PROJECT REPORT

CLOUD COMPUTING CSCI 5409

GROUP: 03

Misha Herasimov
Sneha Jayavardhini Doguparthi
Yuganthi Krishnamurthy
Ruize Nie
Ram Prasath Meganathan
Pratheep Kumar Manoharan

Table of Contents

Scenario for Business Context and High-level Project Plan

High-level scenario

We have incorporated the same scenario for our project, which is used in assignment 6 and 7. It is based on the communication between three companies, i.e. Company X, Company Y and Company Z.

Company X is responsible for managing the Jobs and with the help of their web service, user can do the CRUD operations for Company X. While creating a job, the list of parts from company Y are listed. Company X web service also has the option to view all the orders which are ordered by all the users. This order information is received from Company Y.

Company Y is responsible for maintaining Parts information. It maintains a web application that allows the user to do CRUD operations on parts. It sends information about the list of parts to company X and receives the information from Company Z when an order is placed. This company will minimize the number of ordered parts after a request is received from Company Z.

Company Z is responsible for allowing the user to place an order. It has a web application that enables the search of Jobs and enables the user to place an order based on the availability of jobs and parts. This company receives information regarding Jobs and Parts from companies X and Y. When a user places an order, the request is sent to both X and Y to verify some information and keep the order data. The user is first authenticated and then allowed to place an order, and he/she can place the order only once. After the order is successfully placed, a message is shown to the user mentioning the success. Similarly, the failure message is also displayed if any error has occurred.

Functional requirements

Company Y has to offer the listing of the available parts. User has to be able to submit a new part or edit the existing one. Each part should consist of three values: part ID, part name and the quantity of the available parts. Company Y has to let a user see the orders submitted by Company Z. The user also should be able to filter the list of orders. The public API has to expose all the endpoints used by the website. Additionally, it has to support distributed transaction architecture. Company Y backend has to be able to prepare the XA transaction and offer the ability to commit or rollback the transaction.

Company X, in its turn, has to let the user make full CRUD operations on the job information. It has to list the jobs in the table form. The job data has to consist of the job name, part id and the number of parts required for the job. The user should be able to filter jobs by its name. The creation form should let users compose a new job using the predefined set of parts given by Company Y. The user should not see the ID of any values. The information must be communicated in a human-readable format.

Similarly to Company Y, Company X has to collect information about the orders. The company should be able to participate in the distributed transaction by exposing relevant API. The website functionality has to be offered over the public API. Aside from the CRUD operations, Company X has to allow the job search and the list of relevant parts.

Finally, Company Z has to serve as an intermediary between Company X and Y. It should allow users of the website to search for jobs by providing a job name in the search field or listing them in the table. It should be possible to select particular parts for any job that needs to be ordered. Company Z has to keep the log of the searched as well as all successful orders. Company Z has to manage the XA transaction for the order flow. It has to gather information about rather the job still exists, and the storage for parts can accommodate a new order. Only if operations in Companies Y and X end successfully the Company Z can finish the order flow. Each order has to be assigned to a particular user. The order can only be placed after the user authentication. The protected operations need to be conducted using Single Sign-On protocol. Company Z should not have any public API available for external usage.

Non-functional requirements

Regarding the atomicity of each transaction in the database, it must either fully happen, or not happen at all. We implemented an XA transaction using functionality provided by MySQL to enforce that transactions between those companies will not be partial. When the company Z wants to place an order, it starts the distributed transaction and requests processing of the order in company X and Y. After all the verifications (for example, rather job still exists or is there enough parts for a job), the companies respond with the prepared state confirmation. If there is any error, then the transaction will be stoped, and all the databases involved will be rolled back to the previous state; otherwise, changes will be committed.

Below is a list of non-functional requirements that we wished to enforce.

1. Scalability: As the customers of the company are growing, the workload of the website will increase; Our app should use scalable solutions such as Google Cloud Function to deploy the backend of the application, it will automatically scale out when the load grows.
2. Security: As the company Z requires the user to log in and perform some authentication and authorization, we use the JSON web token technology to enforce that the user's sensitive data will not be stolen by other malicious software.
3. Availability: As all the websites are hosted on AWS and GCP, it will function 24/7/365.
4. Usability: We provide a friendly user interface for all the companies; users of these companies can create the parts, jobs and orders easily without any preconquest study.

Deficiencies

There are no deficiencies all the requirements are satisfied.

Meritable Features/Properties

All the functional requirements are met except for the project scenario; we have decided to use the group assignments' existing idea. Information for all the companies X, Y and Z are stored in three different databases. The Sequelize ORM was used to access the data from the database in Company X. Additionally, we implemented a Single Sign-On in Company Z for the order submission [1]. The user can filter jobs and order in companies X and Y. The distribution transaction requirement was completed in full. We used distributed transactions and two-phase

Commented [SJD1]: Description of any deficiencies related to the assignment's functional requirements.

Optionally, in a separate section, brief description of what you did in excess of the basic requirements or which features you deemed to be deserving merits. Use this to highlight what you did particularly well or in excess of the requirements. In particular, if you are proud of your work and feel it deserves more than satisfying requirements (B+), highlight here why.

commit protocol for the creation of the order. The webpages have a pleasant user interface; we used Bootstrap to create a presentable web page. For Company Z, we have added a search history page and Ordered jobs page where users can see their already placed orders and view the content they have previously searched.

Software Overview

Commented [SJD2]:
High-level overview of your software. Overall description highlighting your system architecture/organization while describing your major software components. For example:
- Overview of your front-end (e.g., webpages used and how hosted)
- Web-services, their end-points, and how hosted
- Workflow coordination description

Company X

The infrastructure of company X is built using an Angular framework for the frontend and Express.js for the backend. From company X, the user can make all the CRUD operations for the job data. This company uses AWS RDS to store the data, and the backend uses ORM to access the information from the database.

The backend for Company X was deployed in a docker container on Heroku, and the frontend was hosted on the AWS S3 bucket [2, 3, 4, 5, 6]. The company X frontend can be accessed via the link: http://new-bucket-with-permisstions.s3-website-us-east-1.amazonaws.com. The company X backend API can be reached with a link: https://enigmatic-everglades-12100.herokuapp.com.

End Points

Table 1 endpoints of company X

S. No	Endpoint name	Endpoint	Description
1	/api/jobs	https://enigmatic-everglades-12100.herokuapp.com/api/jobs	To get, create, update, and delete the jobs
2	/api/jobList	https://enigmatic-everglades-12100.herokuapp.com/api/jobList?jobName=job4	To get a job by its jobname
3	/api/jobById	https://enigmatic-everglades-12100.herokuapp.com/api/jobById?jobName=job5&partId=4	To get a job by its id<
4	/api/orders	https://enigmatic-everglades-12100.herokuapp.com/api/orders	To get, create the order
5	/api/orders/finish	https://enigmatic-everglades-12100.herokuapp.com/api/orders/finish	To post the order which is ordered by user

The interface of Company X is shown in Fig 1. As the user opens the application, the list of jobs is visible along with associated parts and quantity. Users can manipulate the existing data by clicking on 'Delete' and "Edit" buttons in each row to delete and update the jobs. The user can create a job by entering the job name, selecting the parts from the available list, and then clicking on "Create Job". To view all the orders, the user can click on the "View Orders" button and view all the orders which are placed by the users.

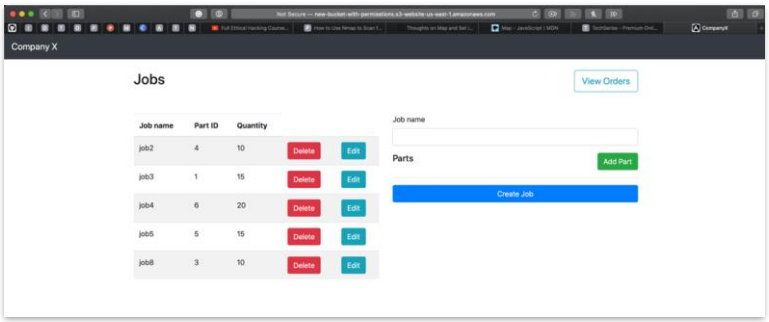

Figure 1 Company X main page

The order page shows a table of all orders submitted by Company Z. The visual interface of this page is shown in Fig. 2.

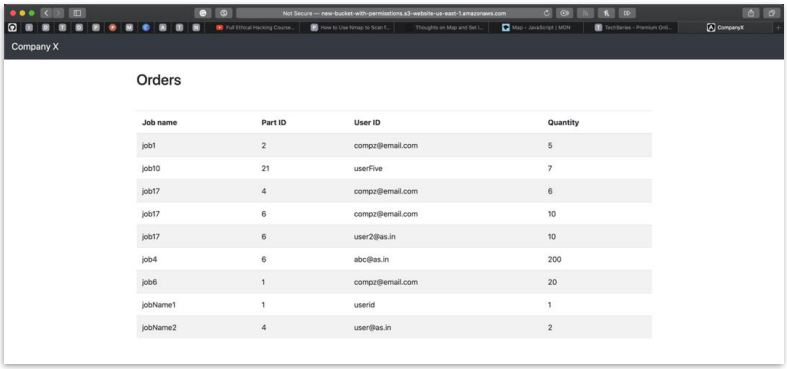

Figure 2 List of order items page

The last page offered by Company X is the editing job. A user is blocked from updating part name or job name.

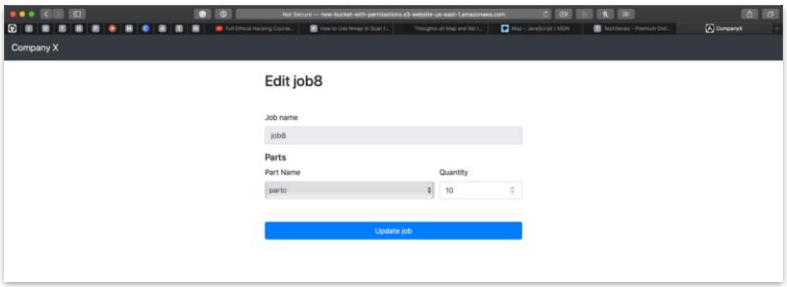

Figure 3 Edit job page

The workflow for such processes as “Show main page,” “Delete job,” and “Create job” are given in Fig. 4.

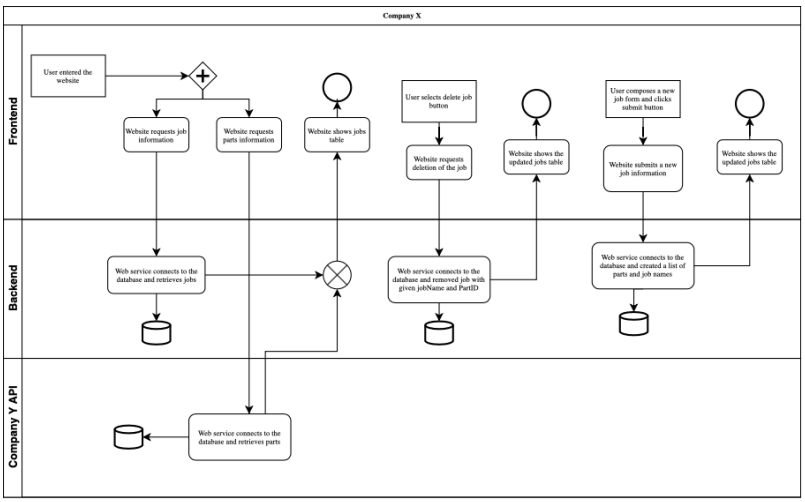

Figure 4. Main page workflow diagram

The “Edit job” workflow diagram is given in Fig. 5. The process is related to the edit job webpage of the Company X website. It consists of two main parts—loading the parts for the single job and updating the actual information.

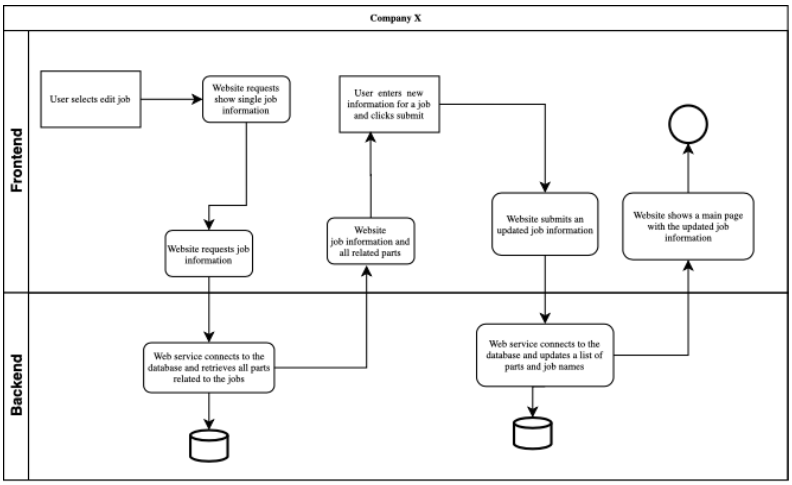

Figure 5. Edit job workflow diagram

The diagram in Fig. 6 demonstrates processes that take place during the submission of an order. The workflow diagram covers the part of the distributed transaction for Company X.

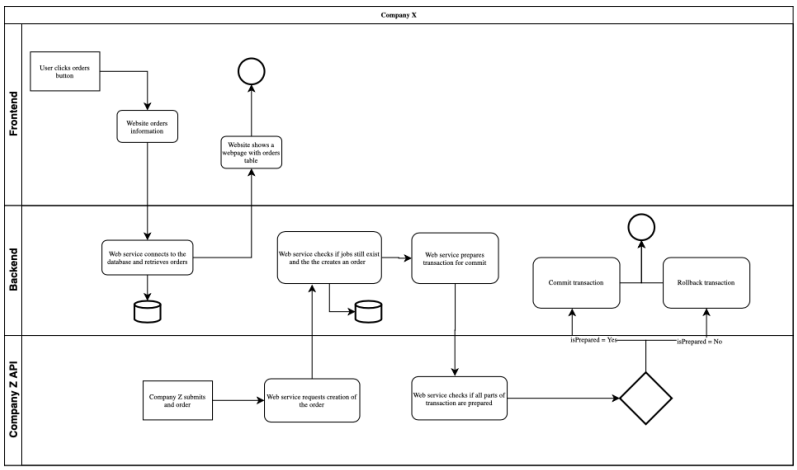

Figure 6. Workflow diagram for the order submission in Company X

Company Y

Company Y is responsible for handling parts. The front-end of Company Y was developed using React.js. The backend was created using Express.js. The website was hosted in the S3 bucket. However, the web service was deployed to the Google Cloud Functions [7, 8].

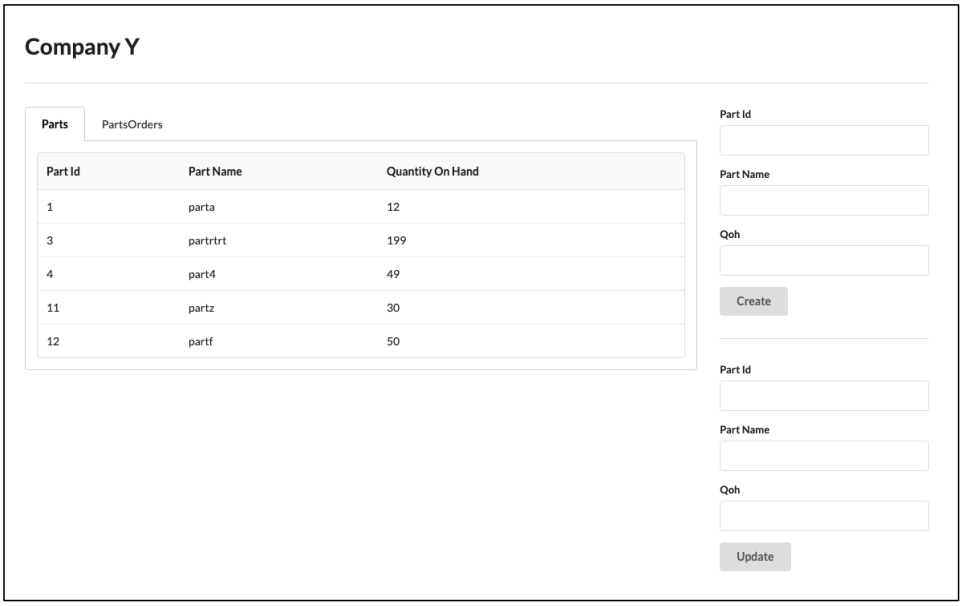

Figure 7. Company Y main page

The company Y's home page is shown above; the website's left panel is a dashboard that displays all the available parts from the database. The right panel of the website is a section in which the user can do some CUR operations for the parts. When the user wants to create a new part, it requires the user to enter the part ID, part name and quantity. After that, the request is sent to the back end of the company Y; it will validate the information. If it encounters any error, the error message is shown below the create button, which indicates the user what to perform next.

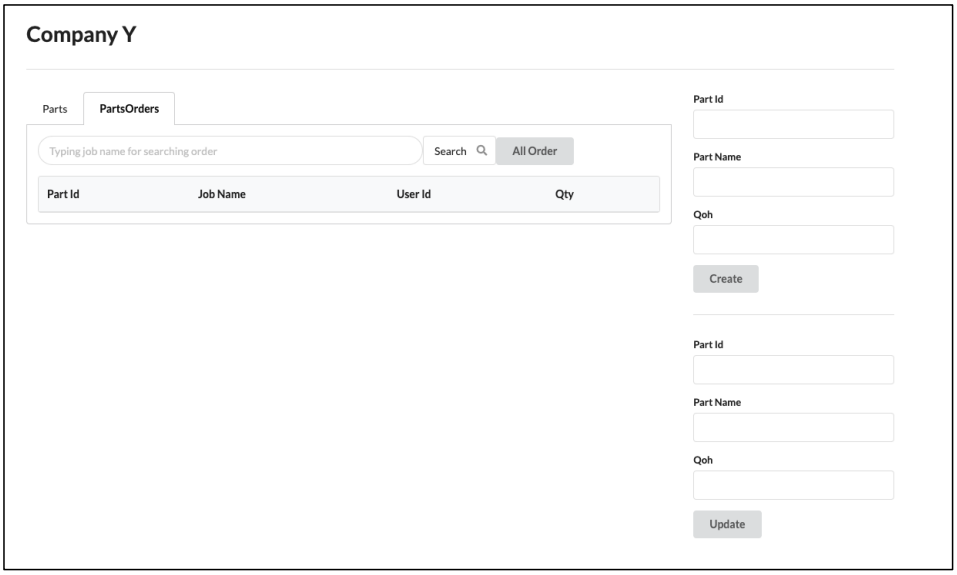

Figure 8. Company Y main page, orders tab

The user can change the tab from parts to orders the check the currently submitted orders. We have also provided the user with a basic search bar to filter the result by the keyword. When the user clicks the All Order button beside the search button, all the orders will be retrieved from the database. The front-end and the back-end URL of Company Y are listed below:

- Frontend: http://a7companyy.s3-website-us-east-1.amazonaws.com
- Backend: https://us-central1-testproject-277421.cloudfunctions.net/cloudproject_compY

Three main processes take place in company Y—create a new part, display the list of available parts and update the part. The detailed description of the activities that take place while the execution of processes is shown in Fig. 9.

Since the Company Y also performs the distributed transaction, the order submitted by Company Z is conducted in two stages. In accordance with the two-phase commit protocol, Company Y will first perform the order submission and then wait for the confirmation. After that, it commits or rolls back the transaction. The illustration of the flow is demonstrated in Fig. 10.

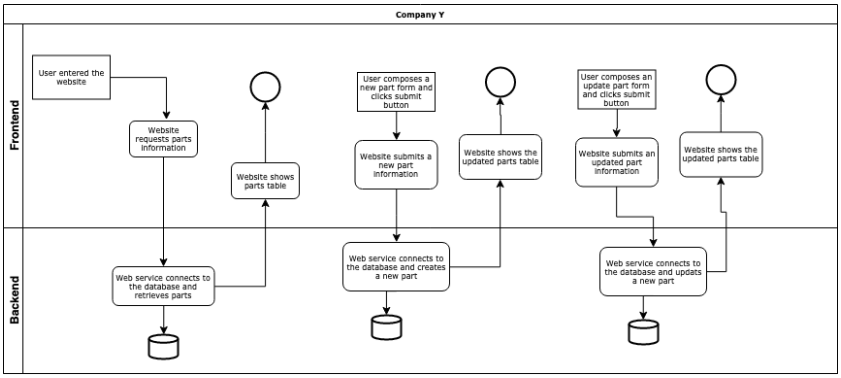

Figure 9. Main page workflow diagram for Company Y

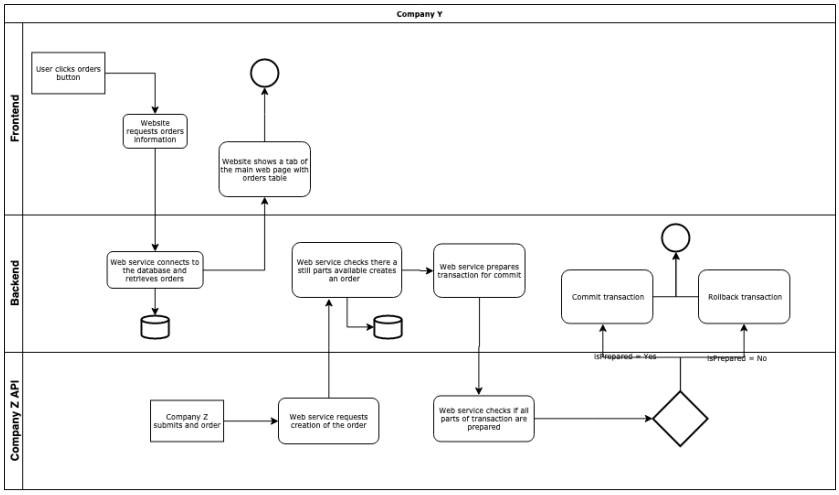

Figure 10. Workflow diagram for the order submission in Company Y

Endpoints

Table 2 Endpoints for company Y

S. No	Endpoint name	Endpoint	Description
1	/parts	https://us-central1-testproject-277421.cloudfunctions.net/cloudproject_compY/parts	Return all the parts from the parts DB
2	/parts/:id	https://us-central1-testproject-277421.cloudfunctions.net/cloudproject_compY/parts/:id	Return specific part with id from the parts DB

3	/parts/create	https://us-central1-testproject-277421.cloudfunctions.net/cloudproject_compY/parts/create	Create new part and then store the data into parts DB It will check if the part exists or not Parameter need to specify when creating new part: partId, partName, qoh
4	/parts/update	https://us-central1-testproject-277421.cloudfunctions.net/cloudproject_compY/parts/update	Updates part and then store the data into parts DB It will check if the part exists or not Parameter need to specify when update part: partId, partName, qoh
5	/order	https://us-central1-testproject-277421.cloudfunctions.net/cloudproject_compY/order	Returns all the partsOrder from the partorderY DB Displayed in sorted order, first by jobName, then by userId, and then by partId
6	/orders	https://us-central1-testproject-277421.cloudfunctions.net/cloudproject_compY/orders	Creates new partsOrder and then store the data into partorderY DB Parameter need to specify when create new orders: transactionName, order: [{jobName, partId, userId, qty }]
7	/orders/finish	https://us-central1-testproject-277421.cloudfunctions.net/cloudproject_compY/orders/finish	Creates new partsOrder and then store the data into partorderY DB Parameter need to finish orders: operationType, transactionName

Company Z

The frontend and backend of Company Z are created as two different modules. The communication between the modules is done securely using the REST API calls. The frontend of the application is developed using the React.js framework. Since this is an extension of Assignment 6 & 7, we have added a reference to both assignments [7, 9, 10].

The frontend user interacting pages of the company Z is added below. Fig. 13 shows the landing page of the company Z application. We have a short description on the landing page taken from the assignment requirement document. The landing page has a navigation bar that can take the users to the Search page and list all the jobs page. The two pages have a REST API calls to company X to fetch the job details. The jobname from company X will be listed on the screen. The user must click on the jobname that needs to be ordered.

All the activities mentioned above can be done without authentication. Any user with an application URL can do. On click of jobname in Search page or Get All Jobs page, the application will take the user to the login page. In the login page, the user must provide a valid username and password to proceed. The valid user credentials will take the user to the homepage of the application. On the homepage, we have the order page and search history page. The order page contains all the details about the job, parts, available quantity, and required quantity. The search history page shows the recent searches done by all users in the application. The order page fetches part details from Company Y. The user must select the parts that they want to order and click the order button. This will trigger three APIs one to each company's backend to store the order details. The front end is deployed in Elastic Bean Stalk, and the URL is http://cloudprojcompzfrontend-env.eba-rccpqxvg.us-east-1.elasticbeanstalk.com/, and the backend is deployed in Google Cloud Function, and the deployed URL is https://us-central1-cloudprojects-279901.cloudfunctions.net/companyZ/[11].

Fig. 11 shows the architecture of Company Z application. The drawing is created using Edraw max tool online [12].

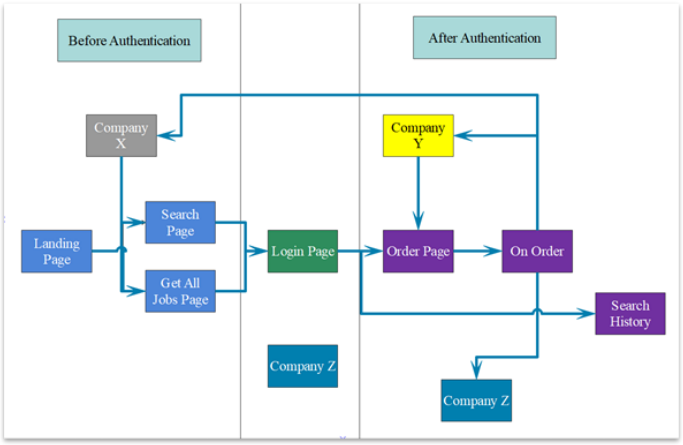

Figure 11. Architecture of Company Z [2]

The backend of the application is developed using the Express.js framework, which is used to create the server and API requests to be called from the frontend. Node MySQL library is used to connect the Node.js backend to the MySQL instance of the AWS RDS database. The following are the libraries used in the backend of the application [3, 4, 7].

Company Z uses three tables mainly to perform CRUD operations

1. Search – used to store the search history of the user
2. User - stores the credentials of the user for authentication
3. JobParts - stores the details of the JobParts that are ordered through the application

Two-phase commit flow for company Z

The two-phase commit flow of company Z is built using XA transactions in such a way that the commit is successful only when all three companies are updated; otherwise, rollback is performed. In each phase, the previous company result is fetched and checked if the result is successful else the user is notified that the result is a failure [13].

Fig. 12 shows the commit flow from Company Z's perspective for all three companies.

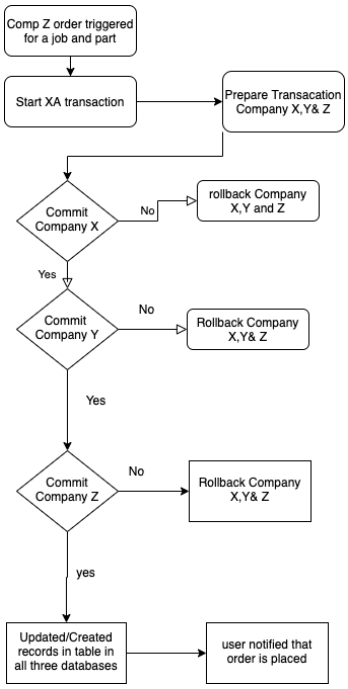

Figure 12. Two phase commit flow for Company Z

Fig. 13 shows the landing page of the application. This is the first page that a user interacts with the application.

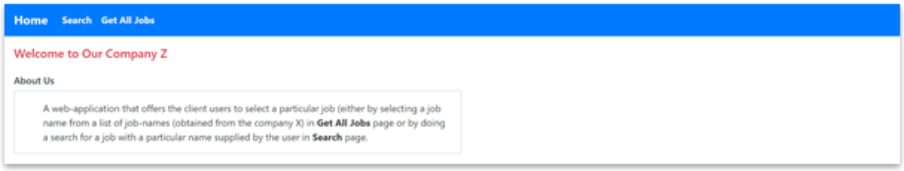

Figure 13. Landing page of Company Z

Fig. 14 shows the search page of the application. The user must enter the job name and click on search to fetch the details from Company X. The search will work even if the user tries with part of job name.

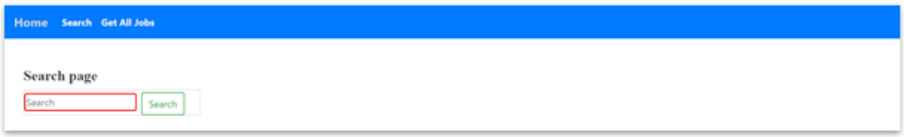

Figure 14. Search page of Company Z

Fig. 15 shows all job names from Company X. The user must click on the required job name and this will take the user to login page.

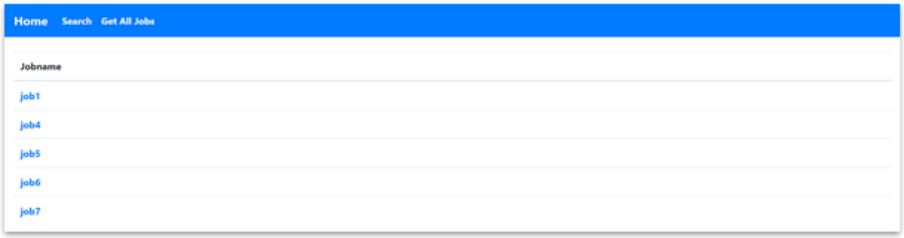

Figure 15. Get all jobs page of Company Z

Fig. 16 shows the login page where the application will authenticate the user.

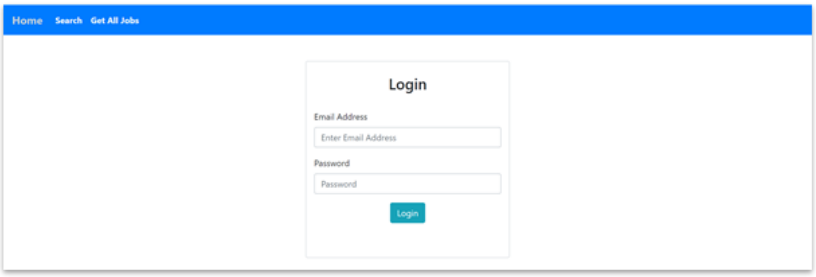

Figure 16. Login page of Company Z

Fig. 17 shows the Order page of the application. The order page fetches the job details from Company X and part details from Company Y. The order page has below logics logic built-in.

- One user cannot order the part that was already ordered.
- User can order any number of parts.
- The order details will be posted to all three companies.
- The minimum frontend validations like required quantity and available quantity comparison are done.

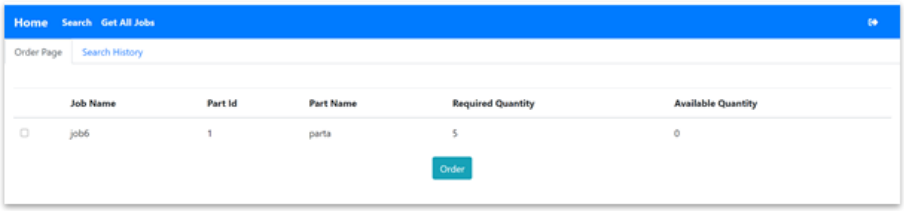

Figure 17. Order page of Company Z

Following are the end points created for Company Z

Endpoints

Table 3 Endpoints for company Z

S. No	Endpoint name	Endpoint	Description
1	/api/jobs/:jobName	https://us-central1-cloudprojects-279901.cloudfunctions.net/companyZ/api/jobs/:jobName	Insert the search history in the database
2	/api/users	https://us-central1-cloudprojects-279901.cloudfunctions.net/companyZ/api/users	Authenticates the user request
3	/api/getOrder	https://us-central1-cloudprojects-279901.cloudfunctions.net/companyZ/api/getOrder	Gets the orders made by the users
4	/api/updateOrderusing2pc	https://us-central1-cloudprojects-279901.cloudfunctions.net/companyZ/api/updateOrderusing2pc	Two-phase commit call for ordering
5	/api/getOrderedJobs	https://us-central1-cloudprojects-279901.cloudfunctions.net/companyZ/api/getOrderedJobs	Get ordered jobs from the database

The main tasks for company Z are searching for the job, authentication of the user and submission of the order. The user can search for jobs using two options. First, one by entering the job name into the search bar or by viewing all jobs in the table. These cases are shown in Fig. 18.

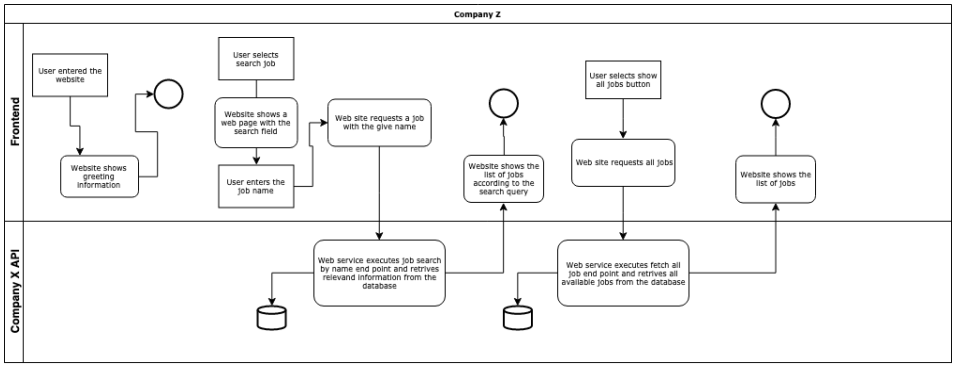

Figure 18. Search jobs workflow

The next important part of the workflow is authentication. The process is quite straightforward. User needs to provide their own credentials, and then a user gets access to the placing of the order. This process is given in Fig. 19.

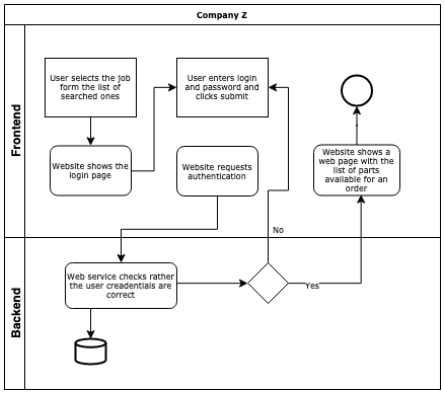

Figure 19. Authenticate user workflow

The most important part of Company Z is the submission of the order. It is implemented using an XA transaction and two-phase protocol. The full description of the actions that take place during submission is given in Fig. 20.

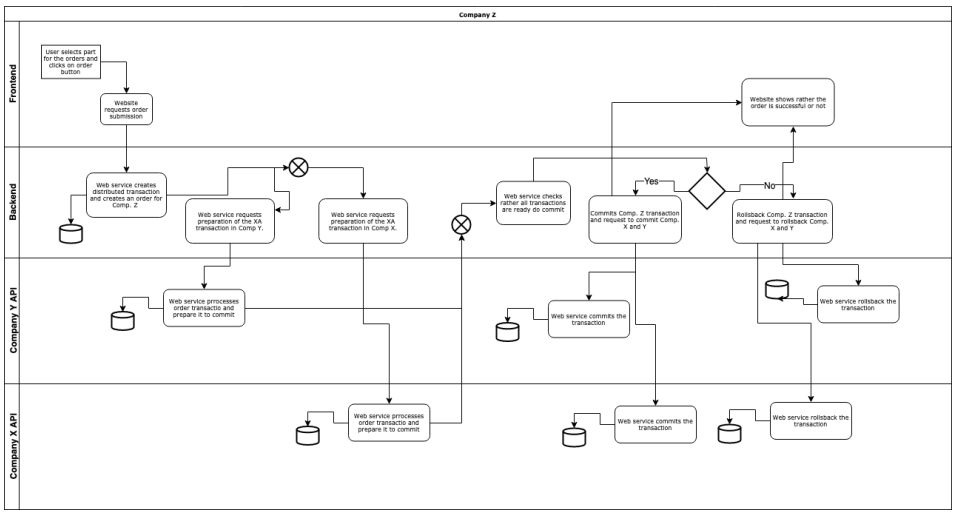

Figure 20. Submit order workflow

Collaboration Summary

Our means of communication was MS Teams; we scheduled the meetings as per everyone's availability and discussed the work to be performed in each meeting. All the meeting discussions and action items are noted in the minutes' document. In the initial meeting, we discussed the project scenario, and since we were not sure about it, we have decided to have another meeting to finalize the idea. In the 2nd meeting, we explored the possible solutions to decide the project and finalized it to continue with the existing business scenario of Assignment 6 and 7. This decision was taken by exploring each teammate's availability as it is the end of the semester, and everyone was engaged in final exams. It was decided by taking a poll, and all the members agreed to the decision. The GIT repository and Trello board is created in the first meeting; all the tasks are decided and assigned to every individual as per their interest. During development, if they were any technical problems, we had short meetings to resolve the issues. All the team members cooperated with the decisions taken and completed the project successfully.

Summary of Minutes of Meetings

Table 3. Meetings summary for the project

Meeting Name	Date and Time	Topics considered	Absentees
Meeting 1	27th July 2020 2 PM	1. Discussed the project description 2. Created the questions for clarifications and sent an email to Professor.	None

Commented [SJD3]: Your group/team will collaborate to make informed decisions, plans, and activities in completing your assignment while using the collaboration tools, namely, your code repository, MS Teams, Trelllo, and minutes of meetings (to be stored on MS Teams) - collectively, these repositories and the minutes of the meetings will be referred to as the ***C****ollaboration* ***I****nformation* ***(CI)***. The intention/objectives of this summary is to provide guidance/narrative to your aluator/marker in reviewing your activities and contributions as they appear in your CI in order to form a view of how well your team functioned in tackling various issues that arose and making necessary adjustments. Without such a summary/narrative, it may be difficult for the evaluator to form an overall view as the related information is stored in separate repositories. Your summary should be at a sufficient level of abstraction to highlight the major decisions and the reasons why they were taken and which option were considered – so that your evaluator will not get "bogged down" in details in forming the overall picture of your team's functioning. In addition, in separate subsections, you should also summarize you minutes of meetings, and trello board as described below.

Commented [SJD4]:
Present the summary of your minutes in a table format, in which each row represents one
meeting and columns include Date, Topics considered, List of absentees (identified by initials … of course, you need to list the members or and their initials somewhere in the table's annotation).

		3. Git repository and Trello board are created for Project. 4. Discussed and action plan for future work.	
Meeting 2	29th July 2020 9 AM	1. Decided to request for an extension 2. 2 project ideas are discussed 3. Initial discussion on XA transaction is done we decided to explore more on XA transactions. 4. The workload for custom project implementation and assignment modification is compared. 5. Decided to use the existing scenario of assignment 6 and 7. 6. Created a preliminary work plan	None
Meeting 3	2nd Aug 2020 11 AM	1. XA transaction implementation 2. Project scenario to be further discussed 3. Company X uses ORM to access the information from database. 4. The backend for company X, Y, Z will be updated for XA transactions. 5. Decided to keep the frontend of companies X and Y in AWS S3 storage.	None
Meeting 4	2nd Aug 2020 9 PM	1. Discussed the project scenario to be considered for the implementation based on a poll and decided the project to be similar as assignment 7. 2. Distributed the workload based on the availability of each team member. 3. Regrouped the sub-teams to expedite development.	None

Trello Board Examples

Commented [SJD5]:
Include a screenshot (or maybe two) as of your Trello board at a particular point in time (e.g., end of sprint) and describe briefly its cards/tasks appearing on it. Trello board should record all tasks, not just for coding tasks. For instance, if you need to do some research (e.g., if you know you will have some technical how-to issue), you should include in your plan a task for its research and resolution. If someone is looking after minutes of meetings or organizational tasks, they should also be included.

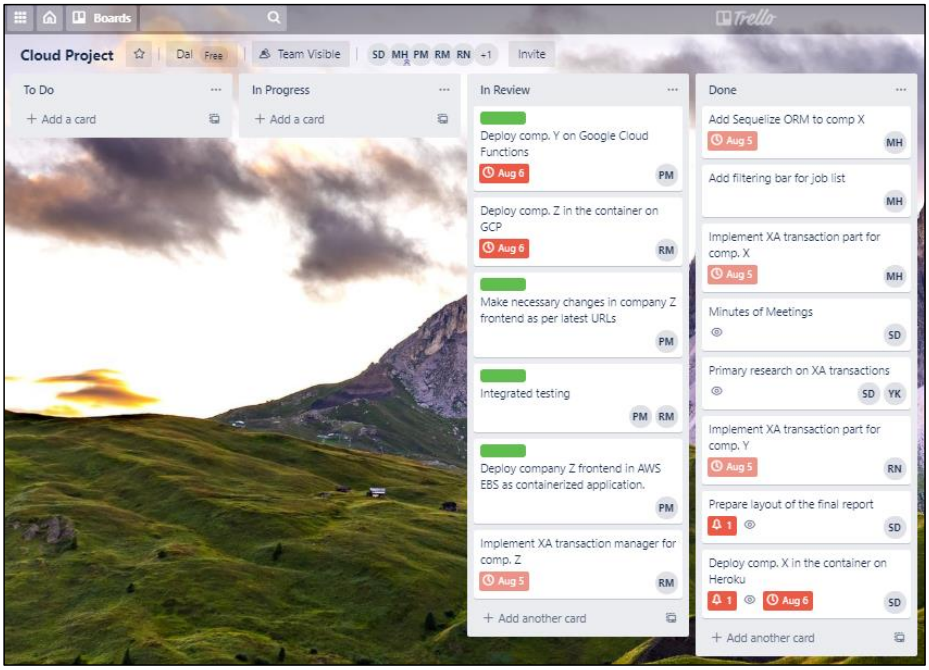

Figure 21. Trello board for Project

Fig. 21 shows our Trello board for the project. It is the screenshot before the end of the sprint. We have added four lists to the Trello board, i.e. "To Do," "In Progress," "In Review," and "Done." The progress of each task is easily tracked with the help of these lists. All the tasks are created as cards and placed in these lists. Each task is updated with a list of subtasks to perform in the project. All the tasks which do not involve coding are also added into the lists. Technical tasks like implementing XA transactions in Company X, Company Y and Company Z are added and assigned to the team members. Non-technical tasks like doing research on XA transactions, creating the minutes' documents, etc., are also added. All these tasks are primarily added in the "To Do" list, and as the development progressed, each team member added the cards in the subsequent lists. We have added the colour code to distinguish the tasks, and the due dates for each card are added in order to keep the team on track to complete the tasks assigned to each team member [14].

Resources

Cloud Technologies

Commented [SJD6]: (1.Descripton of cloud resources and their purpose much as RDS is manages database for 3 companeis…

Company X

Company X web application frontend is an Angular application; it is locally built and uploaded to the S3 bucket. The backend uses AWS RDS to store and manage the data of the application. ORM is used to access the information from RDS. Backend is deployed in a docker container, and it is hosted on the Heroku cloud [1, 4, 6].

Company Y

Company Y front end was deployed using the S3 bucket. The backend is deployed in the Google Cloud Platform. The AWS RDS was used to store the data [4, 5, 8].

Company Z

Company Z frontend is deployed using AWS Elastic Beanstalk, backend in Google Cloud Platform and AWS RDS is used to store the data [4, 8, 11].

Libraries & Frameworks

Commented [SJD7]: 1.List a libraries used for each company and descripe their purpose in perspective to the implentation…

Company X

Table 4 Libraries use for the implementation of Company X

S. No	Library	Purpose
1	sequelize	It is a Relational Object Mapper that accesses the MySQL database and is used in Node.js environment. [1]
2	express.js	Express.js is node js package provided for creating endpoints to perform CRUD operations in the database [3]
3	mysql	MySQL package is used in Express.js to establish a connection with the database [15]
4	express-validator	express-validator is a set of Exress.js middleware's that wraps Validator.js validator and sanitizer functions. [16]
5	cors	CORS is a Node.js package for providing a Connect/Express middleware that can be used to enable CORS with various options. [17]
6	nodemon	Nodemon library monitors the changes which are made in the source code and restart the server. [18]

Regarding the frameworks of Company X, frontend is implemented using Angular framework and backend is developed using Node.js and Express.js.

Company Y

Table 5 Libraries used for implementation of Company Y backend

S. No	Library	Purpose
1	express.js	Express.js is Node.js package provided for creating endpoints to perform CRUD operations in the database [2].
2	mysql	MySQL package is used in Express.js to establish a connection with the database [16]
3	firebase-functions	This package is used to deploy the backend of Company Y to Google Cloud as a serverless function [19]
4	cors	CORS is a node.js package for providing a Connect/Express middleware that can be used to enable CORS with various options. [17]
5	axios	Axios is a promise-based HTTP client used to send asynchronous HTTP requests to backend endpoints from the frontend. [20]

Company Z

Table 6 Libraries used for implementation of company Z frontend

S. No	Library	Purpose
1	axios	Axios is a promise-based HTTP client used to send asynchronous HTTP requests to backend endpoints from the frontend. [20]
2	react	React framework library used to render the component and update the state. [7]
3	react-bootstrap	Styling library is used to add styles, effects, and effectiveness.
4	react-dom	The react-dom package is used at the top level of the app and as an escape hatch to get outside of the React model.
5	react-router-dom	The routing between pages, history maintenance, parent and child page relations are managed using this library.
6	react-scripts	The build and deployment of the app are done using this library. [21]

Table 7 Libraries used for implementation of company Z backend

S. No	Library	Purpose

1	express.js	Express.js is the Node.js package provided for creating endpoints to perform CRUD operations in the database [3].
2	mysql	MySQL package is used in Express.js to establish a connection with the database [15].
3	jsonwebtoken	This library is used to generate the authentication token for providing to the frontend for authenticating and maintaining the user session [22].
4	body-parser	Body-parser is used to read the body of the request and convert it to JSON for further operations [23].
5	firebase-functions	This package is used to deploy the backend of Company Y & Z to google cloud as a serverless function [19].

Code Snippets and Resources

For the implementation of the project, we used code created by us in Assignments 6 & 7. We also reference official pages of various libraries that guided our implementation (Angular, Express, MySQL, Bootstrap, React.js). As for the overall ideas, we reference Udemy tutorials and project submissions of the Advanced Web Technologies course at Dalhousie University CSCI5709 [2, 3, 7, 15, 24, 25, 26].

The code or the project can be fetched from the https://git.cs.dal.ca/herasimov/g03-cloud-project .

References

[1] S. ORM, "Sequelize", *Sequelize ORM*, 2020. [Online]. Available: https://sequelize.org. [Accessed: 07-Aug-2020].

[2] *Angular*. [Online]. Available: https://angular.io/docs. [Accessed: 07-Aug-2020].

[3] "Express - Node.js web application framework," *Expressjs.com*, 2017. [Online]. Available: https://expressjs.com/. [Accessed: 12-Jul-2020]

[4] "Amazon RDS | Cloud Relational Database | Amazon Web Services", *Amazon Web Services, Inc.*, 2020. [Online]. Available: https://aws.amazon.com/rds/. [Accessed: 07-Aug-2020].

[5] "Cloud Object Storage | Store & Retrieve Data Anywhere | Amazon Simple Storage Service (S3)", *Amazon Web Services, Inc.*, 2020. [Online]. Available: https://aws.amazon.com/s3/. [Accessed: 07-Aug-2020].

[6] "Documentation | Heroku Dev Center", *Devcenter.heroku.com*, 2020. [Online]. Available: https://devcenter.heroku.com/categories/reference. [Accessed: 07-Aug-2020].

[7] "React – A JavaScript library for building user interfaces", *Reactjs.org*, 2020. [Online]. Available: https://reactjs.org. [Accessed: 07-Aug-2020].

[8] “Cloud Functions | Google Cloud,” *Google Cloud*, 2020. [Online]. Available: https://cloud.google.com/functions. [Accessed: 07-Aug-2020]

[9] “Submission History - CSCI4145 & CSCI5409 - Cloud Computing (Sec 1) - 2020 Summer - Dalhousie University,” Brightspace.com, 2020. [Online]. Available: https://dal.brightspace.com/d2l/lms/dropbox/user/folders_history.d2l?db=77841&grpid=0&isprv=0&bp=0&ou=124054. [Accessed: 07-Aug-2020]

[10] “Submission History - CSCI4145 & CSCI5409 - Cloud Computing (Sec 1) - 2020 Summer - Dalhousie University,” Brightspace.com, 2020. [Online]. Available: https://dal.brightspace.com/d2l/lms/dropbox/user/folders_history.d2l?db=78735&grpid=0&isprv=0&bp=0&ou=124054. [Accessed: 07-Aug-2020]

[11] "AWS Elastic Beanstalk – Deploy Web Applications", *Amazon Web Services, Inc.*, 2020. [Online]. Available: https://aws.amazon.com/elasticbeanstalk/. [Accessed: 07-Aug-2020].

[12] “Edraw Max, All-in-One Diagram Software.” Edrawsoft, www.edrawsoft.com/edraw-max/. [Accessed: 07-Aug-2020].

[13] "MySQL :: MySQL 5.7 Reference Manual :: 13.3.7.1 XA Transaction SQL Statements", *Dev.mysql.com*, 2020. [Online]. Available: https://dev.mysql.com/doc/refman/5.7/en/xa-statements.html. [Accessed: 07-Aug-2020].

[14] "Trello", *Trello.com*, 2020. [Online]. Available: https://trello.com/b/YBFc7SKB/cloud-project. [Accessed: 13-Jul- 2020].

[15] “mysql,” *npm*, 23-Jan-2020. [Online]. Available: https://www.npmjs.com/package/mysql. [Accessed: 07-Aug-2020]

[16] Express-validator.github.io. 2020. *Getting Started · Express-Validator*. [online] Available at: <https://express-validator.github.io/docs/> [Accessed 7-Aug-2020].

[17] npm. 2020. Cors. [online] Available at: <https://www.npmjs.com/package/cors> [Accessed 7 August 2020].

[18] "nodemon", *Nodemon.io*, 2020. [Online]. Available: https://nodemon.io. [Accessed: 07-Aug-2020].

[19] "firebase-functions", *npm*, 2020. [Online]. Available: https://www.npmjs.com/package/firebase-functions. [Accessed: 07-Aug-2020].

[20] "axios", *npm*, 2020. [Online]. Available: https://www.npmjs.com/package/axios. [Accessed: 07-Aug-2020].

[21] “Semantic UI React The official Semantic-UI-React integration.,” *Introduction - Semantic UI React*. [Online]. Available: https://react.semantic-ui.com/. [Accessed: 07-Aug-2020].

[22] “jsonwebtoken,” *npm*, 18-Mar-2019. [Online]. Available: https://www.npmjs.com/package/jsonwebtoken. [Accessed: 07-Aug-2020]

[23] “body-parser,” *npm*, 25-Apr-2019. [Online]. Available: https://www.npmjs.com/package/body-parser. [Accessed: 07-Aug-2020]

[24] - “Managing Transactions (Node.js Application Developer’s Guide) — MarkLogic 10 Product Documentation,” Marklogic.com, 2020. [Online]. Available: https://docs.marklogic.com/guide/node-dev/transactions. [Accessed: 07-Aug-2020]

[25] "NodeJS - The Complete Guide (MVC, REST APIs, GraphQL, Deno)", *Udemy*, 2020. [Online]. Available: https://www.udemy.com/course/nodejs-the-complete-guide/. [Accessed: 07-Aug-2020].

[26] M. Herasimov, S. R. Muppidi, S. J. Doguparthi, R. Kase, and U. V. Chanda, “Foodably: The ultimate recipe assistant,” Foodably: The ultimate recipe assistant. Project work for CSCI-5709 course as Dalhousie University, 06-Apr-2020.