user data) Resume Model (to fetch and update resume data) Path module to get the file path Crypto module to assign random bytes to files uploaded GridFsStorage for storing and retrieving files Multer for uploading files

Get Public Resumes API

Class Name : getPublicResumes.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities:	Collaborators:
-Retrieve top public resumes sorted by 'num_swipes' - Cache top resumes for efficiency - Serve cached resumes if available - Handle and report errors	Resume Model Cache.js

Cache

Class Name : cache.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities:	Collaborators:
- Connect to Redis server - Retrieve serialized data by key and deserialize - Store data with an expiry using serialization - Handle Redis connection and data operation errors	Redis Node.js

Get Trending Comments API

Class Name : getTrendingComment.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities:	Collaborators:
- Handle the HTTP request to fetch trending comments. - Organize comments and their replies into a structured format. - Send the structured data as a JSON response. - Aggregate votes for each comment, calculating the total number of votes and individual user votes. - Attach vote data to each comment.	Trending Comment model Express.js Vote model Mongoose

Add Trending Comments API

Class Name : postTrendingComment.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities:	Collaborators:
- Handle the HTTP POST request to create a new trending comment. - Validate and extract data from the request body. - Save the new comment to the database.	TrendingComment module Express.js Mongoose

Update Votes API

Class Name : votes.js	
Parent Class(if any): List the parent class if applicable Subclass (if any): List all the subclasses separated by	
Responsibilities:	Collaborators:
- Handle the HTTP request to update the vote count on a comment. - Validate and extract commentId and delta from the request body. - Perform the update operation on the database.	TrendingComment module Express.js

Get Messages API

Class Name: GetMessages.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Queries the database and returns messages sent to or from the current user	mongoose, messages.js, DirectMessages.js

Send Message API

Class Name: GetMessages.js
Parent Class(if any): Subclass (if any):

Responsibilities:	Collaborators:
Inserts a new message object into the database using content passed in from the front-end	mongoose, messages.js, DirectMessages.js

Mark Messages As Read API

Class Name: MarkMessagesAsRead.js	
Parent Class(if any): Subclass (if any):	
Responsibilities:	Collaborators:
Updates all messages sent by another user as being read by the current user	mongoose, messages.js, DirectMessages.js

Add Swipe API

Class Name: addSwipe.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Add document to swipes collection with proper user_id, resume_id, uploader_id, and accept boolean value	mongoose, LandingPage.js

Check Match API

Class Name: checkMatch.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Return boolean value based on whether	mongoose, LandingPage.js

the current user_id has matched with the uploader_id	

Get Swiping Resumes API

Class Name: getSwipingResumes.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Return all the resumes that are public, match user preferences, and also are not the current user's	mongoose, LandingPage.js

Delete Conversation API

Class Name: deleteConversation.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Edits all messages sent by the current user to be "deleted by sender", and marks deleted_by_from field as true	mongoose, DirectMessages.js

Delete Message API

Class Name: deleteConversation.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:

Edits a specific message sent by the current user to be “deleted by sender”, and marks deleted_by_from field as true	mongoose, DirectMessages.js

Set Profile Pic API

Class Name: setProfilePic.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Updates the profile pic string of the passed in user in the database	mongoose, ProfilePage.js

Get Profile Pic API

Class Name: getProfilePic.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Retrieve the profile pic string of the passed in user in the database	mongoose, ProfilePage.js

Block Users API

Class Name:BlockingUser.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
This code defines three API endpoints using Mongoose: one to block a user by adding them to a blocked list, one to check if a user is blocked, and one to fetch the list of blocked users for a given user.	mongoose, DirectMessages.js

updateDMStatus API

Class Name:getDmStatus.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Updates the status of a direct message first sent to another user based on the accept/decline response from the other user.	mongoose, connection.js

getDMStatus API

Class Name:getDmStatus.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Gets and fetches the acceptance status of a DM conversation from the database, when a new conversation is started.	mongoose, connection.js

getUsername API

Class Name:BlockingUser.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Fetches and returns the username of a user based on user ID.	mongoose, user.js

getComments API

Class Name:BlockingUser.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Fetches and retrieves comments for a specific resume by a specific user and returns the comments, resume and metadata.	mongoose, comments.js

addComments API

Class Name:BlockingUser.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Adds comments to resumes and stores comment data, including highlighted text and position.	mongoose, comments.js

Filter hateful comments util

Class Name:CommentFilter.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
Filter out custom-defined profane words from text. Determine if a given text contains any profane or undesirable words. Add custom words to the filter list dynamically.	mongoose, comments.js, Trending comment.js

Votes API

Class Name: votes.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
- Handle voting on a specific comment by a user. - Validate if the user has already voted on the comment. - Update the existing vote if conditions change (e.g., changing from an upvote to a downvote). - Remove the vote if the new vote type is '0' (indicating no vote). - Update the total vote count on the comment after any voting action. - Fetch the current voting status of a specific comment by a specific user. - Return the vote type to the requester.	Vote Model Trending comment model Mongoose, express.js

Get Replies API

Class Name: getReplies.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
- Fetch all replies related to a specific parent comment from the database. - Aggregate votes for each reply to calculate total votes and determine user-specific votes. - Attach the aggregated vote data to each reply.	Vote Model Trending comment model Mongoose, express.js

Get User Resumes and Comments API

Class Name: getUserResumesAndComments.js	
Parent Class(if any): n/a Subclass (if any): n/a	
Responsibilities:	Collaborators:
- Retrieve all resumes uploaded by a specific user from the database. - For each retrieved resume, fetch associated comments. - Combine resume and comment data into a single response object for each resume.	Resume Model, Comment model, express.js, mongoose

Dependencies:

- "bcryptjs": "^2.4.3",
- "cors": "^2.8.5",
- "crypto": "^1.0.1",
- "dotenv": "^16.4.5",
- "express": "^4.19.2",
- "googleapis": "^140.0.0",
- "jsonwebtoken": "^9.0.2",
- "mailgen": "^2.0.28",
- "mongodb": "^5.9.1",
- "mongoose": "^6.13.0",
- "multer": "^1.4.4",
- "multer-gridfs-storage": "^5.0.2",
- "nodemailer": "^6.9.13",
- "path": "^0.12.7",
- "react-auth-kit": "^3.1.3"
- "@emotion/react": "^11.11.4",
- "@emotion/styled": "^11.11.5",
- "@mui/icons-material": "^5.15.20",
- "@mui/material": "^5.15.19",
- "@react-pdf-viewer/core": "^3.12.0",
- "@react-pdf-viewer/default-layout": "^3.12.0",
- "@react-pdf-viewer/scroll-mode": "^3.12.0",
- "@testing-library/jest-dom": "^5.17.0",
- "@testing-library/react": "^13.4.0",

- "@testing-library/user-event": "^13.5.0",
- "body-parser": "^1.20.2",
- "axios": "^1.7.2",
- "react": "^18.3.1",
- "react-dom": "^18.3.1",
- "react-router-dom": "^6.23.1",
- "react-scripts": "^5.0.1",
- "react-select": "^5.8.0",
- "web-vitals": "^2.1.4"

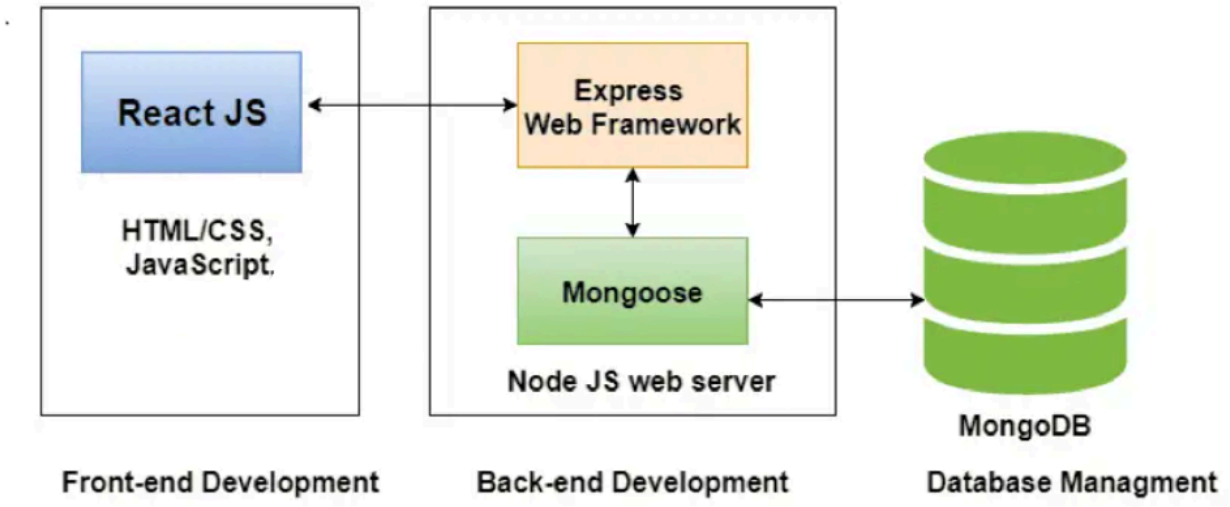
System Interaction:

The system's interaction with its environment assumes:

- Backend compatibility with various server OS (Linux, Windows, macOS) using Node.js and Express.js.
- MongoDB for database interactions, adaptable for local setups or cloud services like MongoDB Atlas.
- Frontend deployment in environments supporting modern browsers (Chrome, Firefox, Safari, Edge) with React.
- Development requires Node.js and npm for dependency management and build processes

Software Architecture:

This project employs a Three-Tier Architecture based on the MERN Stack. Users engage with the application through a React-based frontend, which serves as the client interface. Upon user requests, React sends HTTP requests to the backend server, developed with Node.js and Express.js. The server processes these requests by interacting with a MongoDB database, facilitating data storage and retrieval. This architecture ensures efficient communication between the client and server, providing users with a seamless and responsive experience.



The frontend employs Material-UI to ensure a consistent and visually appealing interface across various components. This setup not only enhances usability but also maintains a cohesive style throughout the application.

On the backend, LinkUp utilizes Node.js coupled with the Express.js framework to handle API requests efficiently, supporting a variety of operations that allow users to update preferences, manage profiles, and interact with the system. These requests facilitate CRUD operations

interacting seamlessly with our MongoDB database, which is designed to store user profiles, preferences, and anonymous interactions securely.

Our database architecture ensures robust data integrity and security, with input validation measures to prevent invalid data submissions. This comprehensive architecture supports efficient data handling and provides a seamless, secure user experience on the LinkUp platform.

LinkUp system decomposition: