

APPLIFY: TRACK AND APPLY

A PROJECT REPORT

Submitted By:

Vani Seth (201B299)

Vinayak Rao Dikshit (201B308)

Virad Chaurasia (201B354)

Under the guidance of: Prof. Mahesh Kumar



MAY-2024

Submitted in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science & Engineering

JAYPEE UNIVERSITY OF ENGINEERING & TECHNOLOGY,

AB ROAD, RAGHOGARH, DT. GUNA-473226 MP, INDIA

DECLARATION

We hereby declare that the project entitled as “**Applify: Track and Apply**” in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science and Engineering submitted at Jaypee University of Engineering and Technology, Guna, as per the best of our knowledge and belief there is no infringement of intellectual property rights and copyright. In case of any violation we will solely be responsible.

Vani Seth (201B299)

Vinayak Rao Dikshit (201B308)

Virad Chaurasia (201B354)

Place: Jaypee University of Engineering & Technology Guna,
Madhya Pradesh (India) – 473226

Date:



JAYPEE UNIVERSITY OF ENGINEERING & TECHNOLOGY

Grade 'A+' Accredited with by NAAC & Approved U/S 2(f) of the UGC Act, 1956

A.B. Road, Raghogarh, Dist: Guna (M.P.) India, Pin-473226

Phone: 07544 267051, 267310-14, Fax: 07544 267011

Website: www.juet.ac.in

CERTIFICATE

This is to certify that the work titled “**Applify: Track and Apply**” submitted by “**Vani Seth, Vinayak Rao Dikshit, Virad Chaurasia**” in partial fulfillment for the award of the degree of Bachelor of Technology of Jaypee University of Engineering & Technology, Guna has been carried out under my supervision. As per best of my knowledge and belief there is no infringement of intellectual property rights and copyright. Also, this work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma. In case of any violation, the student concerned will solely be responsible.

Prof. Mahesh Kumar

Professor, Dept. of Computer Science and Engineering

JUET, Guna, M.P.

Date:

ACKNOWLEDGEMENT

We would like to express our gratitude and appreciation to all those who gave us the opportunity to complete this project. Special thanks are due to our supervisor Prof. Mahesh Kumar whose help, stimulating suggestions and encouragement helped us in all the time of the development process and in writing this report. We sincerely thank him for the time spent proofreading and correcting our many mistakes.

We would also like to thank our parents and friends, who helped us a lot in finalizing this project within a limited period. Last but not least, we are grateful to all the team members of this project.

Vani Seth (201B299)

Vinayak Rao Dikshit (201B308)

Virad Chaurasia (201B354)

Date:

EXECUTIVE SUMMARY

Applify: Track and Apply is a Chrome extension designed to revolutionize job application management across various websites. It automates data collection, offering a centralized hub for application organization, while empowering users with insightful data visualization and data sharing capabilities.

Applify simplifies the process with a single click to initiate application tracking on job boards. The extension leverages Selenium webdriver to automatically gather crucial details like job title, company name, and application date. Users benefit from a dedicated "My Applications" section, providing a comprehensive list of all tracked applications, including company names, application dates, job titles, and current statuses. This centralized hub allows users to easily update the status (Open, Closed, Pending, etc.) for each application, ensuring clear and efficient tracking.

Applify goes beyond simple tracking by offering valuable data insights. The "Summary and Insights" section utilizes pie charts to visually represent the distribution of applications across different stages (e.g., Open, HR Round Cleared). This visual representation empowers users to gain a deeper understanding of their application performance and identify areas for improvement. Furthermore, Applify fosters collaboration by allowing users to share application data. It facilitates exporting all details as a CSV file and importing data from CSV files submitted by others, streamlining the exchange of knowledge and feedback with peers or mentors.

Overall, Applify empowers users to streamline the job application process, gain valuable insights, and collaborate effectively, ultimately enhancing their chances of landing their dream job.

List of Figures

Figure	Title	Page Number
3.2.1	Class Diagram	34
3.2.2	Description of relationship between classes of class diagram	36
3.2.3	Use Case Diagram	37
3.2.4	Sequence Diagram 1	38
3.2.5	Sequence Diagram 2	39
3.2.6	Sequence Diagram 3	40
3.2.7	Sequence Diagram 4	40
3.2.8	State Diagram 1	41
3.2.9	State Diagram 2	42
13.2.10	State Diagram 3	42
3.2.11	Context Diagram	43

3.2.12	Dataflow Diagram Level 1	44
3.3.1	UI Design (Pop up)	45
3.3.2	UI Design (My Application)	45
3.3.3	UI Design (Close Application)	46
3.3.4	UI Design (Summary and Insights)	46
3.3.5	UI Design (Imports and Exports Page)	47
4.1.1	Server side logs of user requests to flask server	48
4.1.2	Testing the scraping API on Postman	49
4.1.3	Organization of User Data in Firestore	49
4.2.1	Calling of Fetch Data API on Postman	50
4.2.2	Server side logs of import API	51
4.3.1	Main popup of chrome extension	51
4.3.2	My Applications page	52

4.3.3	My Applications page with change status	53
4.3.4	The summary and insights page	54
4.3.5	Import prompt	55
4.3.6	Export prompt	55

Abbreviations

1. **UI:** User Interface
2. **URL:** Uniform Resource Locator
3. **API:** Application Programming Interface
4. **AR:** Augmented Reality
5. **IoT:** Internet of Things
6. **CSV:** Comma-Separated Values
7. **SQL:** Structured Query Language
8. **HTTPS:** Hypertext Transfer Protocol Secure
9. **OS:** Operating System
10. **MB:** Megabyte
11. **GB:** Gigabyte
12. **UID:** User Identification
13. **JS:** JavaScript
14. **HTML:** Hyper Text Markup Language
15. **CSS:** Cascading Style Sheets
16. **VS Code:** Visual Studio Code

TABLE OF CONTENTS

	Page No.
Title page	i
Declaration	ii
Certificate	iii
Acknowledgement	iv
Executive Summary	v
List of Figures	vi
Abbreviations	ix
Chapter-1 INTRODUCTION	3
1.1 Problem definition	3
1.2 Project overview	4
1.3 Software specifications	6
1.3.1 Specific Requirements	6
1.4 Hardware specifications	7
1.4.1 Server Infrastructure	7
1.4.2 Remote Debugging Setup	7
1.4.3 Database Requirement	8
1.4.4 Networking	8
1.4.5 Client Side Requirement	8
Chapter-2 LITERATURE SURVEY	10
2.1 Existing system	10
2.2 Proposed system	11
2.3 Feasibility study	13
2.3.1 Technical Feasibility	13
2.3.2 Operational Feasibility	14
2.3.3 Legal and Compliance Feasibility	14
2.3.4 Financial Feasibility	14
2.3.5 Market Feasibility	15
2.3.6 Scalability and Future Development	15

Chapter-3 SYSTEM ANALYSIS & DESIGN	16
3.1 Requirement Specification	16
3.1.1 Requirement Elicitation	16
3.1.1.1 Normal Requirements	16
3.1.1.2 Expected Requirements	17
3.1.1.3 Exciting Requirements	18
3.1.2 Scope	19
3.1.2.1 Engaged Users	19
3.1.2.2 Returning Users	19
3.1.3 Definitions	20
3.1.4 Product Perspective	29
3.1.4.1 System Interface	29
3.1.4.2 User Interface	29
3.1.4.3 Hardware Interface	30
3.1.4.4 Software Interface	30
3.1.4.5 Communication Interface	30
3.1.5 Product Functions	30
3.1.6 User Characteristics	31
3.1.7 Constraints	32
3.1.8 Assumptions and Dependencies	32
3.1.8.1 Assumptions	32
3.1.8.2 Dependencies	33
3.2 System Design	34
3.3 Proposed UI	45
Chapter-4 RESULTS/OUTPUTS	48
4.1 Performance of scraping API	48
4.2 Performance of data retrieval, import, export APIs	50
4.3 Actual User Interface of Extension	51
Chapter-5 CONCLUSIONS/RECOMMENDATIONS	56
Chapter-6 REFERENCES	59
Personal Details	61

CHAPTER 1

INTRODUCTION

1.1 Problem Definition

Our project, titled "Applify: Track and Apply," addresses the significant challenge faced by job seekers in managing and tracking their job applications across various online platforms such as LinkedIn and Indeed.com. The proliferation of job opportunities across multiple websites has made it increasingly complex for individuals to keep track of their applications, leading to potential missed opportunities and organizational difficulties.

To mitigate these challenges, we are developing a Chrome extension that serves as a comprehensive solution for job application tracking and management. The core functionality of the extension revolves around providing users with an intuitive interface to track, monitor, and manage their job applications seamlessly.

The primary features of our extension include a popup interface accessible through the Chrome browser, where users can initiate actions, such as tracking a new job application, accessing their existing applications, and navigating within the extension. One of the key aspects of our system is the use of email IDs as unique identifiers for each user. This allows for personalized and synchronized data management, ensuring that users can access their application history across different devices.

When a user decides to track a job application, the extension interacts with a backend system hosted on a remote server. This backend facilitates the tracking process by extracting relevant information from the job application webpage. This information typically includes details like the job title, company name, application date, and status (e.g., open, closed, pending, etc.).

The data extracted from job application websites is securely stored. Each user has their data organized by their unique email ID, ensuring data isolation and privacy.

Our system also incorporates functionalities for data visualization and analysis. Users can view their application history in a tabular format. This table typically includes columns such as Company Name, Date of Application, Job Title, and Application Status. Users can interact with

this data, updating application statuses as needed (e.g., changing from 'Open' to 'Closed' or indicating progress through recruitment stages).

Additionally, we provide users with insights into their application statistics. Users can visualize their application statuses using interactive charts, such as pie charts. These visualizations help users understand their application progress, identify trends, and make informed decisions about their job search strategy.

Furthermore, our extension offers import/export functionalities for data management. Users can export their application data for external analysis or import data from files into their application database. This feature enhances data portability and enables users to integrate external data sources seamlessly.

In summary, "Applify: Track and Apply" aims to revolutionize the job application tracking experience by providing users with a powerful yet user-friendly tool. Our goal is to empower job seekers with efficient data management capabilities, insightful analytics, and a seamless application tracking process, ultimately enhancing their job search success.

1.2 Project Overview

"Applify: Track and Apply" project aims to revolutionize the job application tracking experience for users across multiple online platforms such as LinkedIn, Indeed.com, and others. The core component of the project is a Chrome extension designed to streamline the process of tracking and managing job applications.

The extension features a user-friendly interface comprising a popup accessible by clicking the extension icon in the Chrome browser. Within the popup, users have access to essential buttons like "Track," "My Applications," and "Exit," providing seamless navigation and functionality.

A key aspect of the system is the use of email IDs as unique identifiers for each user. This approach enables personalized tracking and synchronization of data across devices, enhancing user experience and convenience. The email IDs are retrieved from the user's Chrome browser session using the chrome.identity API and are stored securely in Firestore, a scalable NoSQL database provided by Firebase.

When a user tracks a job application by clicking the "Track" button on a job website, a request containing the current tab URL and user ID is sent to a Flask application hosted on a remote server. The Flask application utilizes Selenium to scrape relevant information from the job application webpage. This information includes details such as job title, company name, application date, and default status (e.g., open, closed, pending).

The scraped data is then stored in Firestore under the respective user's collection, organized by their unique email ID. Each job application is stored as a separate document within the collection, uniquely identified by the timestamp of the application. This structured approach ensures data integrity and accessibility for users.

To handle multiple simultaneous user requests efficiently, the system employs a queue mechanism built using Python's Redis RQ library. This queueing system helps manage the time-intensive task of scraping job application, ensuring smooth performance and responsiveness.

The extension also includes a "My Applications" page, where users can view their application history in a tabular format. This table typically displays columns such as Company Name, Date of Application, Job Title, and Application Status. Users can interact with this data, update application statuses, and manage their applications effectively.

Furthermore, the extension provides additional functionalities such as "Summary and Insights" for visualizing application statistics using interactive charts generated with Chart.js. Users can also import/export their application data in CSV format, facilitating data portability and analysis.

The project utilizes a combination of backend technologies including Flask, Redis, Selenium, and Firebase, along with frontend technologies like HTML, CSS, and JS. Version control is managed using Git and GitHub, and development is carried out using the VS Code IDE.

Overall, "Applify: Track and Apply" aims to enhance the job application tracking process, empower users with insightful analytics, and provide a seamless and efficient tool for managing job applications across various online platforms.

1.3 Software Specification

1.3.1 Specific Requirements (External Interfaces)

- **Chrome Extension Popup Interface:**

The extension popup interface includes buttons for tracking job applications ("Track"), accessing application history ("My Applications"), and exiting the extension ("Exit"). Clicking the "Track" button activates job application tracking on the current tab when visiting job websites like LinkedIn or Indeed.com. User authentication and unique identification are handled using the chrome.identity API, retrieving the user's email ID from their Chrome browser session. The email IDs serve as unique identifiers stored in Firestore, ensuring data synchronization across devices.

- **Flask API for Job Application Tracking:**

When the "Track" button is clicked, a request containing the current tab URL and user ID is sent to a Flask application hosted on a remote server. The Flask application utilizes Selenium WebDriver to scrape job application details from the URL, including job title, company name, application date. Scraped data is stored in Firestore under the user's collection, organized by email ID and timestamp of application.

- **API for My Applications Page:**

The "My Applications" page functionality is supported by an Express.js app running on a separate server. A request containing the user's ID is sent to the Express.js server, which retrieves the user's application data from Firestore. The application data is then displayed in a tabular format on the user's UI, showing columns such as Company Name, Date, Job Title, and Status.

- **Data Import/Export Functionality:**

Users have the option to export their job application details into a CSV file or import a CSV file sent by another user into their applications database. Export functionality is handled by a Flask API that retrieves data from Firestore based on the user's unique ID and forwards it to the user for download in CSV format.

Import functionality is achieved through an Express.js server running on a remote machine, parsing CSV files and storing data in Firestore.

- **Summary and Insights Page:**

The "Summary and Insights" page features a pie chart generated using Chart.js, displaying different job statuses (e.g., open, closed, resume selected, etc.) based on application data. Statistics for the pie chart are derived from the "My Applications" page, representing the percentage of each status in the user's application history.

- **Frontend Development and Version Control:**

Frontend development is done using HTML, CSS, and JavaScript for the Chrome extension's UI and functionality. Version control is managed using Git and GitHub, ensuring collaborative development and code management. Development is carried out in the VS Code IDE, providing a unified environment for coding and debugging.

1.4 Hardware Specification

The "Applify: Track and Apply" Chrome extension and associated backend services require specific hardware specifications to ensure optimal performance and reliability. An outline of the hardware requirements is mentioned as follows:

1.4.1 Server Infrastructure:

A robust server infrastructure is necessary to host the Flask and Express.js applications, along with the necessary APIs and databases. Minimum recommended specifications for the server include:

- **Processor:** Multi-core processor (e.g., Intel Core i5 or equivalent)
- **RAM:** At least 8GB DDR4 RAM for efficient handling of concurrent requests and data processing.
- **Storage:** SSD storage with ample capacity to store application data logs, and temporary files.
- **Network Connectivity:** High-speed internet connection for seamless communication with client-side extensions and external APIs. Connections with an upload and download speed of 10 MB/sec have shown to provide seamless UX.

1.4.2 Remote Debugging Setup:

For remote debugging using Selenium WebDriver, the server hosting the Flask application must have adequate resources to run Google Chrome in remote debugging mode efficiently.

RAM and CPU: Ensure sufficient RAM and processing power to handle multiple instances of Google Chrome and Selenium WebDriver simultaneously. Eight Giga bytes of RAM and any modern-day quad core processor is suitable for the task.

1.4.3 Database Requirements:

Firestore, the chosen database for storing user data, requires reliable storage and backup mechanisms. Considerations for database hosting to be kept in mind:

- **Backup and Recovery:** Regular backups of Firestore data to prevent data loss in case of system failures or accidental deletions must be implemented.
- **Scalability:** The database infrastructure should be scalable to accommodate increasing data volumes as the user base grows.

1.4.4 Networking:

Stable and secure networking infrastructure is essential for communication between the Chrome extension, remote servers, and Firestore database. Networking specifications are:

- **Secure Socket Layer (SSL) Certificates:** SSL certificates should be utilized for secure data transmission over the network, ensuring data privacy and integrity.
- **Firewalls and Security Measures:** Firewalls, intrusion detection systems, and other security measures must be implemented to protect against unauthorized access and cyber threats.
- **Bandwidth:** Sufficient bandwidth is necessary to handle incoming requests from multiple users and ensure responsive performance.

1.4.5 Client-Side Requirements:

While the bulk of processing occurs on the server side, users' devices running the Google Chrome extension should meet basic requirements for web browsing and extension functionality. Client-side requirements include:

- **Compatible Operating Systems:** Support for Chrome extensions on popular operating systems like Windows, macOS, and Linux.
- **Google Chrome Browser:** Users must have Google Chrome installed with support for extensions.
- **Adequate RAM and CPU:** Sufficient resources on the user's device to run Chrome and the extension without significant performance degradation.

By adhering to these hardware specifications, this project can deliver a reliable and efficient job application tracking experience to users while ensuring scalability and security in the backend infrastructure.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing System

- **Huntr - Job Search Tracker and Autofill:**

Huntr offers a comprehensive solution for job application management and automation. It allows users to collect, track, and manage job applications from various websites [1]. One of its key features is the autofill capability, which streamlines the application process by populating application forms with relevant data from the user's Huntr profile. This saves time and effort, especially when applying to multiple positions across different platforms. Additionally, Huntr provides a Kanban board for organizing applications based on their stages, adding a manual but effective way to track the progress of each application. Users can log notes, dates, tasks, descriptions, salaries, locations, company details, and more, creating a centralized hub for their job search activities. Huntr's compatibility with thousands of sites and support for top Applicant Tracking Systems (ATS) enhance its versatility and usability. However, it is important to note that Huntr's application tracking functionality is primarily applicable to LinkedIn, which may limit its scope for users applying through other job boards.

- **Jobscan - Job Search Tracker:**

Jobscan focuses on simplifying job application tracking and organization directly from job listings on popular job boards like Indeed, LinkedIn, and Glassdoor. Its Chrome extension facilitates the seamless saving of favourite jobs into the Jobscan Job Tracker, capturing essential details such as company information, job title, description, listing URL, and salary details automatically. The integration with job listings enables users to perform resume scans to assess their fit for a particular job, all within the job listing page. This eliminates the need to switch between tabs or manually enter job information, streamlining the process and enhancing productivity. With Jobscan Job Tracker, users can organize resumes, job descriptions, important interview dates, notes, and manage multiple job opportunities efficiently in one centralized platform. Its compatibility with major job boards broadens its reach and usefulness for job seekers across different platforms [2].

- **Teal - Job Tracker and Contacts/Companies Management**

Teal offers a comprehensive solution for job tracking, contacts management, and company research. It allows users to bookmark jobs from various job boards, track the status of each application, and access job descriptions conveniently. One notable feature is its ability to extract salary information from job descriptions, providing valuable insights for job seekers. In addition to job tracking, Teal enables users to manage contacts and companies effectively. Users can save connections and companies from LinkedIn, store entire LinkedIn profiles for research purposes, set follow-up reminders for interviews and communications, and use email templates for outreach and follow-ups. Teal's integration with LinkedIn and support for over 40 job boards enhance its versatility and usability for job seekers. Its features cater to different aspects of the job search process, from application tracking to networking and company research, making it a comprehensive tool for managing job applications and related activities [3].

Applify stands out from existing systems with its distinct features that offer enhanced functionality and user experience. Unlike other extensions like Huntr, Jobscan, and Teal, Applify introduces import and export functionalities, allowing users to seamlessly transfer job application data in CSV format. This feature streamlines data management and facilitates easy migration of information between systems, enhancing user convenience and workflow efficiency.

Furthermore, Applify offers a unique summary and insights feature through pie charts, providing users with visual representations of their job application statistics. This analytical tool enables users to gain valuable insights into their application progress, such as the distribution of job statuses like open, closed, resume selected, online test cleared, and more. This summary view not only enhances data visualization but also helps users track their application trends and make informed decisions regarding their job search strategy.

These distinguishing features set Applify apart by combining data import/export capabilities with insightful data visualization, creating a comprehensive and user-friendly solution for managing job applications effectively.

2.2 Proposed System

The proposed system is designed to streamline the process of job application tracking across multiple websites. This system leverages a Chrome extension as its primary interface, offering users intuitive functionalities for efficient job application management. The system architecture

encompasses both frontend and backend components, utilizing a combination of technologies to deliver a seamless user experience.

The frontend of the system, implemented with HTML, CSS, and JavaScript, provides a user-friendly interface through a Chrome extension popup. This popup houses essential features such as a "track button" to initiate application tracking on job websites, a "My applications" button to access application history, and an "exit button" for navigation and control. This intuitive design ensures ease of use and accessibility for users navigating through their job application data.

Behind the scenes, the backend infrastructure comprises a robust combination of Python-based technologies. The Flask framework serves as the foundation for the backend server, handling incoming requests and orchestrating data processing tasks. Redis, a powerful in-memory data structure store, is utilized with the rq library to manage queues effectively, ensuring efficient handling of concurrent user requests and resource utilization.

Selenium WebDriver plays a pivotal role in the system's functionality, enabling dynamic web scraping capabilities. When a user triggers the "track button" on a job website, a request containing the current tab URL and user ID is dispatched to a remote Flask application. This application utilizes Selenium WebDriver in remote debugging mode to scrape relevant job application information, including job title, company name, date of application, and status indicators such as "open," "closed," or specific stages in the application process (e.g., "resume selected," "HR round cleared").

The extracted data is then structured and stored in Firestore, Google's flexible, scalable NoSQL database. Firestore organizes data into collections, with each user having a dedicated collection identified by their unique email ID. Within these collections, individual documents store job application details, timestamped for chronological organization. This structured approach ensures data integrity and facilitates efficient retrieval and analysis of application history.

Additionally, the system incorporates functionalities for data visualization and management. The "My applications" page, powered by an Express.js app, retrieves user-specific data from Firestore and presents it in a tabular format. Users can view detailed information about their job applications, including company name, date, job title, and current status. The interactive interface allows users to modify application statuses, delete entries, and track progress seamlessly.

The system's analytics and insights feature enhances user experience by offering a comprehensive view of application statistics. The "Summary and insights" page utilizes Chart.js, a JavaScript charting library, to generate dynamic pie charts illustrating application statuses. These visual representations provide valuable insights into application distribution across different stages, aiding users in assessing their job search progress and strategizing effectively.

Furthermore, the system facilitates data import and export functionalities to enhance flexibility and data portability. Users can export their job application data, including status details, into CSV format for external analysis or backup purposes. Conversely, they can import CSV files containing application data, seamlessly integrating new information into their existing application history.

The import and export functionalities are supported by dedicated APIs implemented using Flask and Express.js, ensuring smooth data transfer between the system and external sources. These APIs handle data formatting, parsing, and integration with Firestore, maintaining data consistency and accessibility across import/export operations.

Overall, the "Applify: Track and Apply" system presents a comprehensive solution for job application management, combining intuitive user interfaces, robust backend infrastructure, dynamic data processing capabilities, and seamless data integration functionalities. By leveraging a combination of technologies and best practices, the system aims to enhance user productivity, streamline job search workflows, and provide valuable insights for informed decision-making in the application process.

2.3 Feasibility Study

Several aspects need consideration to assess the project's viability:

2.3.1 Technical Feasibility:

- **Chrome Extension Development:** It's feasible as Chrome provides robust APIs for extension development, including identity management for user identification.
- **Backend Infrastructure:** Using Flask, Redis, and Firebase for backend functionalities is feasible, offering scalability and real-time data synchronization.
- **API Integration:** Integrating Flask and Express.js APIs for data exchange and processing is technically feasible.

- **Data Storage:** Firestore is suitable for storing job application data due to its scalability and real-time updates.

2.3.2 Operational Feasibility:

- **User Experience:** The interface with track, applications, and import/export features is user-friendly and operationally feasible.
- **Cross-Device Sync:** Using email IDs for user identification ensures data synchronization across devices, enhancing operational feasibility.
- **API Performance:** Proper handling of API requests and queues ensures operational efficiency, especially during simultaneous user interactions.

2.3.3 Legal and Compliance Feasibility:

Compliance with data privacy regulations is crucial, especially regarding user email IDs and application data storage. Ensuring data encryption and secure transmission is essential. It is crucial to address the legal aspects of web scraping. Obtaining legal permission from websites is mandatory before scraping their data. This requirement stems from the fact that many websites have policies explicitly stating that scraping their content without permission is against their terms of service and may be illegal.

Before proceeding with data scraping, it's essential to review the terms of use, robots.txt file, and any other relevant legal documentation provided by each website. Violating these terms can lead to legal consequences such as legal action, fines, or being blocked from accessing the website.

To ensure compliance and avoid legal issues, the project must implement mechanisms to obtain explicit permission from each website before scraping its content. This can include obtaining API keys or other authorization methods provided by the websites for accessing and retrieving data in a permissible manner. Additionally, the project should incorporate mechanisms to respect the website's rate limits and ensure that the scraping process does not disrupt or overload the website's servers.

2.3.4 Financial Feasibility:

- **Cost of Services:** Consideration of costs associated with hosting servers, using Firebase services, and potential scalability needs is essential. For small scale usage (less than 1000 user requests per day) using firebase does not incur any cost. However, as the number of users increase, migration to firebase's paid plans becomes a necessity.

Furthermore, a single server can only handle a limited number of user requests at a time before it starts to increase waiting time of users. Further research is required to find the number of concurrent users per server, before which, the need to increase backend hardware arises.

- **Development Costs:** Since this was an academic project, there haven't been any development costs for the same, exclusive of the basic hardware required to develop this software.

2.3.5 Market Feasibility:

- **User Demand and Competitive Analysis:** Analyzing the demand for job application tracking tools and similar Chrome extensions can validate market feasibility. Understanding competitors and unique value propositions can help gauge market potential.

2.3.6 Scalability and Future Development:

The architecture's scalability with Firestore, Redis, and Flask allows for handling increased user loads and data volumes. Considering future enhancements like additional features, integrations with more job portals, and user feedback incorporation ensures long-term feasibility.

CHAPTER 3

SYSTEM ANALYSIS & DESIGN

3.1 Requirement Specification

3.1.1 Requirement Elicitation

3.1.1.1 Normal Requirements (Essential):

- **User Authentication:** The system must authenticate users based on their email IDs retrieved from their Chrome browser session using the chrome.identity API. This authentication process is essential for identifying and synchronizing user data across devices.
- **Popup Interface:** The extension must feature a popup interface accessible via the extension icon in Chrome. The popup should include buttons for tracking job applications, accessing application history, and exiting the extension.
- **Job Application Tracking:** Upon clicking the track button on a job website, the extension should activate job application tracking for that website. This functionality is crucial for monitoring the status and details of job applications.
- **Data Storage:** The system must store scraped job application data in Firestore, utilizing collections and documents to organize data per user. Each job application entry should include details like company name, date of application, job title, and status.
- **Status Updates:** Users should be able to change the status of each job application (e.g., open, closed, pending) through dropdown menus in the My Applications interface. These updates must reflect in Firestore to maintain the latest application status.
- **Application History Display:** The My Applications page should display user-specific job application data in a tabular format, including columns for company name, date, job title, and status. This view is crucial for users to manage and track their application history.
- **Import/Export Functionality:** The system must support importing and exporting job application data in CSV format. Users should have options to export all their application details or import data from CSV files sent by others into their applications database.

- **Data Synchronization:** The extension should synchronize user data across devices to ensure a seamless experience regardless of the device used. This synchronization is fundamental for maintaining a consistent view of job application history.
- **API Integration:** The system must integrate with Flask and Express.js APIs running on remote servers for functionalities like data storage, import/export operations, and handling user requests for application history.
- **Security and Legal Compliance:** It is crucial to ensure that the extension complies with legal requirements, including obtaining explicit permission from websites before scraping their data. Adhering to legal standards and ensuring data security are essential aspects of the system.
- **Frontend and Backend Technologies:** The frontend must be developed using HTML, CSS, and JavaScript, while the backend involves technologies like Flask, Redis, Selenium, Firebase, and Express.js. Git and GitHub should be used for version control, and development should be done in an integrated development environment like VS Code.

3.1.1.2 Expected Requirements (Important):

- **User Authentication and Data Security:** The extension shall ensure secure authentication by retrieving the user's email ID from their Chrome browser session. User data, including email IDs and job application details, shall be securely stored in Firestore with appropriate access controls.
- **Real-time Data Synchronization:** Data synchronization across devices shall be implemented to ensure that users can access their job applications and updates from any device seamlessly. Changes made by the user, such as updating job statuses, deleting jobs, or importing/exporting data, shall be reflected in real-time across devices.
- **Efficient Queue Management:** The queue system for handling multiple simultaneous requests shall be optimized to ensure efficient processing of scraping tasks and API requests. Default values and error handling mechanisms, such as using "Not Available" for missing scraped elements, shall be implemented to maintain system reliability.
- **Responsive User Interface (UI):** The UI of the extension, including the popup interface and My Applications page, shall be responsive and intuitive for ease of use. Dropdown menus, side panels, and chart displays shall provide a user-friendly experience for viewing and managing job applications.

- **Chart Visualization and Insights:** The pie chart on the Summary and Insights page shall accurately represent the distribution of job application statuses, providing users with visual insights into their application progress. Statistics for generating the pie chart shall be dynamically updated based on the user's application data and reflect changes made by the user.
- **API Integration and Remote Server Management:** Integration of Flask and Express.js APIs shall be seamless, ensuring smooth export/import functionalities and data retrieval from Firestore. The single remote server hosting all APIs shall be efficiently managed to handle user requests, maintain data integrity, and ensure system reliability.

These above listed expected requirements are crucial for the successful implementation and functionality this project, focusing on user experience, data security, real-time synchronization, and efficient system management.

3.1.1.3 Exciting Requirements (Innovative and Differentiating):

- **Dynamic Job Status Updates:** Implementing real-time updates for job statuses, where users can receive notifications or alerts when there are changes in their application statuses, such as moving from "Open" to "HR Round Cleared."
- **AI-Powered Insights and Recommendations:** Integrating machine learning algorithms to analyze user's application history and provide personalized insights, such as recommended job opportunities based on their skills and preferences.
- **Gamification and Engagement Features:** Incorporating gamification elements like badges, achievements, or progress bars to motivate users in their job search journey and encourage consistent use of the extension.
- **Interactive Visualizations and Data Exploration:** Enhancing the Summary and Insights page with interactive visualizations, allowing users to drill down into specific data points and gain deeper insights into their application trends.
- **Collaborative Job Search Tools:** Enabling collaboration features where users can share job listings, insights, and application tips with their network, fostering a community-driven approach to job hunting.
- **Smart Resume Builder and Optimization:** Developing a smart resume builder within the extension that analyzes job descriptions, suggests optimizations, and generates tailored resumes for different applications.

- **Integration with Career Development Resources:** Integrating with online learning platforms or career development resources to provide users with relevant courses, certifications, and resources to enhance their skills and marketability.
- **Voice and Natural Language Processing (NLP):** Implementing voice commands or NLP capabilities for hands-free interaction with the extension, allowing users to perform tasks and access information using natural language.
- **Blockchain-based Credential Verification:** Exploring blockchain technology for secure credential verification, allowing users to validate their qualifications, certifications, and achievements directly within the extension.
- **Augmented Reality (AR) Job Search Experience:** Experimenting with AR features for an immersive job search experience, such as visualizing job locations, company cultures, or virtual career fairs within the extension.

3.1.2 Scope

3.1.2.1 Engaged Users:

- **Job Seekers:** These are the core users who actively use the extension to track and manage their job applications. They are highly engaged as they regularly interact with the extension's features, such as tracking new applications, updating application statuses, and reviewing their application history.
- **Regular Data Analyzers:** Users interested in analyzing their job application trends and statistics fall into this category. They frequently access the insights and statistics section of the extension to gain valuable information about their application progress and success rates.
- **Power Users:** Advanced users who extensively utilize the import/export functionalities for data backup, sharing with colleagues, or integrating their application data with other tools or platforms.
- **Remote Workers:** Individuals working remotely may use the app for accessing print services without the need for physical access to a printer. This category of users finds value in remote document printing services.

3.1.2.2 Returning Users:

- **Job Seekers on Ongoing Job Hunts:** Users who are actively searching for jobs and consistently adding new applications to track. They return to the extension frequently to update application statuses, add new applications, and review their progress.

- **Data-Driven Professionals:** Users who rely on data analysis for decision-making in their job search. They return to the extension to review past insights, track changes in application trends, and adjust their job application strategies based on the data provided.
- **Collaborative Users:** Users who engage with the import/export functionalities to collaborate with others or maintain data continuity across devices. They return to the extension to import new data, export updated information, or share data with colleagues or peers.

The engagement level of users can vary based on their individual needs, job search intensity, and usage patterns. Engaged users are those who actively utilize the core features of the extension and derive value from its functionalities, while returning users are those who revisit the extension over time for ongoing job search management, data analysis, or collaborative purposes.

3.1.3 Definition

To ensure clarity and understanding in the project report, a list of definitions, abbreviations, and acronyms is provided. This list helps readers navigate the document and comprehend technical terms and references.

3.1.3.1 Applify -Track and Apply

Applify is like a helper for people looking for jobs. It's a tool that works with Google Chrome [4], making it easier to keep track of all the job applications you've sent out. It gathers important details like where you applied, when, and what's happening with each application. Instead of having to remember or write down everything, it puts it all in one place. You can see which jobs you've applied to, which ones you're waiting to hear back from, and any updates on your applications. It's like having a personal organizer for your job search, helping you stay on top of things and making the process less stressful.

3.1.3.2 UID

A UID, or Unique Identifier, serves as a distinctive label for individuals or entities within various systems, often associated with an email address for online identification purposes. In this context, an email-based UID is a specific identifier generated from an individual's email

address, distinguishing them from others in digital platforms and service [5]. When someone signs up for an online account using their email address, the system typically assigns a UID to uniquely identify them. This UID becomes their digital fingerprint within that system, allowing for personalized interactions, secure authentication, and targeted communication. The email-based UID acts as a bridge between the user's email account and the platform's database, enabling seamless integration of services and facilitating efficient communication.

3.1.3.3 API

An API, or Application Programming Interface, is like a digital middleman that allows different software applications to communicate and interact with each other. It defines the methods and protocols for how different software components should interact, essentially serving as a bridge that enables seamless integration and data exchange between disparate systems. [6] At its core, an API specifies the rules and conventions for how software components should interact. It provides a set of rules and protocols that govern how one piece of software can request and receive information or perform actions from another piece of software, regardless of the underlying technology or programming languages involved. It comes in various forms, including web APIs, library APIs, and operating system APIs, each designed for specific purposes. Web APIs, for example, are commonly used for enabling communication between web-based applications, allowing developers to access services and data over the internet. Library APIs, on the other hand, provide predefined functions and procedures that developers can use to perform specific tasks within their applications. One of the key benefits of them is their ability to promote interoperability and modularity in software development [7]. By providing standardized interfaces for communication, APIs enable developers to build upon existing software components without needing to understand their internal workings. This promotes code reusability, accelerates development cycles, and fosters innovation by allowing developers to focus on building new features rather than reinventing the wheel. Moreover, they facilitate the creation of ecosystems where developers can leverage third-party services, data, and functionality to enhance their own applications. For example, social media platforms offer APIs that allow developers to integrate features like user authentication, posting updates, or fetching user data into their own applications, enriching the user experience and expanding the platform's reach. They play a vital role in modern software development by enabling seamless communication and integration between diverse software systems. They empower developers to build more powerful and feature-rich applications by leveraging the capabilities of existing

services and platforms, ultimately driving innovation, and enhancing user experiences across the digital landscape [8].

3.1.3.4 URL

A URL, or Uniform Resource Locator, is a fundamental component of the internet that serves as an address for identifying and locating resources such as web pages, documents, images, and other files. It functions as a standardized format for specifying the location of resources on the World Wide Web. At its core, a URL consists of several essential components, each serving a specific purpose in directing users to the desired resource [9]. The primary components include the scheme, domain name, path, and optional parameters. The scheme, which is typically indicated at the beginning of the URL, specifies the protocol or method used to access the resource. Common schemes include "http://" for Hypertext Transfer Protocol (HTTP) and "https://" for secure HTTP. Other schemes like "ftp://" for File Transfer Protocol or "mailto:" for email addresses may also be used. The path, which comes after the domain name, specifies the specific location or directory within the server where the resource is located. It may include directories, subdirectories, and filenames, providing a precise pathway to the desired resource on the server. Together, these components form a coherent and structured address that enables users to access resources on the internet efficiently. URLs are essential for navigating the vast network of interconnected web pages and resources, providing a standardized method for accessing information and services online. They play a foundational role in the functioning of the World Wide Web, facilitating seamless communication and interaction across diverse digital platforms and services.

3.1.3.5 Flask

Flask is a lightweight and flexible web framework for Python designed to build web applications quickly and with minimal boilerplate code. It follows the WSGI (Web Server Gateway Interface) specification, making it compatible with various web servers and platforms. At its core, it provides a simple yet powerful structure for developing web applications, offering features like URL routing, request handling, and template rendering. One of its key strengths is its simplicity and ease of use, allowing developers to get started with building web applications without the overhead of complex configurations. It embraces the concept of "micro-framework," meaning it focuses on providing the essential components for web development while allowing

developers the freedom to add additional functionality as needed. This minimalist approach makes it particularly suitable for small to medium-sized projects or prototypes where simplicity and flexibility are prioritized. One of its notable features is its built-in development server, which enables developers to test and debug their applications locally without the need for external server configurations. This feature accelerates the development process by providing a convenient environment for rapid iteration and experimentation [10].

3.1.3.6 Selenium

Selenium is an open-source automation tool widely used for automating web browsers, offering a suite of tools and libraries for various programming languages like Python, Java, and JavaScript. It operates by controlling web browsers programmatically, simulating user interactions such as clicking buttons, entering text, and navigating through web pages. The WebDriver API, a key component of Selenium, acts as a bridge between Selenium and web browsers, providing commands and methods for interacting with web elements on a page. This functionality enables developers to automate complex browser actions with ease, making Selenium a preferred choice for automated testing of web applications across different browsers and platforms. Furthermore, Selenium supports cross-browser testing, allowing developers to write test scripts that run seamlessly across multiple browsers to ensure consistent behaviour and compatibility of web applications. Additionally, Selenium can be utilized for web scraping tasks, extracting data from web pages by programmatically navigating through them and extracting relevant information. Its flexibility, extensibility, and robustness, coupled with an active community and extensive documentation, have contributed to Selenium's popularity and widespread adoption in the software development ecosystem [11].

3.1.3.7 Firestore

Firestore is a versatile, cloud-based database solution developed by Google as part of the Firebase platform, offering developers a scalable and real-time database for building web and mobile applications. It operates as a NoSQL database, storing data in flexible, hierarchical structures of collections and documents. Each document holds key-value pairs of data, organized within collections, providing developers with a powerful and intuitive data model for organizing and querying data. One of its's standout features is its real-time data synchronization capability, enabling seamless updates across multiple clients and platforms in real-time. This

makes it particularly suitable for applications requiring instant data updates, such as chat applications, collaborative tools, and live analytics dashboards. It also provides robust offline support, allowing applications to continue functioning even when disconnected from the network. Changes made while offline are automatically synchronized with the database once connectivity is restored, ensuring data consistency and reliability. Additionally, it offers advanced querying capabilities, enabling developers to filter, sort, and query data efficiently based on specific criteria, empowering them to retrieve and manipulate data according to their application's needs. Security is a top priority in it, with features such as authentication, access control rules, and data validation, enabling developers to implement granular permissions and restrictions to safeguard sensitive data and comply with regulatory requirements. Furthermore, it seamlessly integrates with other Google Cloud services, Firebase features, and third-party platforms, providing developers with a comprehensive ecosystem for building and deploying modern applications. Its serverless architecture eliminates the need for infrastructure management, allowing developers to focus on application logic and functionality rather than infrastructure maintenance. It is a powerful and flexible cloud database solution that simplifies data management and enables developers to build real-time applications with ease. Its real-time synchronization, offline support, advanced querying capabilities, robust security features, and seamless integration options make it a preferred choice for developers seeking to create scalable and responsive applications for various use cases [12].

3.1.3.8 Redis

Redis is a versatile and high-performance open-source, in-memory data store widely used for its speed, simplicity, and flexibility. As a key-value store, Redis excels in storing and retrieving data with lightning-fast read and write operations, making it ideal for applications requiring rapid data access. Its support for various data structures, including strings, lists, sets, hashes, and sorted sets, allows developers to efficiently store and manipulate diverse types of data within a single Redis instance, catering to a wide range of use cases such as caching, session management, real-time analytics, and messaging. One of its standout features is its ability to persist data to disk periodically or asynchronously, ensuring data durability and resilience against system failures or restarts. Additionally, Redis offers high availability and clustering capabilities, enabling replication and distribution of data across multiple nodes to achieve scalability and fault tolerance in distributed environments. Its pub/sub messaging feature facilitates real-time communication between clients through channels and subscribers, making it a popular choice for building real-time applications, message queues, and event-driven

architectures. Its rich ecosystem of client libraries and plugins for various programming languages and frameworks simplifies integration with existing applications, while its active community and extensive documentation provide resources and support for developers of all skill levels [13].

3.1.3.9 Express.js

Express.js, commonly referred to as Express, is a lightweight and flexible web application framework for Node.js, designed to simplify the process of building web applications and APIs. It provides a minimalist yet powerful set of features and tools for creating robust and scalable web servers using JavaScript, making it a popular choice among developers for building server-side applications. It simplifies the handling of HTTP requests and responses, routing, middleware integration, and templating, allowing developers to focus on building the core functionality of their applications without getting bogged down by boilerplate code [14]. Its minimalist design and unopinionated nature provide developers with the freedom to choose the tools and libraries that best suit their project requirements. It follows a middleware-based architecture, where incoming HTTP requests pass through a series of middleware functions before reaching the final request handler. This modular approach enables developers to modularize their application logic, handle authentication, error handling, and perform various pre-processing tasks easily. It is highly extensible, with a vast ecosystem of third-party middleware and plugins available through the npm package manager. These middleware and plugins extend the functionality of Express, enabling developers to add features such as session management, authentication, logging, and database integration seamlessly [15].

3.1.3.10 CSV

CSV stands for Comma-Separated Values, a simple and widely used file format for storing tabular data in plain text. In a CSV file, each line represents a row of data, and fields within each row are separated by commas or other delimiters, such as semicolons or tabs. This format allows for the organization and storage of structured data in a human-readable and easily editable form. They are commonly used for exchanging data between different applications, systems, and platforms due to their simplicity and compatibility. They can be opened and edited using basic text editors, spreadsheet software like Microsoft Excel or Google Sheets, and programming languages such as Python and R. This universality makes these files a popular

choice for data interchange in various industries and domains [16]. One of the key advantages of these files is their lightweight nature, as they do not include complex formatting or metadata. This simplicity results in smaller file sizes compared to proprietary file formats like Excel (.xlsx) or databases like SQL, making them efficient for storing and transferring large volumes of data. They are also highly versatile, supporting different data types such as strings, numbers, dates, and Booleans within the same file. Additionally, CSV files can accommodate hierarchical data structures by using delimiter characters to represent nested levels of data, although this approach may require additional processing during data parsing. Despite their simplicity and widespread usage, these files have some limitations. They lack built-in support for advanced features like data validation, formulas, and formatting, which are commonly found in spreadsheet applications. Furthermore, they may encounter issues with data integrity if fields contain special characters or newline characters, necessitating careful handling and encoding during data manipulation and transmission. Despite these limitations, CSV remains a popular and practical choice for storing and exchanging tabular data due to its simplicity, versatility, and widespread support [17].

3.1.3.11 UI

UI, or User Interface, refers to the visual elements and interactive components of a software application or system that users interact with to perform tasks, access information, and navigate the interface. The primary goal of UI design is to create an intuitive and user-friendly interface that enhances the overall user experience (UX) by providing an efficient and enjoyable interaction with the software. It encompasses various elements such as buttons, menus, input fields, icons, and visual layouts, as well as the overall design aesthetics, color schemes, typography, and branding elements. Effective UI design considers factors like usability, accessibility, consistency, and responsiveness to ensure that the interface meets the needs and expectations of its target users [18]. UI design often involves the use of design principles and methodologies such as user-centered design (UCD), which focuses on understanding the needs and preferences of users through research and iterative testing. Prototyping and wireframing tools are commonly used in UI design to create mockups and prototypes of the interface, allowing designers to visualize and refine the layout and functionality before implementation. In addition to visual design, it also encompasses interaction design (IxD), which involves defining how users interact with the interface through gestures, clicks, swipes, and other input methods. This includes designing logical navigation flows, feedback mechanisms, error handling, and animations to guide users and provide a seamless and engaging interaction [19].

Overall, UI plays a crucial role in shaping the user experience of software applications and systems, influencing factors such as usability, satisfaction, productivity, and brand perception. By focusing on user needs and preferences and incorporating principles of good design, UI design aims to create interfaces that are intuitive, aesthetically pleasing, and effective in facilitating user tasks and goals.

3.1.3.12 JS

JavaScript, often abbreviated as JS, is a versatile and widely used programming language primarily employed for creating dynamic and interactive content on the web. Initially developed by Brendan Eich in 1995, JavaScript has evolved into one of the most essential technologies for web development, enabling developers to add functionality, interactivity, and behavior to web pages. One of its key features is its ability to run directly within web browsers, making it accessible to developers and users without the need for additional software or plugins. As a client-side scripting language, JavaScript interacts with the Document Object Model (DOM), allowing developers to manipulate HTML elements, handle user interactions, and dynamically update the content and appearance of web pages in response to user actions or events. It is also used extensively for server-side development through platforms like Node.js, which allows developers to build scalable and efficient web servers using JavaScript. This versatility enables full-stack developers to work seamlessly across both client-side and server-side environments, leveraging JavaScript's capabilities to create end-to-end web applications. Furthermore, it has a rich ecosystem of libraries, frameworks, and tools that streamline development and extend its capabilities. Popular JavaScript frameworks like React.js, AngularJS, and Vue.js provide developers with powerful tools for building single-page applications (SPAs), while libraries like jQuery simplify DOM manipulation and event handling. Its popularity and widespread adoption have led to its use beyond web development, with applications ranging from mobile app development (using frameworks like React Native and Ionic) to game development (using libraries like Phaser.js) and even desktop application development (using tools like Electron). Overall, it is a versatile and essential programming language for web development, offering capabilities for both client-side and server-side development. Its ability to create dynamic and interactive web content, coupled with its extensive ecosystem and community support, makes JavaScript a foundational technology for modern web development.

3.1.3.13 HTML

HTML, which stands for Hypertext Markup Language, is the standard language used to create and structure web pages on the World Wide Web. Developed by Tim Berners-Lee and his team in the early 1990s, HTML serves as the foundation of web development, providing a markup system for defining the structure and content of web documents [20]. At its core, it consists of a series of elements or tags, each representing a specific type of content or structure within a web page. These elements are enclosed in angle brackets (<>) and typically come in pairs: an opening tag and a closing tag, with content nested in between. For example, the <p> tag denotes a paragraph, <h1> to <h6> tags represent headings of varying levels, and <a> tags create hyperlinks. HTML elements are organized into a hierarchical structure known as the Document Object Model (DOM), which represents the logical structure of a web page as a tree-like structure of nodes. This structure enables web browsers to interpret and render HTML documents, displaying them as interactive and visually appealing web pages. In addition to defining the structure and content of web pages, it also supports attributes, which provide additional information or properties to elements. Attributes modify the behavior or appearance of elements and can include attributes like id, class, src, href, and style. It is complemented by other web technologies such as Cascading Style Sheets (CSS) and JavaScript, which enhance its capabilities by providing styling and interactivity, respectively. Together, these technologies form the cornerstone of web development, enabling developers to create rich, engaging, and dynamic web experiences for users across different devices and platforms. Overall, it plays a foundational role in web development, serving as the backbone of web pages and providing a standardized markup language for structuring and presenting content on the web. Its simplicity, versatility, and widespread adoption make it an essential tool for anyone involved in creating or maintaining web content [21].

3.1.3.14 CSS

CSS, or Cascading Style Sheets, is a fundamental technology used to style and visually format web pages created with HTML. Developed as a separate specification from HTML by the World Wide Web Consortium (W3C), CSS provides a powerful mechanism for controlling the presentation of web content, including elements such as fonts, colors, layouts, and spacing [22]. At its core, it works by associating style rules with HTML elements, specifying how those elements should be displayed or formatted in a web browser. Style rules consist of selectors, which target specific HTML elements, and declarations, which define the desired appearance

or behavior of those elements. For example, a CSS rule targeting all paragraphs (<p> elements) might specify a particular font size, color, and margin. It offers a wide range of styling capabilities, including the ability to apply colors, backgrounds, borders, and shadows to HTML elements, as well as to control text formatting, alignment, and spacing. CSS also supports responsive design techniques, allowing developers to create layouts that adapt and respond to different screen sizes and devices, enhancing the usability and accessibility of web pages across desktops, tablets, and smartphones. One of its most powerful features is its ability to cascade and inherit styles, which allows developers to create consistent and maintainable style sheets. CSS rules can be applied in a hierarchical manner, with styles cascading from parent elements to their children, and inherited styles propagating down the document tree. This enables developers to define global styles that apply to entire documents or specific sections, while also allowing for fine-grained control over individual elements. In addition to traditional CSS, modern web development often involves the use of preprocessor languages like Sass and LESS, which extend CSS with features such as variables, mixins, and nesting, providing developers with more flexibility and efficiency in writing and managing stylesheets. Overall, it plays a critical role in web development, enabling developers to create visually appealing, responsive, and user-friendly web experiences by controlling the presentation and layout of HTML content in web browsers. Its versatility, flexibility, and widespread adoption make it an essential tool for anyone involved in designing and building websites [23].

3.1.3 Product Perspective

3.1.4.1 System Interfaces

- **Chrome Extension Popup:** Interface for users to interact with the extension, including buttons like "track," "My applications," and "exit."
- **Firebase Firestore:** Stores user data, including job applications and their statuses, organized by user email IDs.
- **Remote Servers:** Host Flask and Express.js APIs for handling requests related to job tracking, data storage, and data retrieval.
- **Redis RQ Queue:** Manages API requests in a queue to handle multiple simultaneous user requests efficiently.

3.1.4.2 User Interfaces

- **Chrome Extension Popup:** Provides a user-friendly interface with buttons for tracking applications, accessing application history, and navigation options.
- **My Applications Page:** Displays job application data in a tabular format, allowing users to view, update application statuses, and delete applications.
- **Summary and Insights Page:** Presents a pie chart visualizing application status to provide users with insights into their application progress.
- **Import/Export Page:** Allows users to import/export job application data in CSV format for backup and sharing purposes.

3.1.4.3 Hardware Interfaces

- **Chrome Browser:** Hosts the Chrome extension and provides the interface for users to interact with the extension's features.

3.1.4.4 Software Interfaces

- **Chrome Identity API:** Retrieves user email IDs from the Chrome browser session for user identification.
- **Flask API:** Handles requests related to job application tracking, scraping, and data storage in Firestore.
- **Express.js API:** Manages requests for retrieving user data from Firestore and handling import/export functionalities.
- **Firebase Firestore:** Stores user data in a structured format based on user email IDs and job application details.
- **Chart.js Library:** Used for creating interactive charts and graphs for data visualization.

3.1.4.5 Communication Interfaces

- **API Requests:** Communication between the Chrome extension and remote servers (Flask and Express.js) via HTTP requests to perform various actions like tracking applications, retrieving data, and managing import/export functionalities.
- **Firebase:** Serves as the communication interface for storing and retrieving user data, ensuring synchronization across devices for seamless user experience.

3.1.5 Product Functions

3.1.5.1 Job Application Tracking

The extension shall provide a "Track" button within the Chrome extension popup. Clicking the "Track" button while on a job posting webpage initiates application tracking for that specific webpage. The extension shall utilize Selenium webdriver to scrape relevant job application details from the webpage. Scraped details shall include, but are not limited to:

- Job title
- Company name
- Date of application

Scraped data shall be associated with the unique user identifier (email address retrieved from Chrome session) and stored securely in Firestore.

3.1.5.2 Application Management

The extension shall offer a "My Applications" button within the popup window. Clicking "My Applications" opens a user interface displaying a list of all tracked applications. The application list shall display the following information for each entry: Company Name (obtained from scraped data), Date Applied (obtained from scraped data), Job Title (obtained from scraped data), Application Status (with options to select from a predefined list - Open, Closed, Pending, Resume Selected, Online Assessment Cleared, HR Round Cleared, Technical Round Cleared). The user interface shall allow updating the application status from the predefined list for each application entry. Updates to application status shall be reflected in the user's Firestore database.

3.1.5.3 Data export and import:

The application shall provide a functionality to export all user application data as a CSV file. Initiating the export function shall trigger a separate Flask API call with the user's unique identifier. The API shall retrieve the user's data from Firestore and format it into a downloadable CSV file. The user shall be directed to their system's download menu to access the exported CSV file. The application shall allow importing application data from a user-uploaded CSV file. Upon upload selection, the application triggers an Express.js API call, sending the user's identifier and the CSV data. The API shall parse the uploaded CSV file, ensuring data integrity and format compliance. Validated data from the CSV file shall be stored in the user's Firestore collection, appended to existing data. Imported application data shall be reflected in the "My Applications" user interface.

3.1.6 User characteristics

- **Tech-Savviness:** Users should be comfortable using web browsers and Chrome extensions. They should understand basic concepts like opening tabs, clicking buttons, and navigating web interfaces.
- **Job Seekers:** The target users are individuals actively looking for jobs or managing their job applications. They should have a need to track and organize their job search activities.
- **Chrome Users:** Since the extension is built for Chrome, users should be using Google Chrome as their web browser.
- **Email ID Ownership:** Users must have an email ID associated with their Chrome browser, as the extension uniquely identifies users based on their email IDs.
- **Basic Data Management Understanding:** Users should be familiar with basic data management concepts like importing/exporting data in CSV format and understanding different job application statuses.
- **Interest in Data Insights:** Users interested in gaining insights into their job application activities, such as viewing pie charts and statistics based on their application statuses.
- **Comfort with Online Tools:** Users should be comfortable with using online tools and services, as the extension interacts with remote servers for data storage, synchronization, and API calls.
- **Attention to Privacy:** Users should understand and appreciate the privacy implications of using the extension, including data synchronization and identification based on email IDs.

3.1.7 Constraints

- **Time constraints:** Thirteen weeks
- **Cost Constraints:** Running the APIs and backend services on a single remote server incurs hosting costs, which should be considered based on the server specifications, data transfer rates, and storage requirements. If any third-party APIs or services are used (e.g., Firebase), their usage may have associated costs based on usage tiers or API calls.
- **Resource Constraints:** The extension relies on system resources such as CPU and memory, especially during tasks like web scraping and data processing. Ensuring optimal resource management is essential for performance. Handling multiple simultaneous user requests, especially during web scraping, can strain network bandwidth. Efficient queue management

and data transfer strategies are important. Developing and maintaining the extension, backend APIs, and server infrastructure require continuous time and effort from developers, impacting project timelines and resource allocation. As the user base grows, scalability concerns may arise in terms of server capacity, database performance, and overall system responsiveness. Planning for scalability is crucial to accommodate increasing user demands.

3.1.8 Assumptions and dependencies

3.1.8.1 Assumptions

- **Users are comfortable with Chrome extensions:** This project relies on users installing and using a Chrome extension. We assume users have a basic understanding of how Chrome extensions work and are comfortable adding them to their browser.
- **Job application websites have consistent structures:** The scraping functionality relies on the assumption that job application websites have a consistent structure across different companies and platforms. This allows the scraper to identify and extract relevant data points like job title, company name, etc.
- **Users have unique email addresses:** The project uses email addresses as unique identifiers for users. We assume each user has a unique email address associated with their Chrome browser session.
- **Data privacy regulations are met:** The project collects and stores user data (job application details). We assume all data privacy regulations are met, and users are informed about data collection practices.

3.1.8.2 Dependencies

- **Software:** Chrome browser, Python 3.x, Flask framework, Selenium webdriver, Firebase Admin SDK, Redis queueing system, Node.js and Express.js framework, Chart.js library, Git version control system, VS Code IDE.
- **Hardware:** Remote server to run Flask and Express.js applications.
- **Libraries:** request library (Python), datetime library (Python), Queue library (Python), redis-rq library (Python), csv library (Python), stringIO library (Python), csv-parser library (Node.js), multer library (Node.js), fs library (Node.js)
- **Services:** Cloud storage for Firestore database.

3.2 System Design

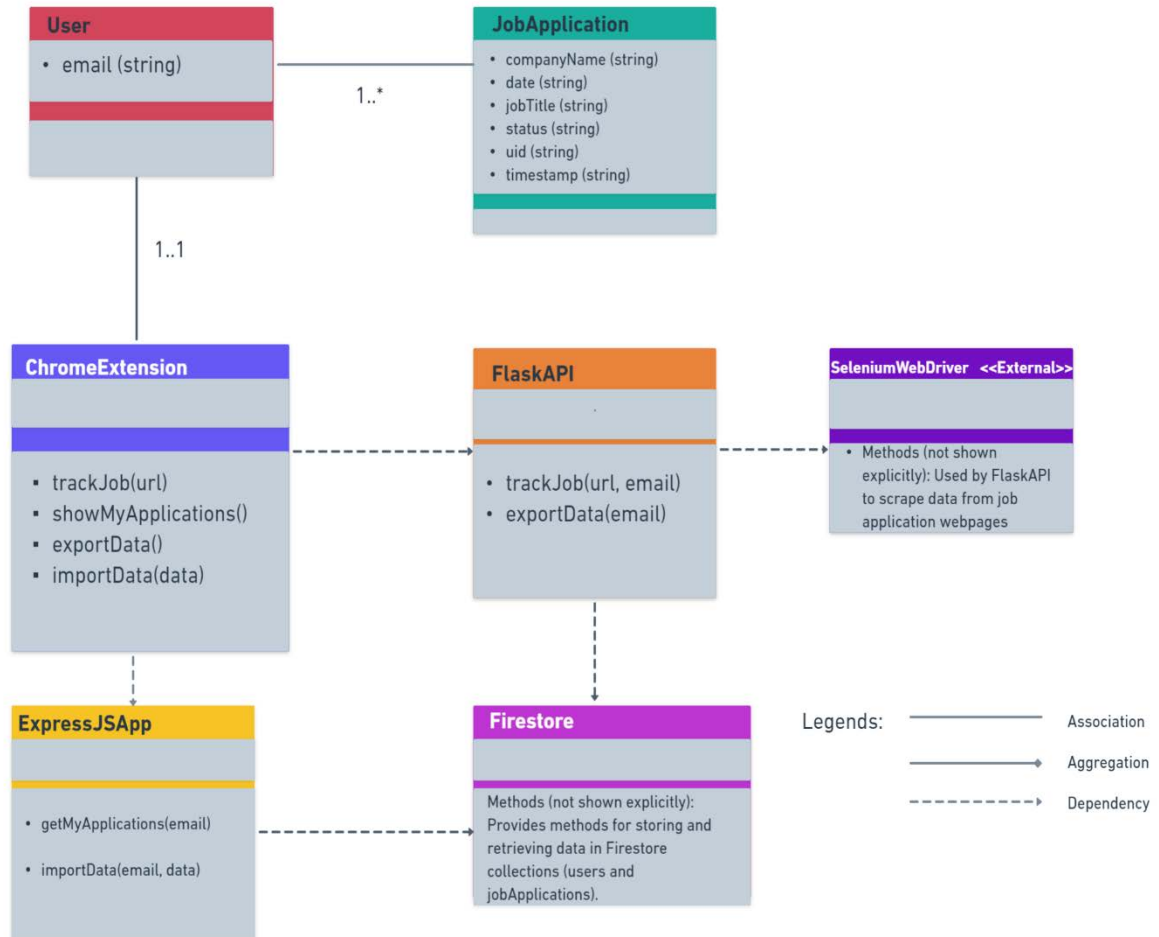


Fig. 3.2.1 Class Diagram

In Fig 3.2.1, the following attributes are there:

- **Class “User”**

Attributes are email (string): Unique identifier for the user

- **Class “Job Application”**

Attributes are companyName (string), date (string), jobTitle (string), status (string) - (Open, Closed, Pending, Resume Selected, Online Assessment Cleared, HR Round Cleared, Technical Round Cleared), uid (string): Foreign key referencing User.email, timestamp (string): Unique identifier for the job application (generated upon creation)

- **Class “Chrome Extension”**

Methods are trackJob(url): Initiates job application tracking for a given URL, showMyApplications(): Opens the "My Applications" page, exportData(): Exports user's job application data as a CSV file, importData(data): Imports job application data from a CSV file.

- **Class “Flask API”**

Methods are trackJob(url, email): Receives a track request, scrapes data using Selenium, and stores it in Firestore, exportData(email): Retrieves user's data from Firestore and prepares a CSV file for download.

- **Class “ExpressJSApp”**

Methods are getMyApplications(email): Retrieves user's job application data from Firestore, importData(email, data): Parses uploaded CSV file and stores data in Firestore.

- **Class “Firestore”**

Methods (not shown explicitly): Provides methods for storing and retrieving data in Firestore collections (users and jobApplications).

- **Class “SeleniumWebDriver” (External Class)**

Methods (not shown explicitly): Used by FlaskAPI to scrape data from job application webpages.

Class 1	Class 2	Relationship	Description
ChromeExtension	FlaskAPI	Dependency	ChromeExtension relies on FlaskAPI to scrape and store data from track requests.
ChromeExtension	ExpressJSApp	Dependency	ChromeExtension relies on ExpressJSApp for retrieving user data and importing CSV files.
User	JobApplication	Association (One-to-Many)	A User can have many JobApplications. Each JobApplication belongs to a specific User.
FlaskAPI (and ExpressJSApp)	Firestore	Dependency	Both FlaskAPI and ExpressJSApp depend on Firestore for data storage and retrieval.
FlaskAPI	SeleniumWebDriver (External Class)	Dependency	FlaskAPI utilizes SeleniumWebDriver for scraping data from job application webpages.

Fig. 3.2.2 Description of relationships between classes of class diagram

Fig 3.2.2 represents a table which describes the relationship between classes that are mentioned in the class diagram (Fig 3.2.1).

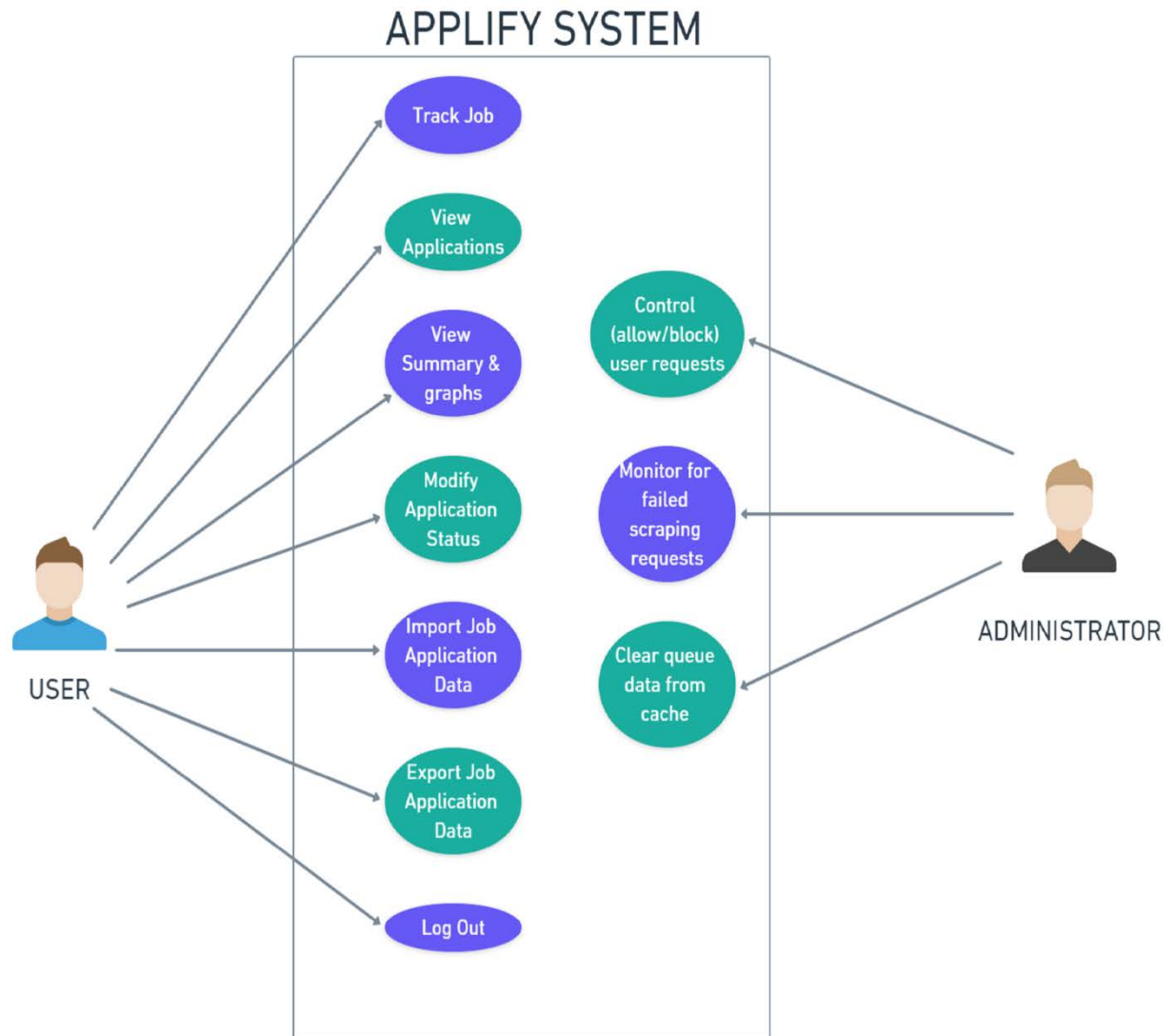


Fig. 3.2.3 Use Case Diagram

In the Figure 3.2.3, Actors: User, Admin. The "User" actor represents the user of the application, which interacts with various use cases. The "Admin" actor represents individuals or roles responsible for administering and managing the system, including transaction monitoring and issue resolution.

Use Cases of the Figure 3.2.3 are mentioned below

- **User tracks job:** The user clicks on the track button and sends request to scrape job details from that website
- **View Applications:** The user visits the "My Applications" page to look for all his job applications.

- **View Summary and Graphs:** User visits the “Summary and insights” menu to see his application data in a pie chart
- **Modify Application Status:** The user can choose to change the status of his application from a drop-down menu present beside each entry.
- **Import job application data:** User imports a csv file into his existing job applications.
- **Export job application data:** User downloads a copy of his job application data onto his local system in a csv file format.
- **Log Out:** User logs out of his google chrome browser, and in doing so, also logs out of the applify extension.
- **Control user requests:** Administrator can choose to suspend the services of any particular user, helpful in user regulation.
- **Monitor for failed scraping requests:** Administrator can see in the logs of the scraping API if there were any failed requests
- **Clear queue data from cache:** The administrator can manually clear the cache of the redis queue in case it fails to clear out automatically. If the cache gets full, new incoming requests will start getting declined.

User clicks on track, Job data is scraped, Scraped data stored in Firestore

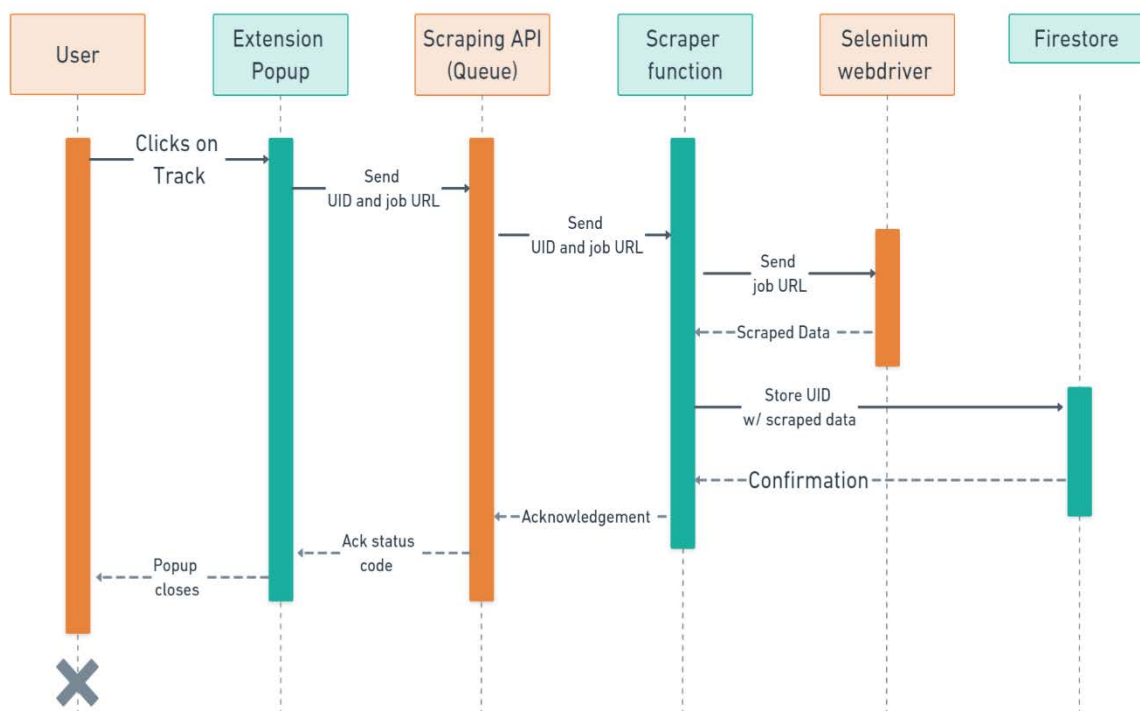


Fig. 3.2.4 Sequence Diagram 1

Fig. 3.2.4 denotes the scenario when the user clicks on “Track” button of the extension on a particular job website.

User clicks on My Applications (and on Summary and Insights later on), Data fetched from Firestore, Data displayed in a new tab

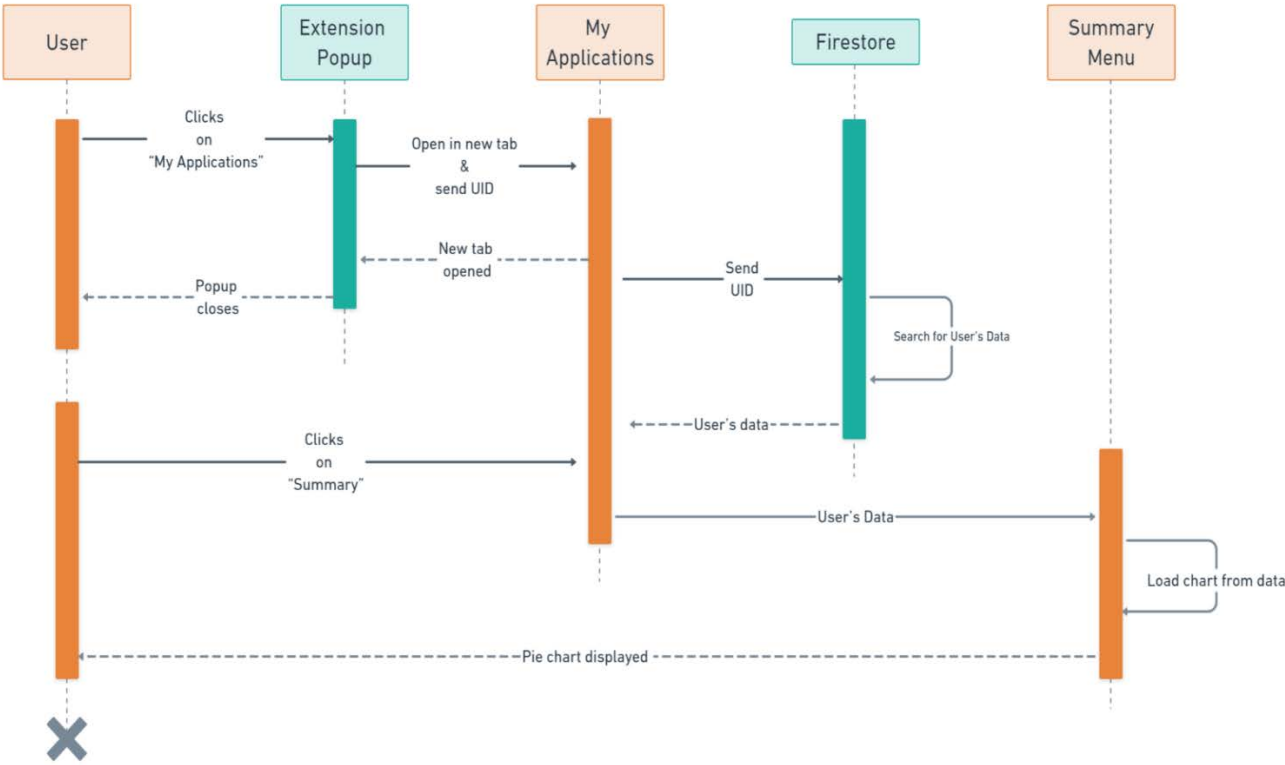


Fig. 3.2.5 Sequence Diagram 2

Fig. 3.2.5 denotes the scenario of user trying to view his applications’ data and statistics using “My Applications” and “Summary and Insights” menus respectively.

User uploads a csv file of job application data, Call made to express server to store the data into firebase, Data added to user's collection in firebase

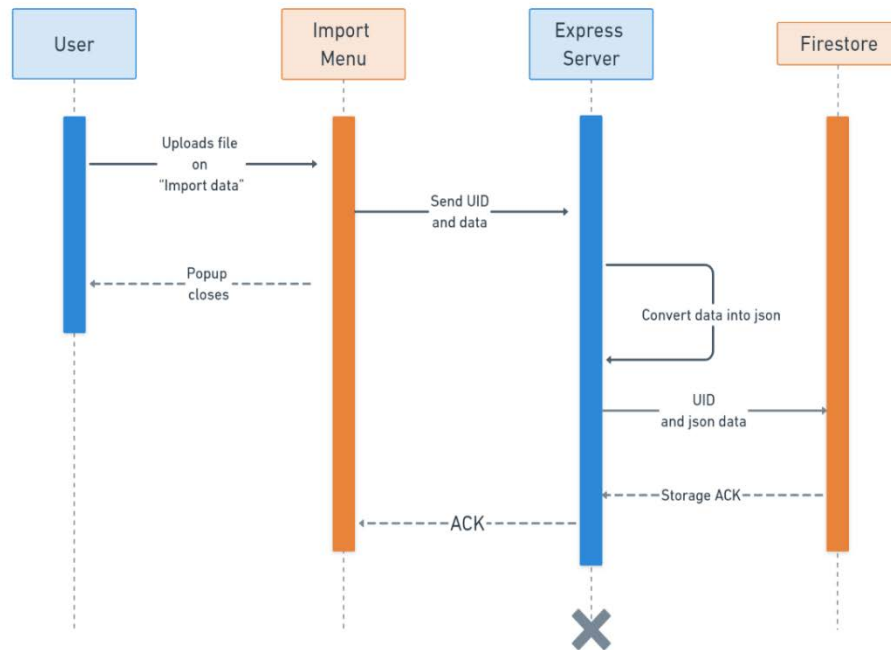


Fig. 3.2.6 Sequence Diagram 3

Fig 3.2.6 denotes the scenario of Importing a CSV file (containing job application data) by the user.

User clicks on "Export Data" downloads a CSV file of job applications

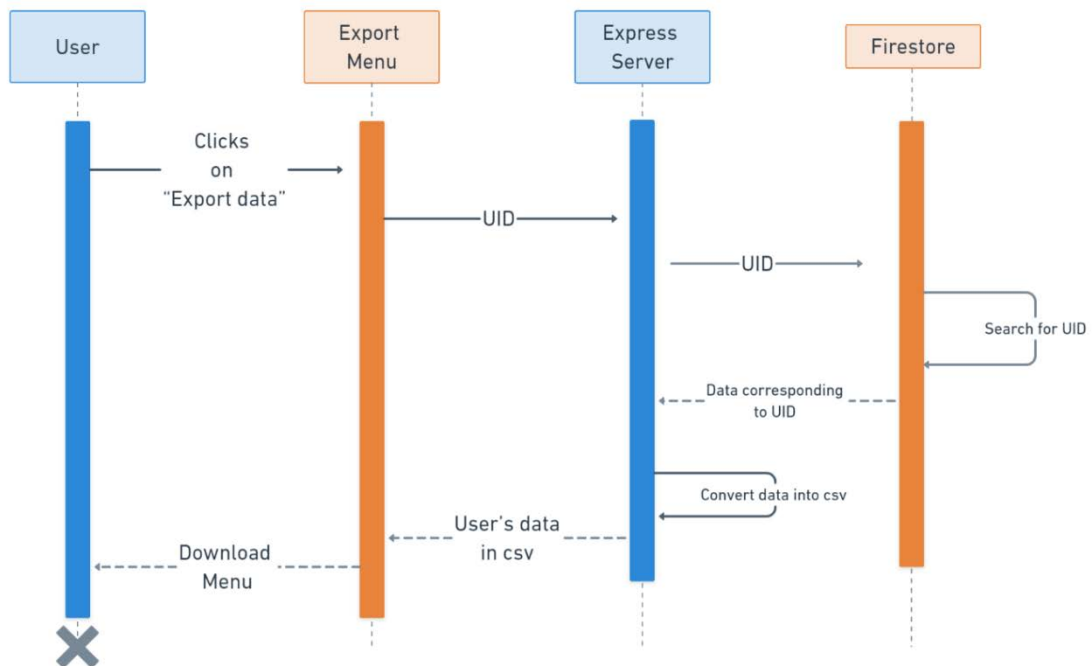


Fig. 3.2.7 Sequence Diagram 4

Fig.3.2.7 denotes the scenario of exporting the user's job application data onto a CSV file in their local system.

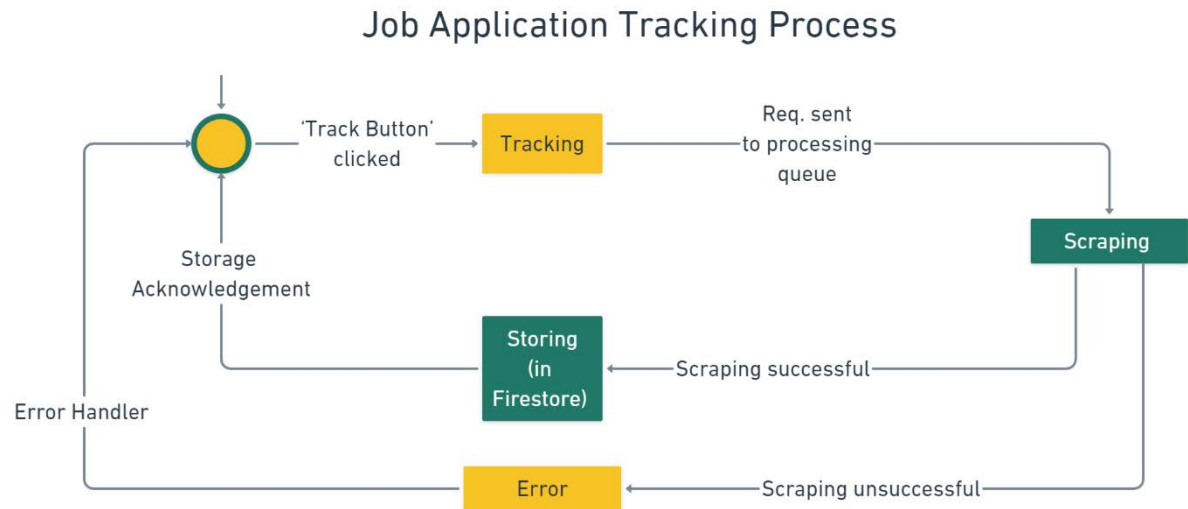


Fig. 3.2.8 State Diagram 1

Fig. 3.2.8 Represents the state transitions that occur from the moment a user decides to track a job application to the final storage of the data in Firestore. The different states of Fig.3.2.8 are explained below

- **Idle State:** The default state when no tracking is active.
- **Tracking State:** Activated when the user clicks the 'track button', initiating the tracking process.
- **Scraping state:** Data scraping starts after the request is sent to the processing queue.
- **Storing State:** Upon successful data scraping, the data is stored in Firestore.
- **Error:** This state is reached if scraping fails, with the system handling the error and returning to Idle.

Export Data

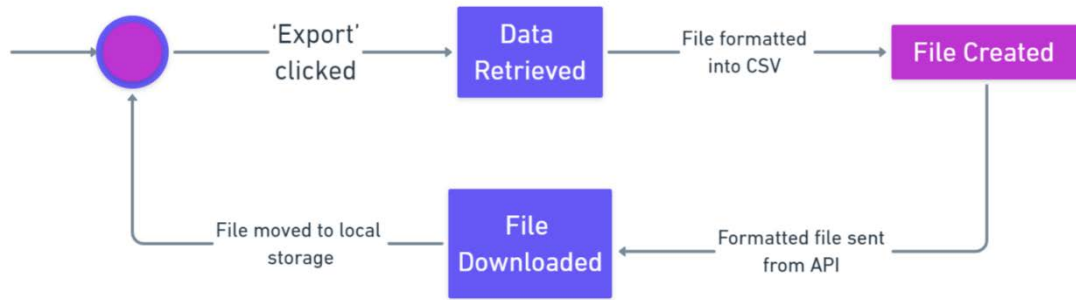


Fig. 3.2.9 State Diagram 2

Fig. 3.2.9 Illustrates the states and transitions involved in exporting data from the extension to a CSV file. The different states of Fig.3.2.9 are explained below

- **Idle State:** Initial state where the extension is ready to export data.
- **Data Retrieved State:** Transition after the user initiates the export process.
- **File Created State:** Data is retrieved and formatted for CSV file creation.
- **File Downloaded State:** CSV file is created and available for download
- **Idle State:** Export process is complete, returning to the initial state.

Import Data Process

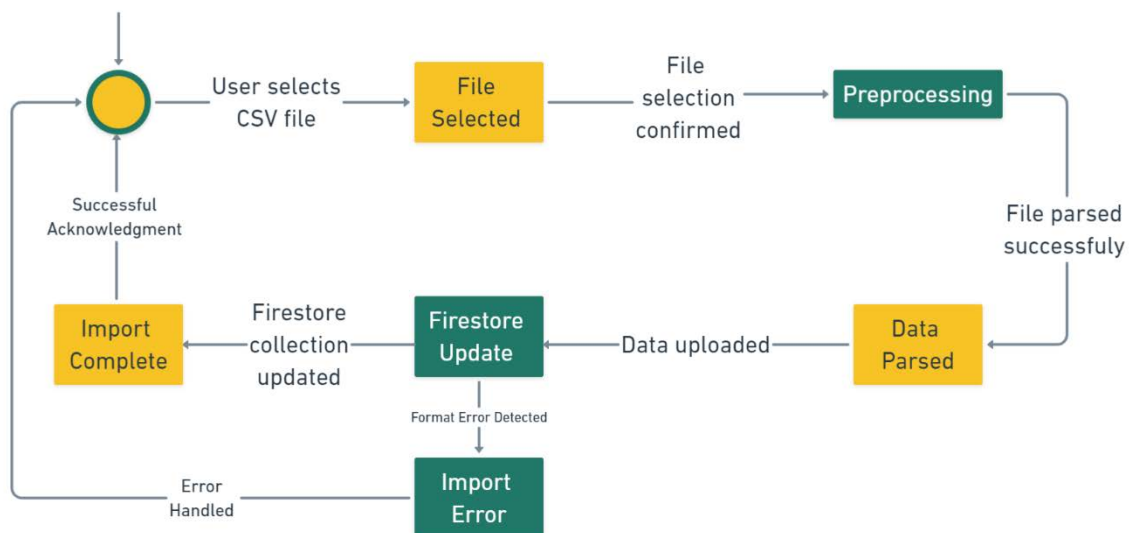


Fig. 3.2.10 State Diagram 3

Fig. 3.2.10 Outlines the states and transitions involved in importing data into the extension from a CSV file. The different states of Fig.3.2.10 are explained below

- **Idle State:** Initial state where the extension is waiting for the user to select a CSV file.

- **File Selected State:** Transition after the user selects a CSV file for import.
- **Preprocessing State:** State where the CSV file is processed, and data is parsed.
- **Data Parsed State:** Successful parsing of CSV data, ready for upload.
- **Firestore Update State:** Data is sent and updated in Firestore.
- **Import Complete State:** Successful completion of the import process.
- **Import Error State:** State reached if there's an error during parsing or upload.

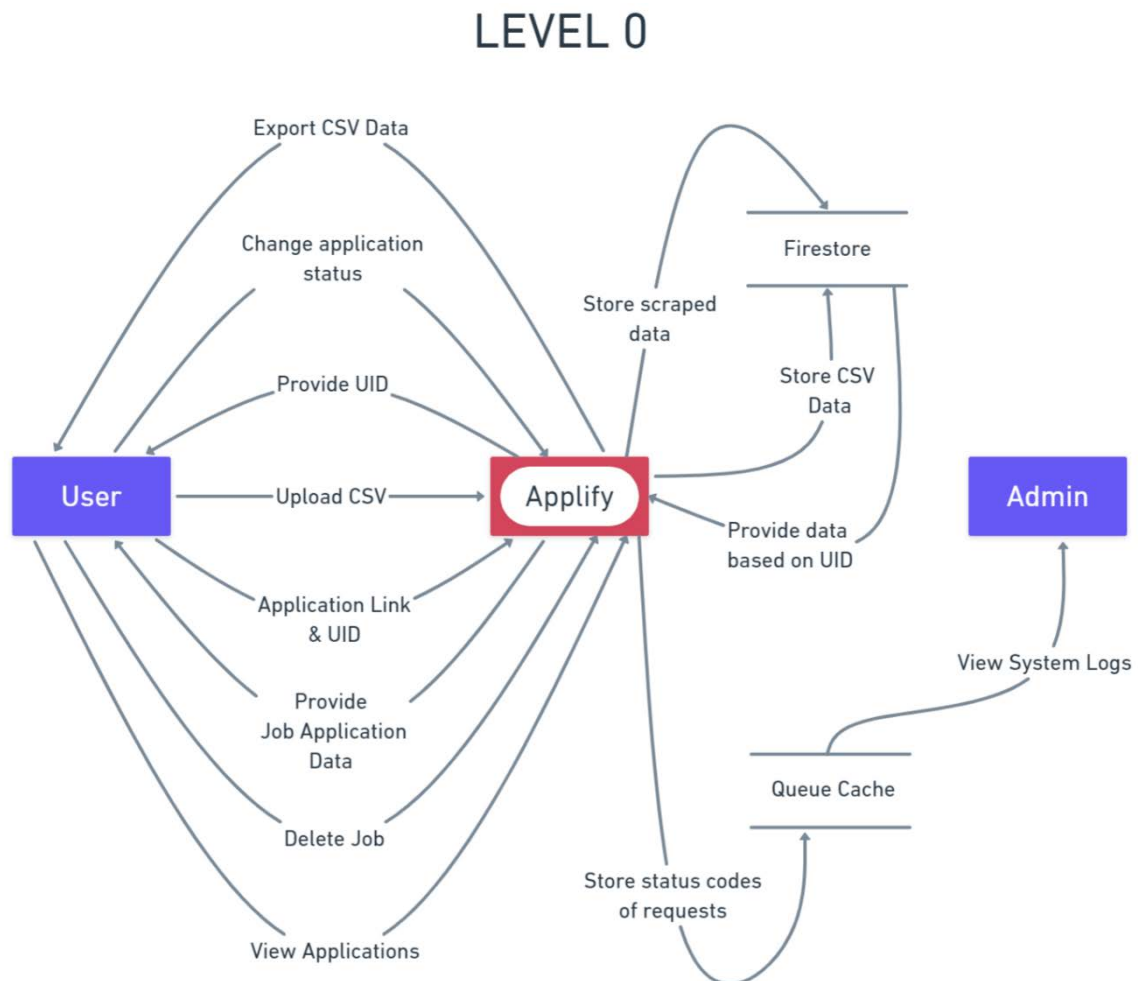


Fig. 3.2.11 Context Diagram

The Fig. 3.2.11 depicts a context diagram. The components of the context diagram are:

- **Data Stores:** Firestore (Job application data), Redis Queue Cache (Results of scraping, Status codes of each request)
- **External Entities:** User, Admin
- **Data flows:** Provide UID (Application to User), Application Link and UID (User to Application), CSV File (User to Application), Status change request (User to Application), Delete Job (User to Application), CSV File (download) (Application to User), Job

Application Data (Application to User), View Applications (User to Application), Store Scraped Data (Application to Firestore), Store CSV Data (Application to Firestore), Data retrieval by UID (Firestore to Application), Store Status Codes (Application to Queue Cache), View System Logs (Queue Cache to Admin)

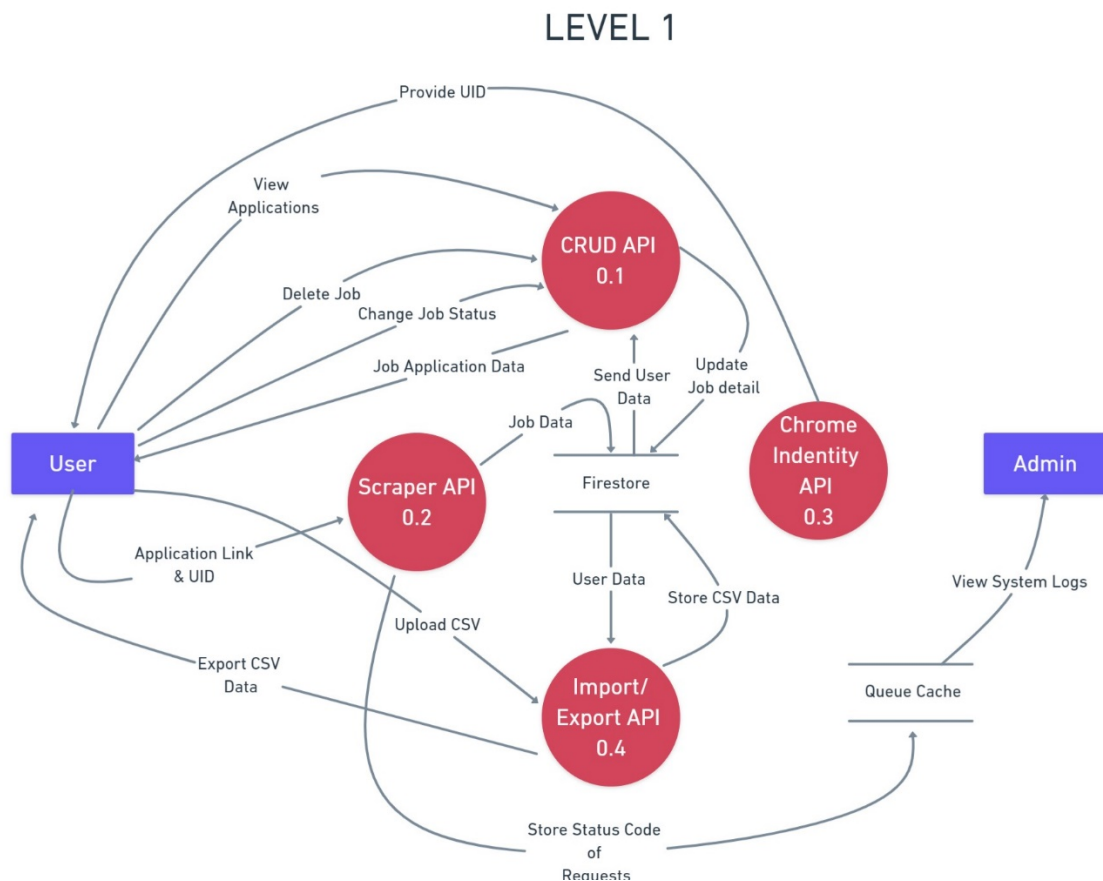


Fig. 3.2.12 Data flow Diagram Level 1

Fig. 3.2.12 depicts the level 1 of the data flow diagram. The components are explained below:

Datastores, External Entities and Dataflows, same as context diagram

Processes:

- Chrome.identity API
- Scraper API
- CRUD API
- Import/Export API

3.3 Proposed User Interface Design of the project

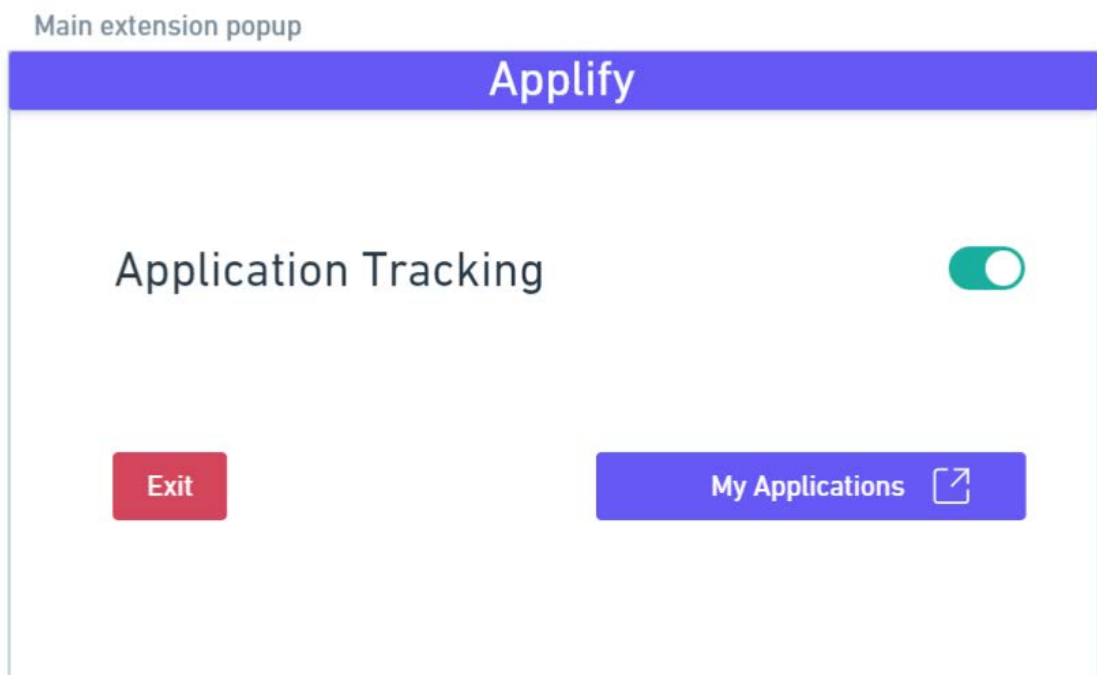


Fig. 3.3.1 UI design of the extension popup

Fig. 3.3.1 Shows the primary interface for all the users.

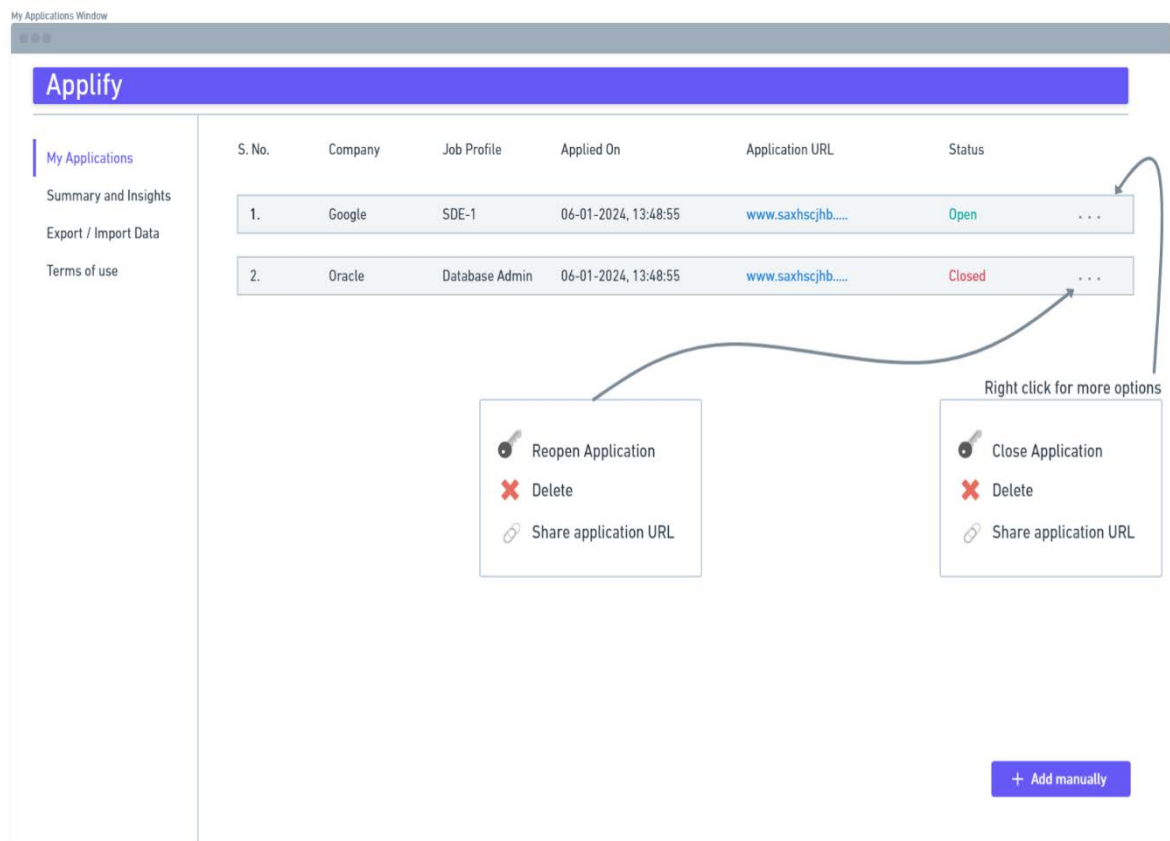


Fig. 3.3.2 UI design of the “My Applications” page

Fig. 3.3.2 Shows the menu containing the job data of the users in a tabular format. Here users can change the status of their applications, delete applications, or share any application’s URL.

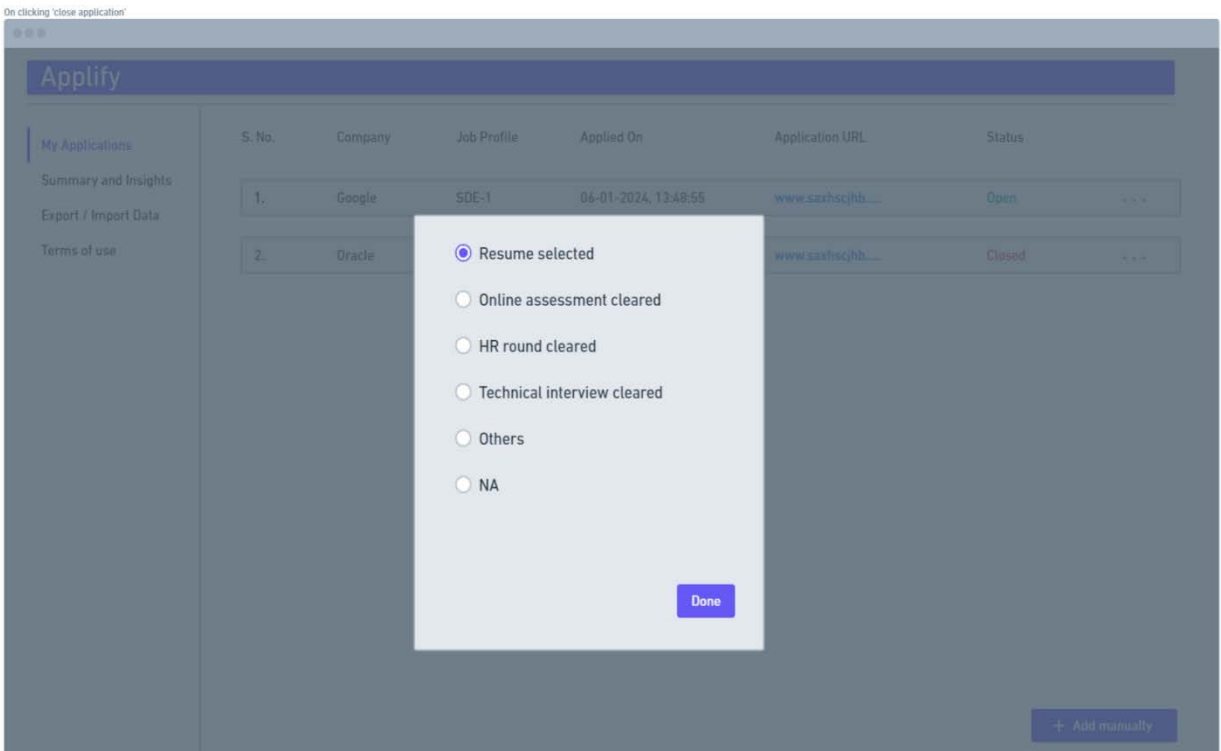


Fig. 3.3.3 UI design of “Close Application” page.

Fig. 3.3.3 show the UI design of when a user decides to close an application and clicks on the close button inside the overflow menu. A list of options appears, prompting the user to select a stage till where the job application went forward.

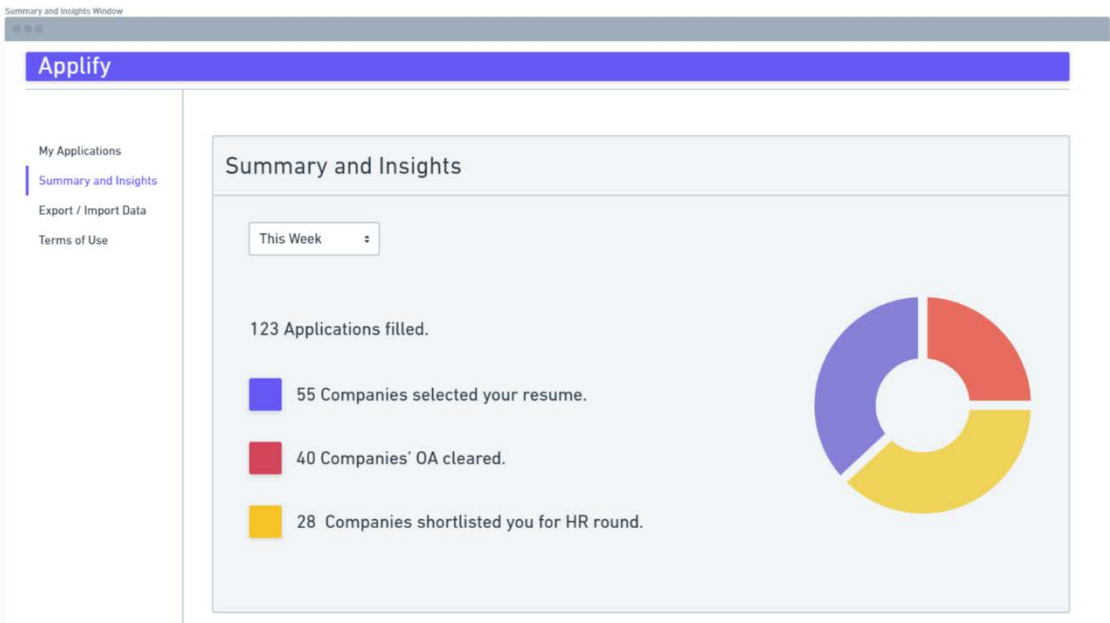


Fig. 3.3.4 UI design of the “Summary and Insights” page.

Fig. 3.3.4 shows a pie chart of the job applications, and the chart is sorted by the various stages that the user’s job applications went through.

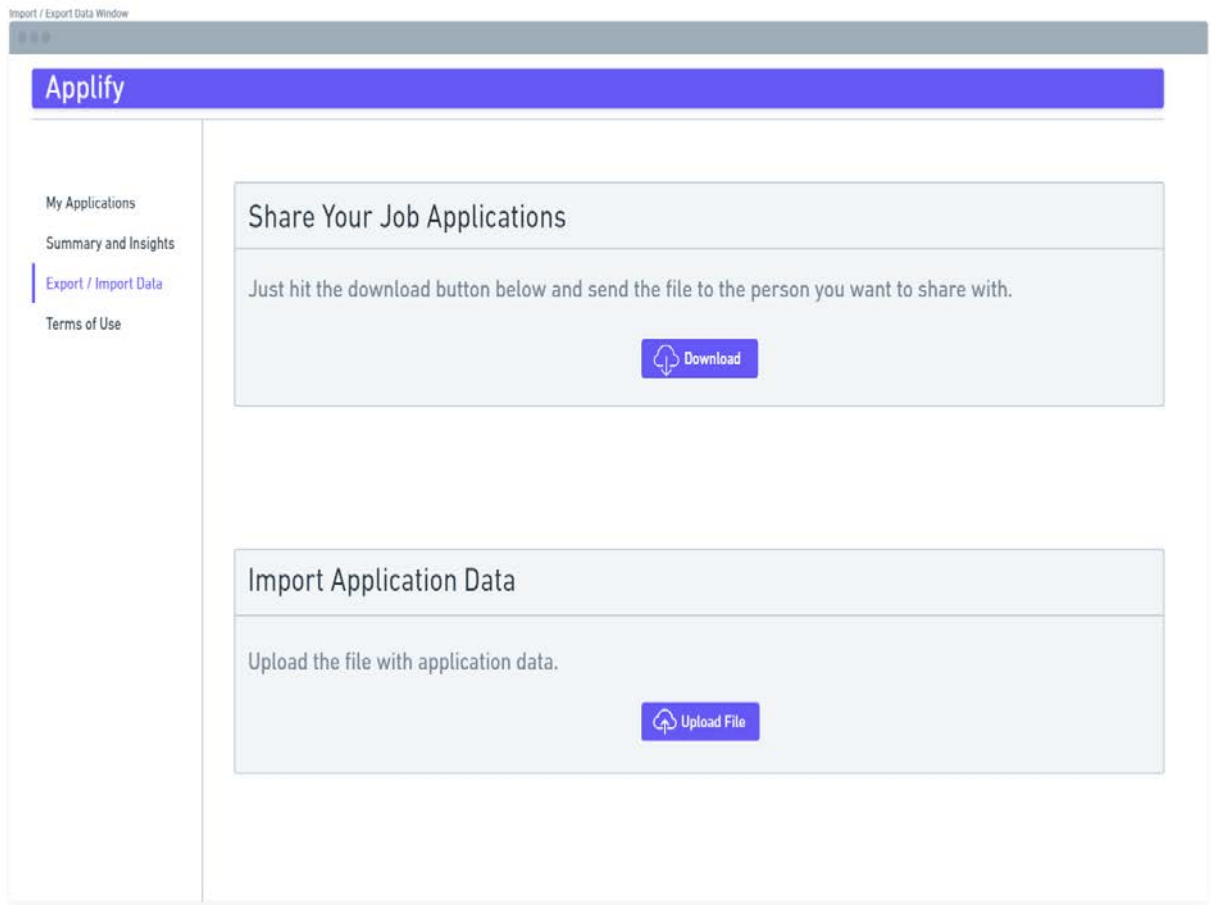


Fig. 3.3.5 “UI design of the Imports and Exports” page

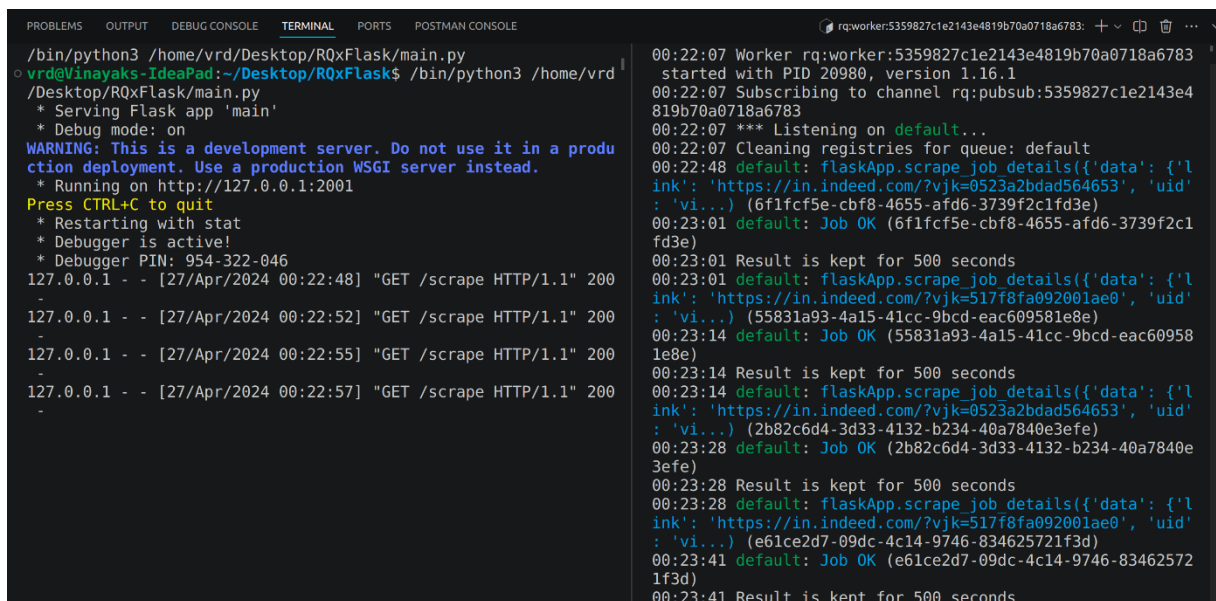
Fig. 3.3.5 shows the UI design of the page from where the user can download all his job application data in a comma separated file format by clicking on the “Download” button. Furthermore, the user can also upload a CSV file containing the job data of any other use. This data shall be appended to their existing data.

CHAPTER 4

RESULTS/ OUTPUTS

4.1 Performance of scraping API

In our testing, the scraping API running on the Flask server takes (on an average) approximately five (5) seconds to process one request, given the bandwidth of internet connection to the system is 20 MB/sec. Here, the internet speed plays a larger role in deciding processing time of the requests rather than the efficiency of the scraping algorithm. This is because the actual scraping task i.e. opening the link sent by the user in a chrome session, waiting for the page to load, and then scraping for relevant data is the actual complex task in this scenario. This is the reason why implementing a queue was both necessary and beneficial to solve our problem. The redis queue stores the scraped data in its cache for 500 seconds (approximately 8 minutes) before flushing it off. In our testing, we tested for 10 concurrent requests and the system processed each one of them without failure. The cache storage time of 500 seconds might become a problem for large number of concurrent requests; however, further testing is required to find such upper limit.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
/bin/python3 /home/vrd/Desktop/RQxFlask/main.py
vrd@vinayaks-IdeaPad:~/Desktop/RQxFlask$ /bin/python3 /home/vrd/Desktop/RQxFlask/main.py
* Serving Flask app 'main'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:2001
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 954-322-046
127.0.0.1 - - [27/Apr/2024 00:22:48] "GET /scrape HTTP/1.1" 200 -
127.0.0.1 - - [27/Apr/2024 00:22:52] "GET /scrape HTTP/1.1" 200 -
127.0.0.1 - - [27/Apr/2024 00:22:55] "GET /scrape HTTP/1.1" 200 -
127.0.0.1 - - [27/Apr/2024 00:22:57] "GET /scrape HTTP/1.1" 200 -
00:22:07 Worker rq:worker:5359827c1e2143e4819b70a0718a6783 started with PID 20980, version 1.16.1
00:22:07 Subscribing to channel rq:pubsub:5359827c1e2143e4819b70a0718a6783
00:22:07 *** Listening on default...
00:22:07 Cleaning registries for queue: default
00:22:48 default: flaskApp.scrape_job_details({'data': {'link': 'https://in.indeed.com/?vjk=0523a2bdad564653', 'uid': 'vi...'} (6f1fcf5e-cbf8-4655-afd6-3739f2c1fd3e)
00:23:01 default: Job OK (6f1fcf5e-cbf8-4655-afd6-3739f2c1fd3e)
00:23:01 Result is kept for 500 seconds
00:23:01 default: flaskApp.scrape_job_details({'data': {'link': 'https://in.indeed.com/?vjk=517f8fa092001ae0', 'uid': 'vi...'} (55831a93-4a15-41cc-9bcd-eac609581e8e)
00:23:14 default: Job OK (55831a93-4a15-41cc-9bcd-eac609581e8e)
00:23:14 Result is kept for 500 seconds
00:23:14 default: flaskApp.scrape_job_details({'data': {'link': 'https://in.indeed.com/?vjk=0523a2bdad564653', 'uid': 'vi...'} (2b82c6d4-3d33-4132-b234-40a7840e3efe)
00:23:28 default: Job OK (2b82c6d4-3d33-4132-b234-40a7840e3efe)
00:23:28 Result is kept for 500 seconds
00:23:28 default: flaskApp.scrape_job_details({'data': {'link': 'https://in.indeed.com/?vjk=517f8fa092001ae0', 'uid': 'vi...'} (e61ce2d7-09dc-4c14-9746-834625721f3d)
00:23:41 default: Job OK (e61ce2d7-09dc-4c14-9746-834625721f3d)
00:23:41 Result is kept for 500 seconds
```

Fig. 4.1.1 Server-side logs of user requests to the flask server

Figure 4.1.1 shows the flask scraping server running on port 2001 and a rq worker (redis queue) listening for requests. It demonstrates 4 simultaneous being handled on the scraping server with an average response time of 2 seconds. This is made possible by adding the tasks as they arrive, to the queue where they actually get processed, while giving a “200 OK” response code to the client at the time of queueing. The queue is taking around 13 seconds to scrape and store for each request.

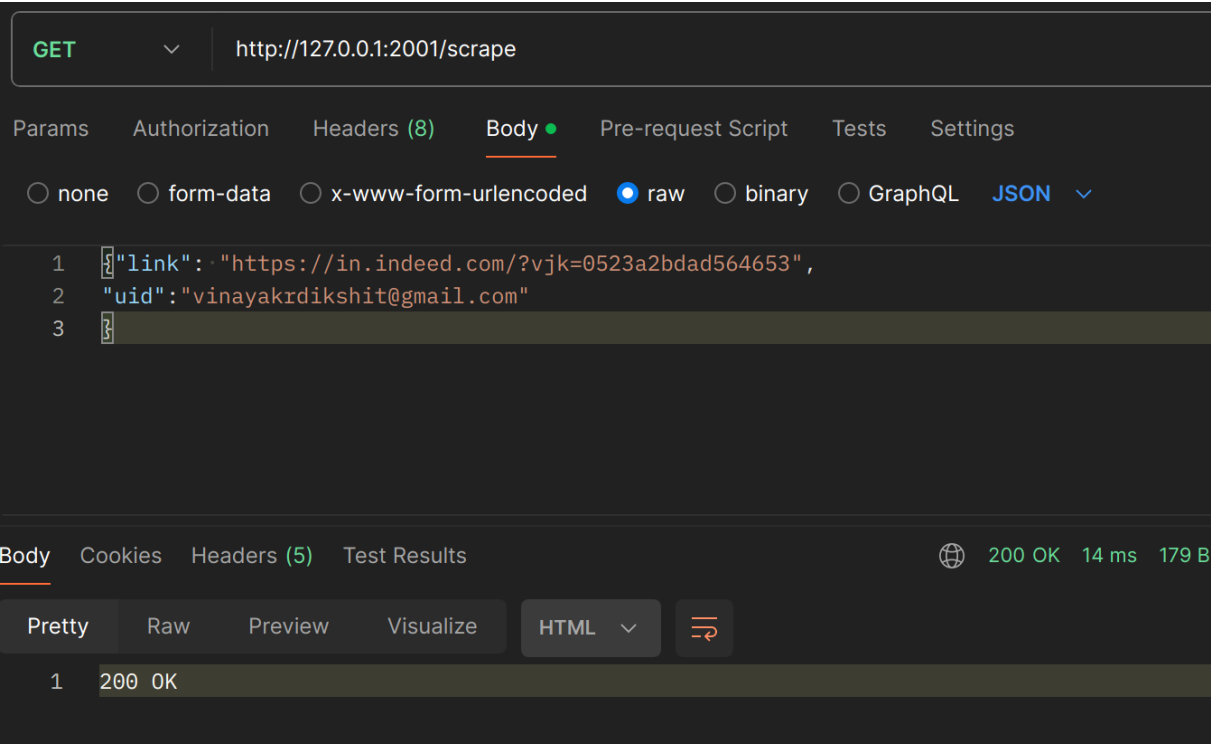


Fig. 4.1.2 Testing the scraping API on Postman.

In Fig.4.1.2 The API shown requires user’s UID and link (to be scraped) to be sent in the body of the request. A 200 OK response code suggests successful execution of the request.

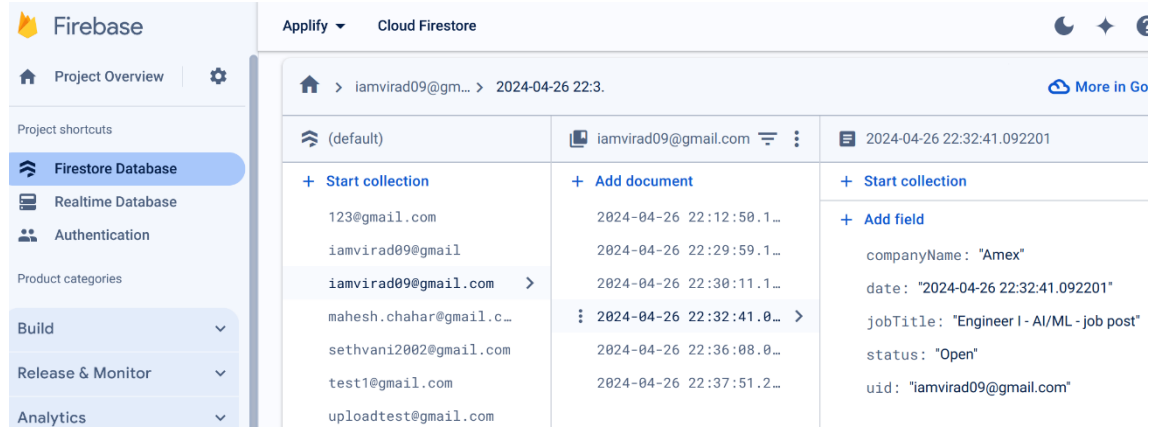


Fig. 4.1.3 Organisation of user’s data in Firestore

Figure 4.1.3 shows how data is organised in the firestore database. Each user has his own collection which is uniquely identified by the user's UID (email ID). Inside a user's collection are the jobs which he has tracked. The jobs are organised into documents and each job for a particular user is uniquely identified via the timestamp of application. Each job document holds the necessary job details in the form of key-value fields.

4.2 Performance of data retrieval, import, export APIs

All these APIs take processing time in the order of milli seconds and have found to be rather susceptible to high traffic. This is because they perform simple tasks such as parsing and converting csv into json and vice versa, reading from (or writing into) firestore database etc. The upper limit for concurrent users of these APIs shall also require further testing.

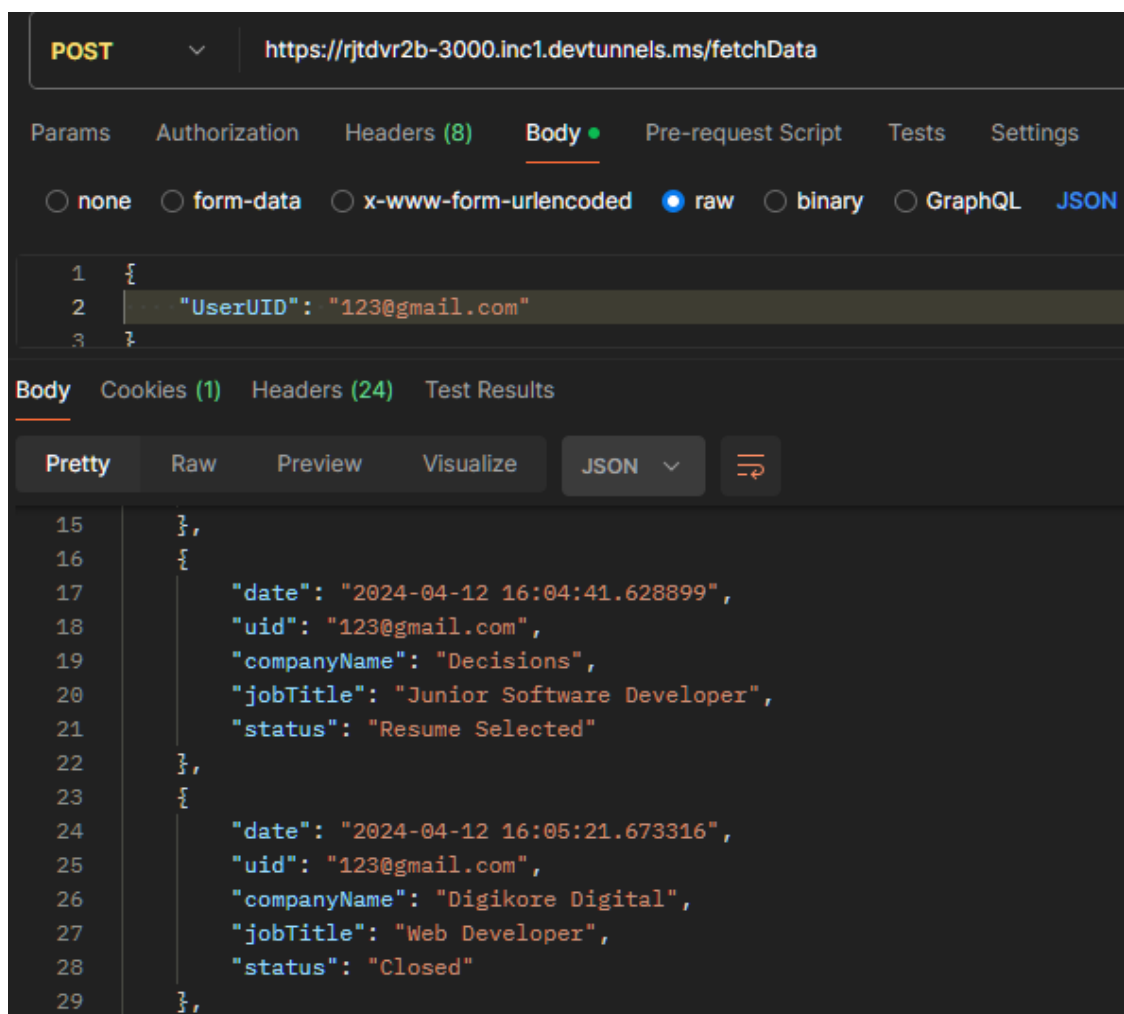
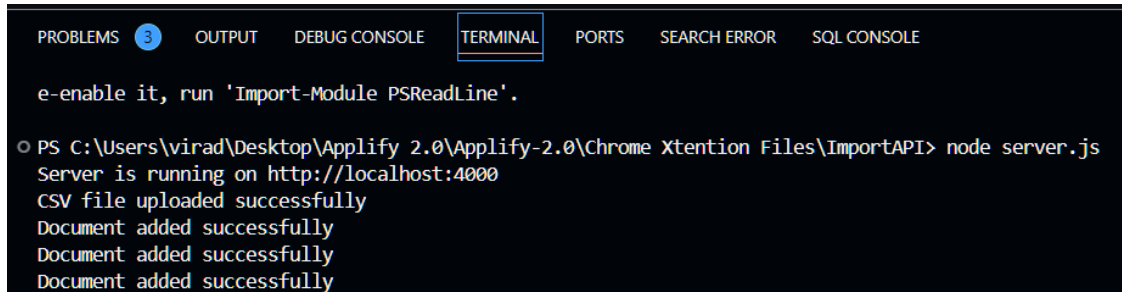


Fig. 4.2.1 Calling the fetch data API on Postman.

In Fig. 4.2.1 The fetch data API is responsible for populating the jobs' details whenever the user clicks on the "My Applications" page. It retrieves user's data from firestore and puts it in the table on the "My Applications" page. The API requires the user's UID to be sent in the request body.



```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR SQL CONSOLE

e-enable it, run 'Import-Module PSReadLine'.

PS C:\Users\virad\Desktop\Applify 2.0\Applify-2.0\Chrome Xtention Files\ImportAPI> node server.js
Server is running on http://localhost:4000
CSV file uploaded successfully
Document added successfully
Document added successfully
Document added successfully
Document added successfully
```

Fig. 4.2.2 Server-side logs of Import API.

In Fig. 4.2.2 The API is called when the user uploads a CSV file. The logs tell the status of when the file is uploaded to the APIs cache, gets passed from the Multer middleware, and finally gets added to the user's firestore collection.

4.3 Actual User Interface of the extension

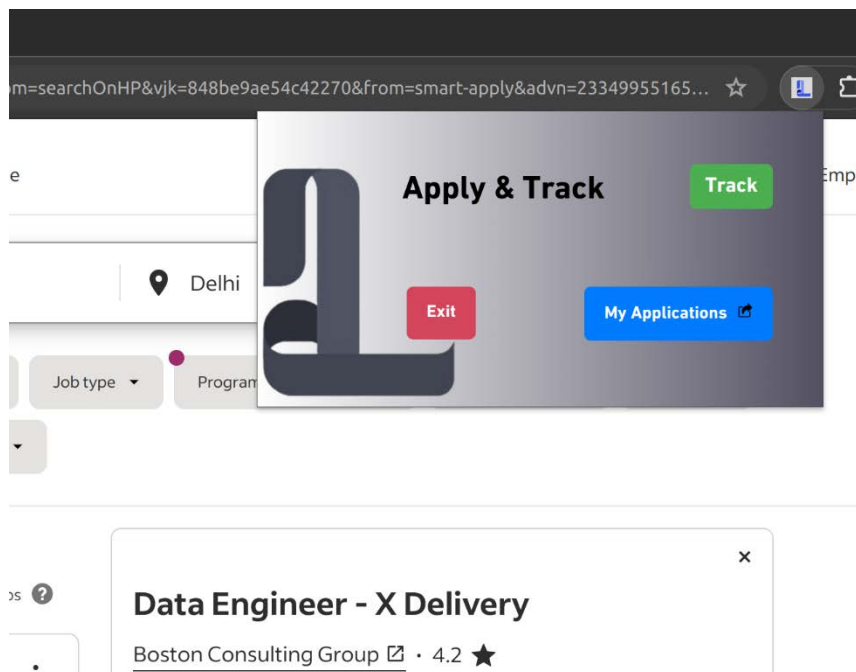
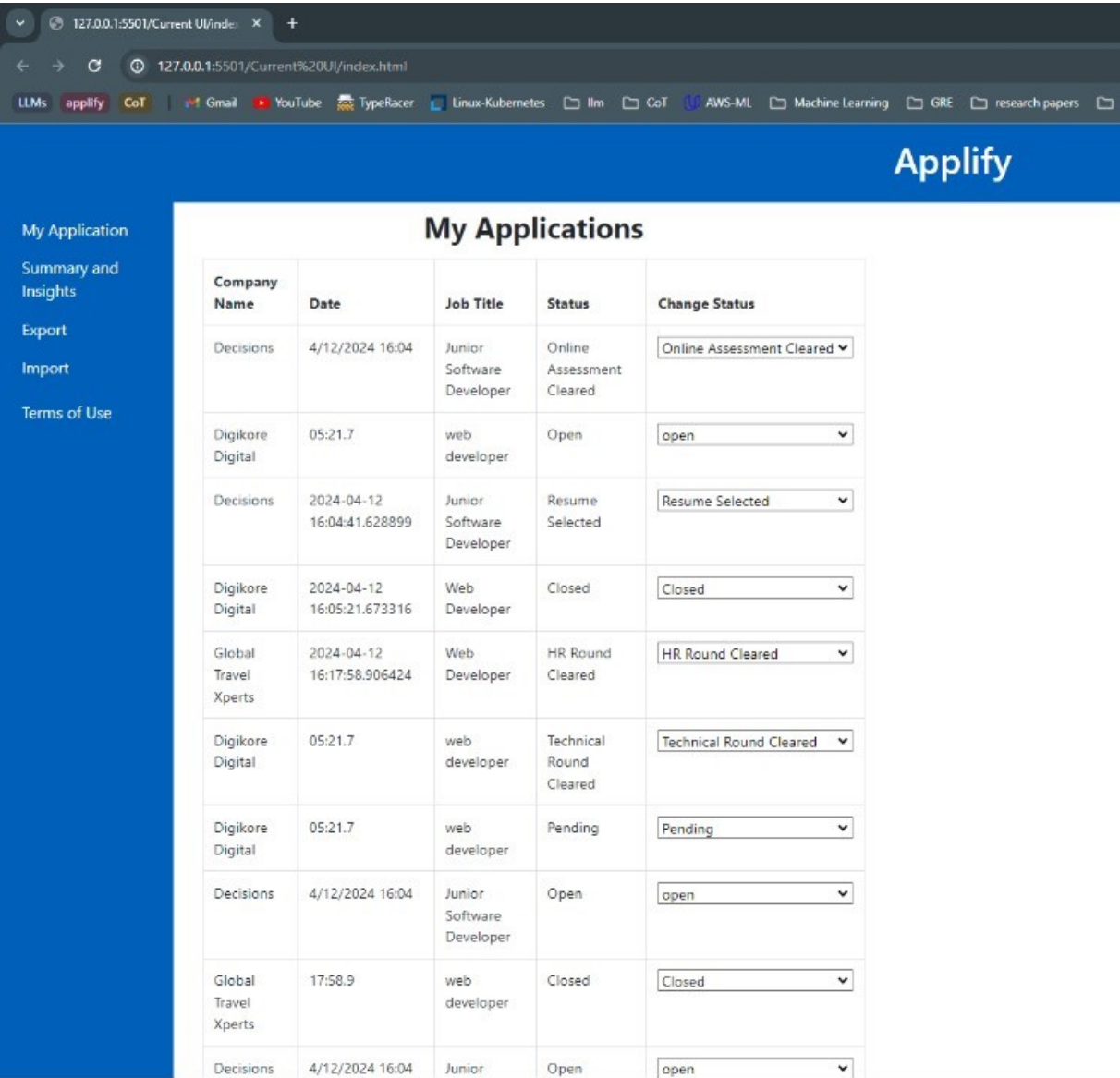


Fig. 4.3.1 Main Popup of the chrome extension - Applify

In Fig. 4.3.1 Shows the popup of Applify opened on Indeed.com's webpage. The popup appears on clicking the extension's icon in Google Chrome. The popup consists of a "Track", a "My

applications” button and an “Exit” button. Opening a job website on a new tab and clicking track button turns on job application tracking on that website.



Company Name	Date	Job Title	Status	Change Status
Decisions	4/12/2024 16:04	Junior Software Developer	Online Assessment Cleared	Online Assessment Cleared ▾
Digikore Digital	05:21.7	web developer	Open	open ▾
Decisions	2024-04-12 16:04:41.628899	Junior Software Developer	Resume Selected	Resume Selected ▾
Digikore Digital	2024-04-12 16:05:21.673316	Web Developer	Closed	Closed ▾
Global Travel Xperts	2024-04-12 16:17:58.906424	Web Developer	HR Round Cleared	HR Round Cleared ▾
Digikore Digital	05:21.7	web developer	Technical Round Cleared	Technical Round Cleared ▾
Digikore Digital	05:21.7	web developer	Pending	Pending ▾
Decisions	4/12/2024 16:04	Junior Software Developer	Open	open ▾
Global Travel Xperts	17:58.9	web developer	Closed	Closed ▾
Decisions	4/12/2024 16:04	Junior	Open	open ▾

Fig. 4.3.2 The “My Applications” page.

In Fig. 4.3.2 Upon clicking on the “My Applications” button, a new tab opens with a side panel, containing options such as “My applications”, “Summary and Insights” and “Import/export data”. The default selection on the side panel is My Applications.

My Applications

Company Name	Date	Job Title	Status	Change Status
Decisions	4/12/2024 16:04	Junior Software Developer	Online Assessment Cleared	Online Assessment Cleared ▼
Digikore Digital	05:21.7	web developer	Open	Open ▼
Decisions	2024-04-12 16:04:41.628899	Junior Software Developer	Resume Selected	Resume Selected ▼
Digikore Digital	2024-04-12 16:05:21.673316	Web Developer	Closed	Closed ▼
Global Travel Xperts	2024-04-12 16:17:58.906424	Web Developer	HR Round Cleared	HR Round Cleared ▼
Digikore Digital	05:21.7	web developer	Open	Open ▼
Digikore Digital	05:21.7	web developer	Pending	Pending ▼
Decisions	4/12/2024 16:04	Junior	Open	Open ▼

Fig. 4.3.3 The “My Applications” page with “Change status” dropdown menu.

In Fig. 4.3.3, in my applications menu, a user can- View all his applications, Change the status of each job, Delete each job. A job can have the following statuses - “Open”, “Closed”, “Pending”, “Resume Selected”, “Online Assessment Cleared”, “HR Round Cleared”, “Technical Round Cleared”. These statuses appear in a drop-down menu beside each job from where they can be updated.

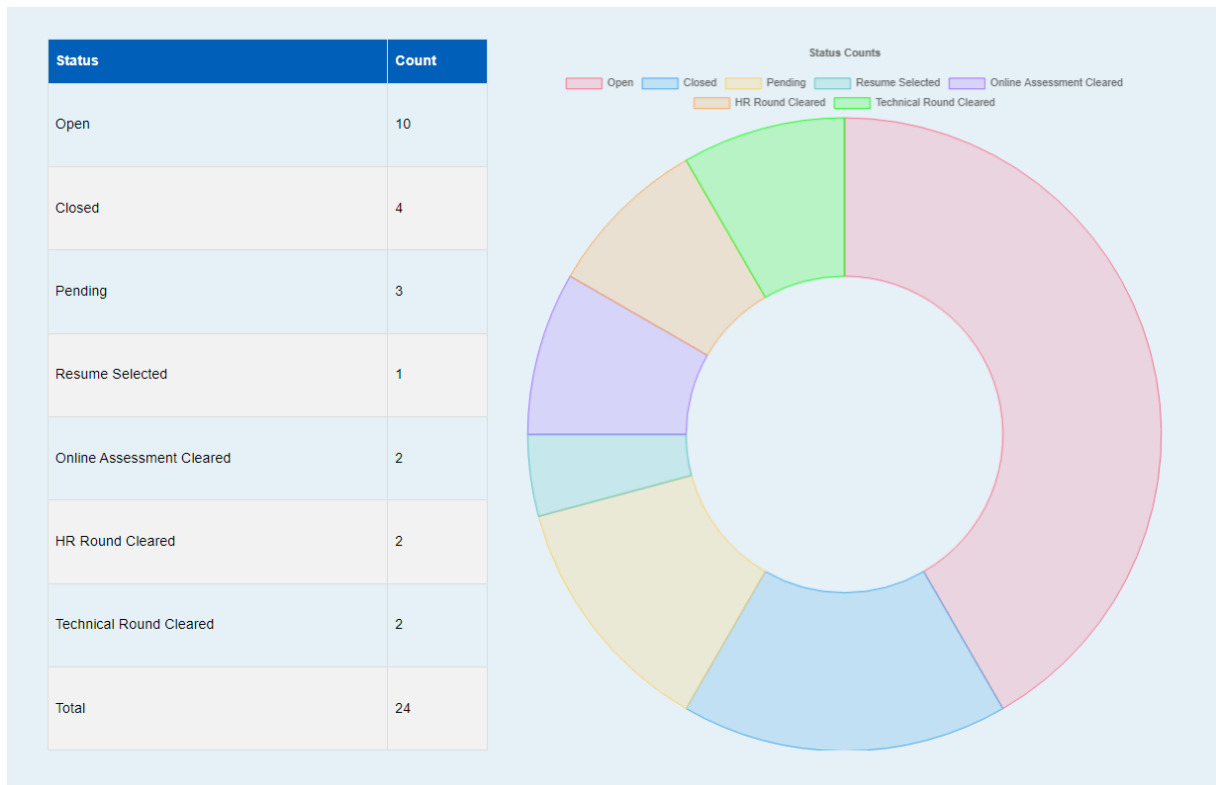


Fig. 4.3.4 The “Summary and Insights” page.

In Fig. 4.3.4, in the summary and insights page, the user can see a pie chart of the applications filled by him. The different sectors of the pie chart are the different job statuses (like open, closed, resume selected, online test cleared, HR round cleared etc). chart.js is used for making this chart. The statistics to create the sectors of the pie chart are taken from the My applications page (each sector represents the percentage of each status). For example, if the user has applied to 100 jobs in total and out of them, 25 have proceeded to the HR round, then the HR round sector will occupy 1/4th area in the pie chart.

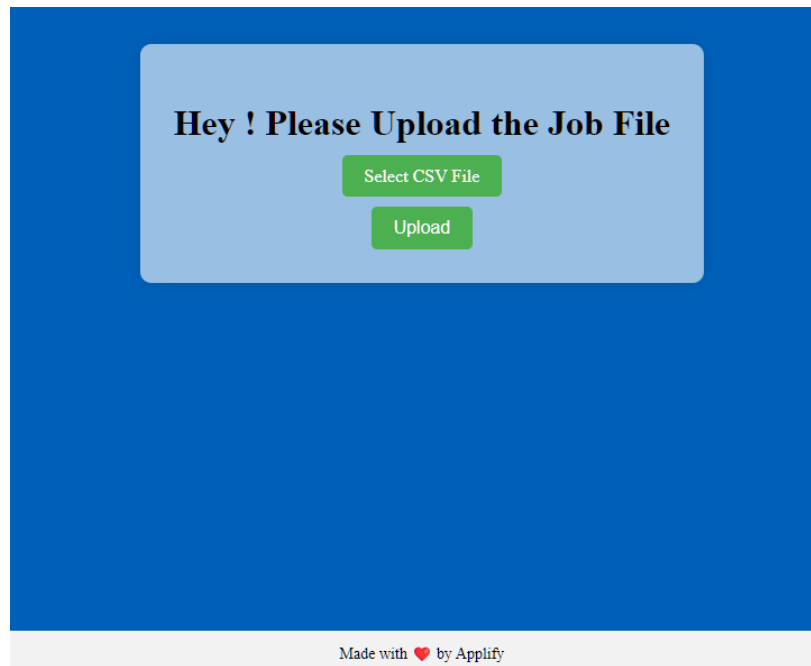


Fig. 4.3.5 The “Import” (CSV file) prompt.

In Fig. 4.3.5, in the import menu, the user can import a CSV file sent by another user to his own applications database.

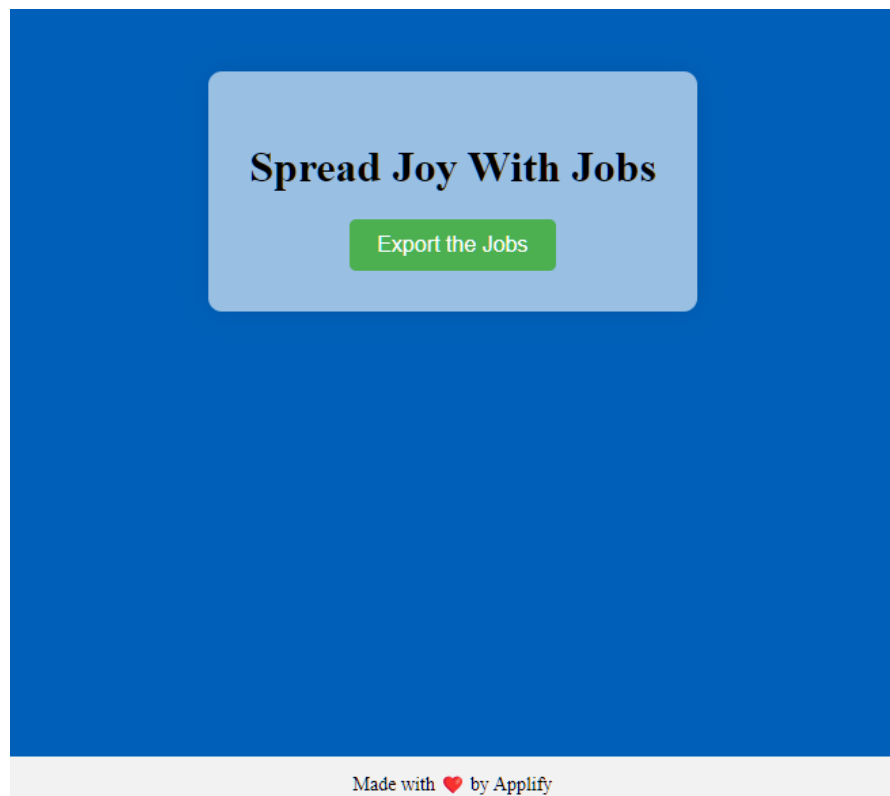


Fig. 4.3.6 The “Export” (into CSV file) prompt.

In Fig.4.3.6, in the export menu, user can export all his job details into a CSV file, which is downloaded into their system’s local storage.

CHAPTER 5

CONCLUSIONS/RECOMMENDATIONS

The project has achieved its primary objectives, providing a seamless and efficient solution for tracking job opportunities on websites like LinkedIn.com and Indeed.com.

The system is highly scalable, accommodating future growth and additional features. This scalability is a valuable asset for the users and developers.

Applify has emerged as a promising tool for users seeking to streamline their job application management across various online platforms. However, several areas offer exciting opportunities for further development that can enhance user experience, optimize performance, and explore potential revenue streams.

Following are the recommendations for future versions (if any) of the project:

- **Enhancing Scraping Efficiency:**

While Selenium WebDriver offers a powerful solution for browser automation and testing, its resource-intensive nature might not be ideal for real-world deployment within Applify. Investigating alternative scraping methods specifically designed for web data extraction can significantly improve scraping speed and efficiency. Libraries like BeautifulSoup or Scrapy offer robust functionalities tailored for this purpose. Reducing scraping time from the estimated 10 seconds would not only improve user experience but also enhance the overall scalability of Applify in handling multiple user requests simultaneously.

- **Strengthening Security and Scalability:**

As Applify's user base grows, implementing an administrator panel becomes increasingly crucial. This dedicated interface would empower administrators to monitor user activity, identify potential scraping abuse, and ensure that Applify's functionalities are used responsibly. A robust administrator panel would bolster Applify's security posture and allow for proactive management of the platform's resources.

- **Exploring Monetization Strategies:**

Applify currently caters to a specific user need, offering valuable application management features. However, the potential exists to explore monetization strategies that would allow for

continuous development and expansion of Applify's functionalities. Implementing a premium subscription model could offer a viable solution. Premium subscribers could benefit from additional features like advanced filtering options for applications, automated notifications for application status updates, or increased data storage capacity to manage a larger volume of job applications.

- **Unlocking Automation Potential:**

Applify currently relies on user input for application status updates. While this approach offers clear benefits for user control, exploring the possibilities of automatic status updates could further enhance the platform's value proposition. Investigating the feasibility of integrating APIs with specific job boards could potentially enable Applify to retrieve current application statuses automatically. This automation would require careful consideration of job board terms of service and potential collaborations with these platforms to ensure adherence to their data access policies.

- **Expanding Website Coverage:**

One of Applify's core strengths lies in its ability to manage applications across various job boards. To further extend its reach and cater to a broader user base, expanding scraping functionality to encompass a wider range of job websites is crucial. By supporting a more diverse set of job boards, Applify would become a truly comprehensive application management solution for users seeking opportunities across different industries and sectors. However, this expansion must prioritize legal considerations.

- **Addressing Legal Implications:**

Many websites have terms of service that restrict web scraping practices. Applify's developers should carefully research and ensure compliance with all relevant legal implications. In certain cases, reaching out to website owners and obtaining explicit permission for scraping their data might be necessary. A commitment to responsible data collection will bolster Applify's credibility and user trust.

- **Additional Considerations:**

Beyond the specific recommendations outlined above, additional areas merit attention as Applify evolves. Implementing unit tests for critical functionalities like scraping and data storage would enhance its reliability by identifying and mitigating potential bugs. Additionally, focusing on user interface (UI) improvements would further enhance user experience by

providing a clear, intuitive, and visually pleasing platform for managing applications, data visualization, and export/import functionalities.

By implementing these recommendations, Applify has the potential to transform from a valuable tool into a comprehensive and efficient job application management platform. Embracing scraping efficiency, robust security measures, and innovative monetization strategies can position Applify for long-term success. Further exploration of automation possibilities, website coverage expansion, and strict adherence to legal considerations will solidify Applify's position as a trusted and effective resource for users navigating the job market.

CHAPTER 6

REFERENCES

- [1] <https://huntr.co/product/job-tracker> [Accessed on 7 January 2024]
- [2] <https://www.jobscan.co/job-tracker> [Accessed on 8 January 2024]
- [3] <https://www.tealhq.com/tools/job-tracker> [Accessed on 9 January 2024]
- [4] https://www.google.com/intl/en_in/chrome/ [Accessed on 15 January 2024]
- [5] <https://firebase.google.com/docs/rules/rules-and-auth> [Accessed 21 January 2024]
- [6] <https://aws.amazon.com/what-is/api/> [Accessed on 25 January 2024]
- [7] <https://www.mulesoft.com/resources/api/what-is-an-api> [Accessed on 28 January 2024]
- [8] <https://www.ibm.com/topics/api> [Accessed on 7 February 2024]
- [9] <https://en.wikipedia.org/wiki/URL> [Accessed on 14 February 2024]
- [10] <https://flask.palletsprojects.com/en/3.0.x/> [Accessed on 24 February 2024]
- [11] <https://www.simplilearn.com/tutorials/selenium-tutorial/what-is-selenium> [Accessed on 29 February 2024]
- [12] <https://firebase.google.com/docs/firestore> [Accessed on 4 March 2024]
- [13] <https://en.wikipedia.org/wiki/Redis> [Accessed on 12 March 2024]
- [14] <https://expressjs.com/> [Accessed on 18 March 2024]
- [15] <https://en.wikipedia.org/wiki/Express.js> [Accessed on 20 March 2024]
- [16] https://en.wikipedia.org/wiki/Comma-separated_values [Accessed on 25 March 2024]
- [17] <https://flatirons.com/blog/the-ultimate-guide-to-csv-files/> [Accessed on 27 March 2024]
- [18] <https://www.interaction-design.org/literature/topics/> [Accessed on 28 March 2024]
- [19] https://en.wikipedia.org/wiki/User_interface [Accessed on 04 April 2024]
- [20] <https://developer.mozilla.org/en-US/docs/Web/HTML> [Accessed on 07 April 2024]
- [21] <https://en.wikipedia.org/wiki/HTML> [Accessed on 09 April 2024]
- [22] <https://en.wikipedia.org/wiki/CSS> [Accessed on 14 April 2024]

[23] <https://developer.mozilla.org/en-US/docs/Web/CSS> [Accessed on 15 April 2024]

PERSONAL DETAILS

1. Name: Vani Seth

Er. No.: 201B299

Mobile No.: 7906966199

Email id: 201b299@juetguna.in, sethvani2002@gmail.com



2. Name: Vinayak Rao Dikshit

Er. No.: 201B308

Mobile No.: 8120215502

Email id: 201b308@juetguna.in, vinayakrdikshit@gmail.com



3. Name: Virad Chaurasia

Er. No.: 201B354

Mobile No.: 9559028632

Email id: 201b354@juetguna.in, iamvirad09@gmail.com

