SYSTEM DESIGN DOCUMENT

LINK UP

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

Contents:

Front-end

Login

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

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 upload

Class Name : ResumeUpload.js, UploadPopUp.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, ResumeUpload.css UploadPopUp.css
Manage public/private status of uploaded resume	React, ResumeUpload.css, UploadPopUp.css
Handle submission and update upload status upon completion or error	Axios, server.js, FormData API
Handle file selection and validate type (PDF only)	Browser’s File API

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

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)

Back-end

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
<ul><li>`anon_username: String`</li><li>`email: String`</li><li>`password: String`</li><li>`field_of_interest: String`</li><li>`work_experience_level: String`</li><li>`education: String`</li><li>`location: String`</li><li>`avatar: String`</li><li>`salt: String`</li><li>`verified: Boolean`</li><li>`verificationToken:String`</li></ul>	

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: 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 success or error messages.	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. -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.	

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

Dependencies:

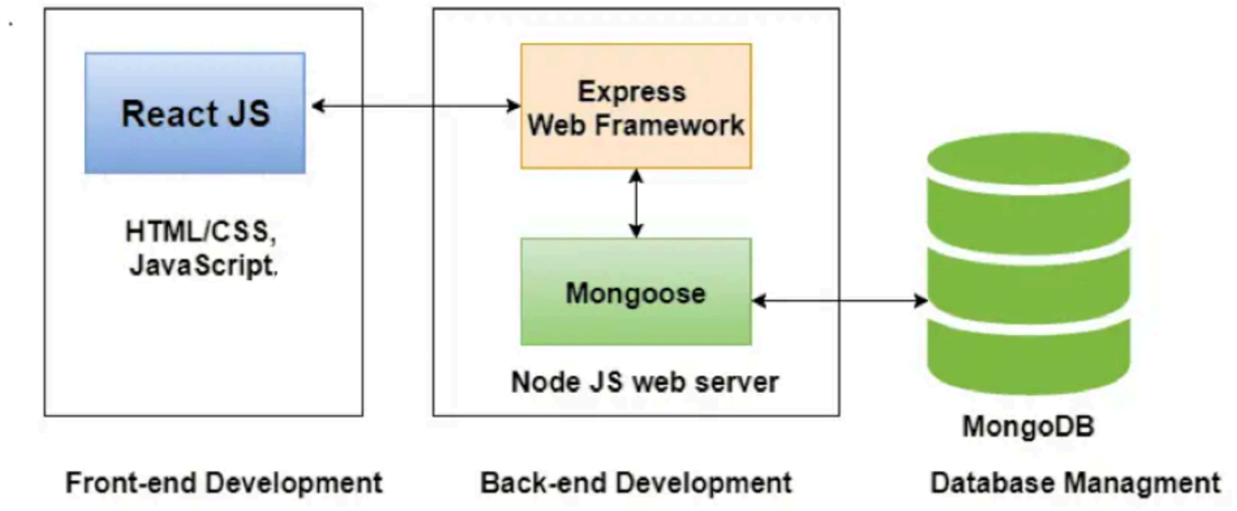
- "bcryptjs": "^2.4.3",
- "cors": "^2.8.5",
- "crypto": "^1.0.1",
- "dotenv": "^16.4.5",
- "express": "^4.19.2",
- "googleapis": "^140.0.0",
- "jsonwebtoken": "^9.0.2",
- "mailgen": "^2.0.28",
- "mongodb": "^5.9.1",
- "mongoose": "^6.13.0",
- "multer": "^1.4.4",
- "multer-gridfs-storage": "^5.0.2",
- "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"

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.

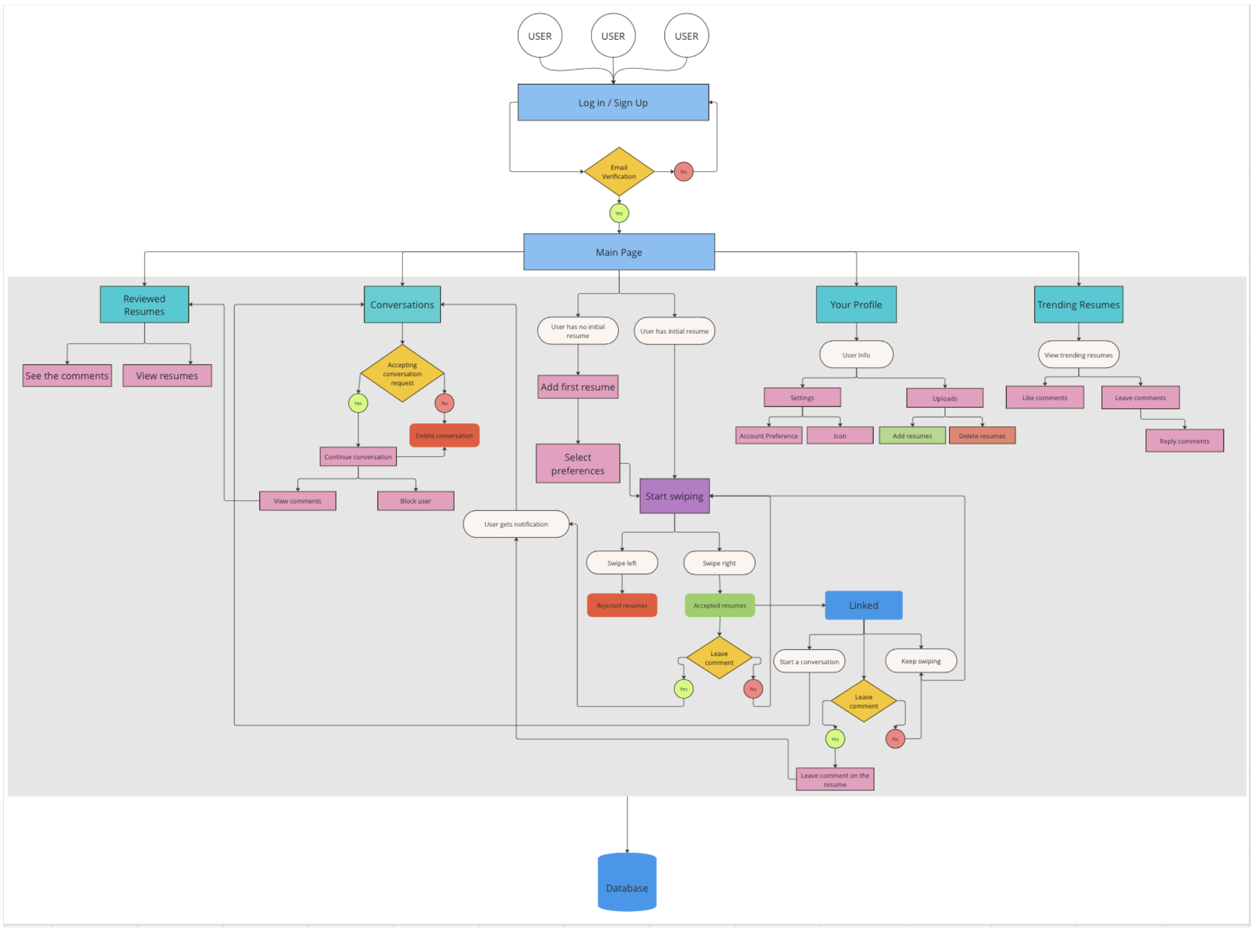


The frontend employs Material-UI to ensure a consistent and visually appealing interface across various components. This setup not only enhances usability but also maintains a cohesive style throughout the application.

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


USER
USER
USER
Log in / Sign Up
Email Verification
No
Yes
Main Page
Reviewed Resumes
See the comments
View resumes
Conversations
Accepting conversation request
Yes
No
Delete conversation
Continue conversation
View comments
Block user
User gets notification
User has no initial resume
User has initial resume
Add first resume
Select preferences
Start swiping
Swipe left
Swipe right
Rejected resumes
Accepted resumes
Leave comment
Yes
No
Linked
Start a conversation
Keep swiping
Leave comment
Yes
No
Leave comment on the resume
Your Profile
User Info
Settings
Uploads
Account Preference
Icon
Add resumes
Delete resumes
Trending Resumes
View trending resumes
Like comments
Leave comments
Reply comments
Database