Final Documentation

Project 1: API Back End Development
University of British Columbia Okanagan
COSC499 - Summer 2022

Date: August 16, 2022

Contacts:

Wasi-ul-Hassan Raza(wasiulhassanraza@gmail.com)

Calvin Qu(calvin000128@gmail.com)

Po-Chia Wei(lunch33665566@gmail.com)

Austin Wong(austinwong1@yahoo.ca)

Software Description

HelpYa is a trade service procurement app that connects clients and businesses. Using the platform, customers will be able to search and negotiate services with the business of their choosing. The client would like to design and develop a backend for an iOS app with plans of creating a cross-platform for android and the web in the future. During our time in capstone, our scope was to create an API that can handle login authentication, add, create, delete, update, report user information and handle monetary transactions. We will be doing this by building a RESTful API endpoint that will integrate with the system and perform these functions. The key requirements of the system will be: a database to store values, a way to process payments, login/logout features, messaging, and the endpoints that interact with the system. There are multiple kinds of test files in the software. The endpoint tests are able to test all the endpoints for each table to ensure that all the functions can work properly. The unit tests use a mock-up database to test every independent module to determine if there are any issues when developers build the database. The endpoint tests and unit tests do not require a connection to the actual database. The way to test the interaction between integrated units is integration tests; it connects to the real database and processes dummy data based on different functions.

User Groups

Consumer account

A consumer account will be what every regular user starts off with. Consumers will be able to view businesses, see their ads, send messages and make purchases. Consumers will have the ability to manipulate and edit their account. Lastly, the consumers will have the ability to upgrade to a business account.

Business account

A business account will be used by any business who wants to offer their services to the public. Businesses will have the ability to show off their account as well as create ads the consumers can view and message about. The business accounts will have the same abilities as the consumer account; the only difference is that they will require more information when creating the account and have the ability to create ads to show off their business. A consumer account becomes a business account via signing up for subscriptions.

Admin account

Admin accounts will have access to both the front-end and back-end of the app and will have the ability to edit any data associated with the system. The admins will also have the ability to ban and unban any accounts. As they have access to the back-end, admins will have the ability to delete certain things, such as ads or anything inappropriate shown on the app. Admins also have the ability to refund subscriptions.

HelpYa account

The HelpYa accounts only purpose is to create update reports and delete subscriptions and discount models. HelpYa can also drop the database.

System Architecture

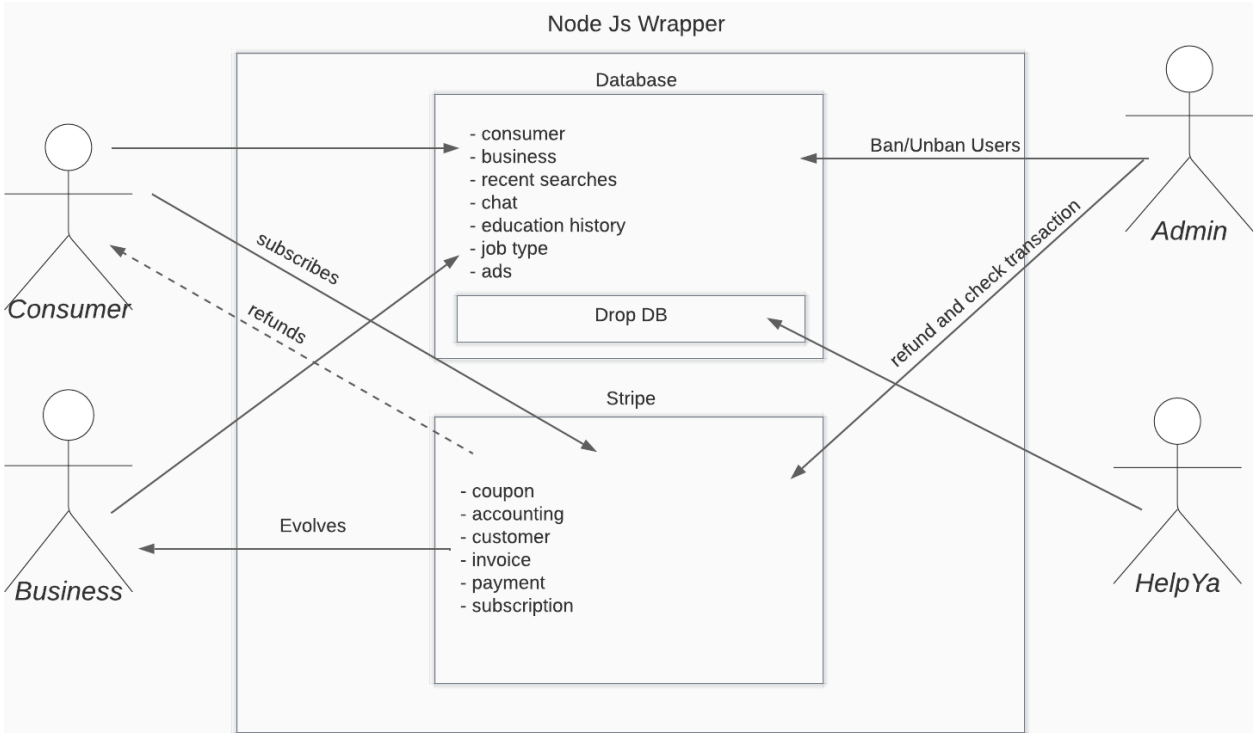

Figure 1: Describes the conditions we have and how our api interacts with the system. Because stripe handles payment, they would be responsible for handling the payment, security and rejecting payment as well, the wrapper created will depend on stripe for these features.

Flow Chart

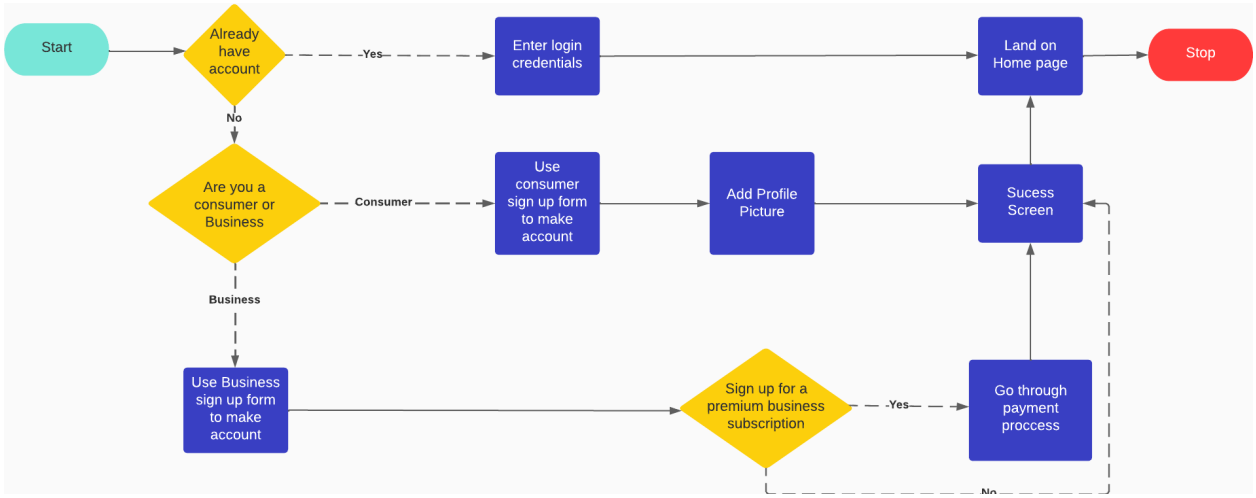

Figure 2: Is a flow chart that shows how the system processes if you already have an account. The flow chart also shows the process of creating an account and the process of upgrading the account.

User Interface

UI Description

This will be a brief description of the different interfaces the users will be able to see. However, this is already going to be done by another party and we are only responsible for the back-end endpoints. An example of an endpoint is localhost:8080/api/consumer followed by an API function, i.e get, post, put, delete. The value of the local host is wherever the server is being hosted, 8080 would be the port name. API states it is using the API function and can be interchanged with “stripe” if the stripe function is being used.

Non-Functional Requirements

- Stripe will handle payments done throughout the app along with storing transactional information.
- Ads will be displayed to the user with pagination
- A route guard is implemented to make sure the correct user is able to use the correct features
- Login credentials are email and password
- There are 4 types of accounts:
    - Consumer account
    - Business account
    - Admin account
    - HelpYa account

Functional Requirements

- The system will search a business using a collection of keywords and name
- The system will be able to delete or tombstone user accounts
- The system will be able to recognise an update to account information and update accordingly
- The system will be able to recognise the different types of accounts upon login
- The system will handle all transactional data and sales data via the Stripe suite and API
- The system will only allow a single HelpYa account
- The system will only allow up to 3 ads for a single business account

Technical Requirements

- The system will use a Node .js framework to retrieve data from a MySQL database
- The system will be hosted by Heroku for continuous implementation
- The system will use Drone by Harness for continuous deployment
- JWT tokens are used for authentication and session handling

User Requirements

- Users will be able to view other user's profile
- Users are able to log in, log out and recover accounts
- Consumer users are able to view ads made by business users
- Consumers will be able to purchase a subscription via the Stripe API that will change their accounts to Business
- Business users are able to create, delete and update ads
- A business will be able to post multiple ads
- Consumer users and business users are able to communicate over a chat feature
- Admin accounts will add delete update user credentials and information
- Admin accounts are able to access the Stripe information dashboard and view, refund and update transactional data
- HelpYa account will be able to drop database
- HelpYa account will be able to add delete or update a subscription model and coupons

Entity Attribute Description

Database ER Diagram

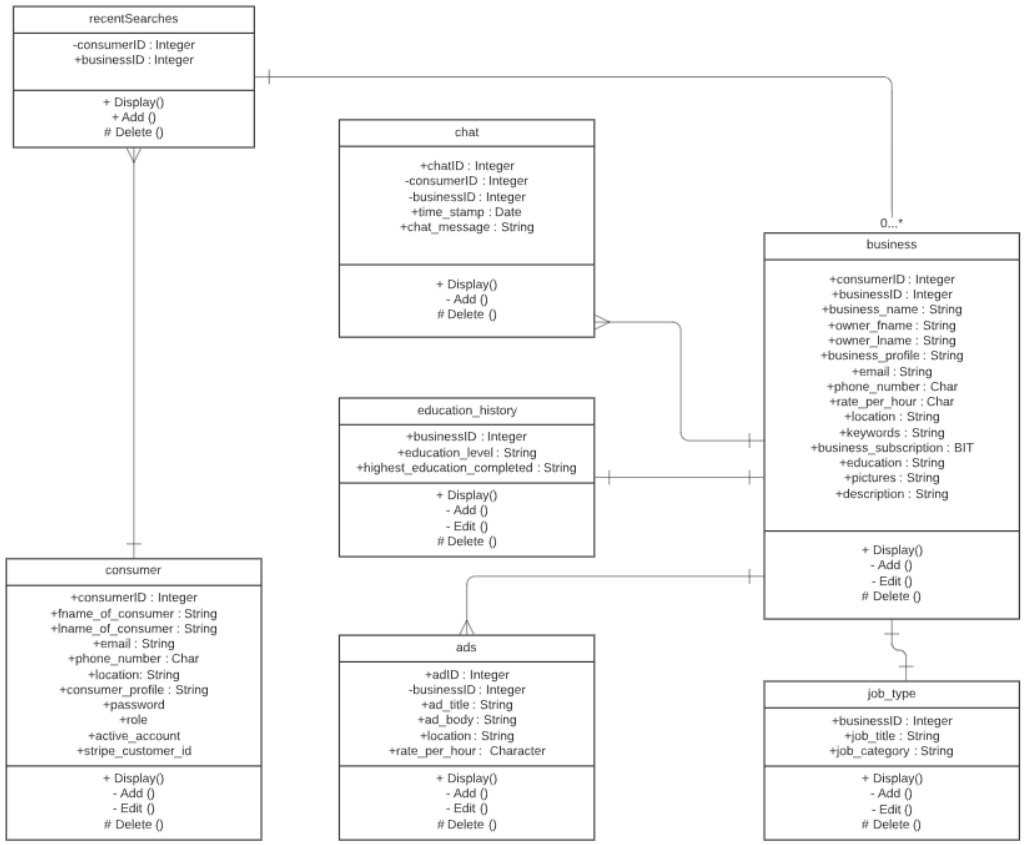

Figure 3: The following database ER diagram shows how data will be represented as in our relational schema.

Consumer

A consumer is one of the clients that will access the app. The consumer will need to create an account and provide the specified information, which includes their name, email, phone number, location, and password. Attributes like a profile picture will be added later.

Attribute	Description
consumerID: INT **[PK]**	Unique identifier given to every consumer. Admins will be able to use this ID to find consumers.
fname_of_consumer: varchar(150)	Attribute holds the first name of the consumer. Has a NOT NULL condition because businesses need a way of identifying the consumer.

lname_of_consumer: varchar(150)	Attribute holds the last name of the consumer. Has a NOT NULL condition because businesses need a way of identifying the consumer.
email: VARCHAR(150)	Consumer's email is used as a way of sending notifications. Because the consumer needs a method of receiving these notifications the value will not be NULL. The attribute also needs to be UNIQUE so multiple accounts will not be created under the same email.
phone_number: CHAR(15)	Attribute will hold the phone numbers of the consumers. A Char value of 15 is given to include international numbers and area codes as well. The phone number will be accessible to the business only after an appointment has been made.
location: VARCHAR(50)	Attribute is used to filter the business within the consumer's general location. This value will not be NULL because businesses with the same location need to be shown to the consumer.
consumer_profile: varchar(250)	Attribute will hold an image link that will have a photo that will be used as a profile picture.
consumer_password: VARCHAR(50)	Attribute will hold the password the consumer sets for their account.
role : VARCHAR(50)	Attribute is used to differentiate consumer users and business users. As these users both start out as a consumer, this will have either the value of consumer or business.
active_account : BIT(50)	This attribute is used to determine if the account is still active or not.
stripe_customer_id : VARCHAR(250)	This attribute's value is given by stripe and is used in conjunction with payment

recent_searches

This table will be used so that the consumer will be able to easily see the business they had recently viewed. They will be able to view things like the name of the business and the business's profile.

Attribute	Description
businessID: INT **[PK]**	Attribute is used to uniquely identify and show the business. In this instance, the attribute is used to save the saved_searches table so that clients can view the recently searched business.
consumerID: INT **[FK]**	Attribute holds the consumerID that the saved_searches is connected to.

Business

A business is one of the users that will access the app. The consumer will need to create an account and provide the specified information, which includes the owner's name, business email, business phone number, location, password, business name, the rate they charge, keywords, education level, pictures and a general description. Attributes like a profile picture will be added later.

Attribute	Description
businessID: INT **[PK]**	Unique identifier given to every consumer. Admins will be able to use this ID to find consumers.
business_name: VARCHAR(150)	Attribute holds the name of the business, which can be viewed by anyone on the app. Value will not be NULL because it is needed for users to identify.
owner_fname: VARCHAR(150)	Attribute holds the first name of the business owner(s). Provides information for other users to say who owns the business.
owner_lname: VARCHAR(150)	Attribute holds the last name of the business owner(s). Provides information for other users to say who owns the business.
business_password: VARCHAR(50)	Attribute will hold the password the business sets for their account.

business_subscription: BIT	This attribute is a value that returns true or false depending on whether the business wants to purchase a premium account.
business_profile: VARCHAR(250)	Attribute will hold an image link that will have a photo that will be used as a profile picture.
email: VARCHAR(150)	A business email is used as a way of sending any notifications. Because the business needs a method of receiving these notifications the value will not be NULL. The attribute also needs to be UNIQUE so multiple accounts will not be created under the same email.
phone_number: CHAR(11)	Attributes will hold the phone numbers of the business. A Char value of 15 is given to include international numbers and area codes as well. The phone number will be accessible to the business only after an appointment has been made.
rate_per_hour: CHAR(10)	Value holds the rate per hour, set by the business, that they are charging consumers for their services. Value will not be NULL because consumers need to know the view before proceeding.
location: VARCHAR(50)	Attribute is used to find consumers in the same areas as the business.
keywords: VARCHAR(50)	Keywords used to attach to the business. These could be subjects or areas the business relates to. Consumers will use certain keywords to find business concerning a specific term.
education: VARCHAR(150)	This attribute holds the education level of the business. This could be the certifications they have, qualifications they have or any professional documents they have received.
pictures: VARCHAR(500)	This attribute enables businesses to add photos to their account that are associated with their business. This will include examples of work done or any information where pictures are better than words.
description VARCHAR(500):	This attribute holds the information of the business. Here, the business can enter anything about their business that consumers can see.

Chat

This table will contain information about the chatting system between the consumer and business. The messaging system will post a message and then receive a message. This is the method to use for communication between the two clients. The clients will be able to see very basic information about each other in the chat window.

Attribute	Description
chatID : INT **[PK]**	This attribute is the unique identifier for the chat.
consumerID: INT	This attribute is the unique identifier for the consumer.
businessID: INT	This attribute is the unique identifier for the business.
time_stamp : DATE	This value will hold the time the message was sent to the other user.
chat_message : VARCHAR(500)	This value holds the contents of the message the user wants to send.

Education History

This table will store information based on the businesses education history. A business will be able to enter their highest completed education as well as their education level which is visible to anyone who visits their profile. Aside from the businessID, all values can be edited and changed.

Attribute	Description
businessID : INT **[PK]**	This attribute is the unique identifier for the business.
education_level : VARCHAR(255)	This attribute holds the education level of the business.
highest_education_completed : VARCHAR(50)	This attribute holds what the highest level of education completed by the business

Job Type

This table will store information regarding the job type the business is. A business will be able to create a title for the jobs they are offering and it will be categorized into a particular category. Consumers will then be able to search for a service (an AD) using these values.

Attribute	Description
businessID : INT **[PK]**	This attribute is the unique identifier for the business.
job_title : VARCHAR(50)	This value will hold the title of the job shown to the consumers.
job_category : VARCHAR(50)	This value will hold the category the job falls into.

Ad

This table will store information pertaining to ads. When businesses want to create an AD they will fill out the following information and will then be visible for consumers to see. Ads is a way for businesses to advertise the different jobs they are offering.

Attribute	Description
adID : INT **[PK]**	This attribute is the unique identifier for the ad.
businessID : INT	This attribute is the unique identifier for the business.
ad_title : VARCHAR(50)	This value will be the title the business chooses for the ad.
ad_body : VARCHAR(500)	This attribute will contain information for the ad (service) the business is offering.
location : VARCHAR(50)	This value will show the location of where the business is offering the service.
rate_per_hour : CHAR(10)	This attribute holds the rate the business charges per hour for the service.

Project Management Breakdown

Work Breakdown Structure

Week 1 to 3 are empty as these weeks were introduction to the class, lectures and guest speakers. No official group work or planning started until Week 4.

	Austin	Wasi	Calvin	Po-Chia	Estimated Total	Actual Total
Week 4	23:22:42	22:00:00	12:15:04	06:36:00	64	64:13:46
Week 5	06:45:53	12:00:00	14:46:02	06:10:14	64	39:42:09
Week 6	12:37:00	13:51:44	10:18:41	08:33:04	64	45:20:29
Week 7	11:56:00	08:36:15	07:30:00	08:35:15	64	36:37:30
Week 8	06:24:15	18:06:24	07:30:37	10:13:58	64	42:15:14
Week 9	17:33:08	16:47:24	13:30:13	19:27:00	64	67:17:45
Week 10	20:43:56	08:19:21	12:35:00	12:38:57	64	54:17:14
Week 11	20:37:43	17:38:00	16:09:00	21:24:28	64	75:49:11
Week 12	20:24:15	32:51:00	23:24:00	21:27:00	64	98:06:15
Week 13	15:15:56	22:45:00	13:20:00	09:53:00	64	61:13:56
Week 14	10:27:26	10:13:00	10:17:00	11:56:00	64	
Total time:	166:08:14	192:51:08	145:51:37	145:05:56	640	649:56:55

The graph shows the times for each member starting from week 4 and ending at week 13. Each member is to have an estimated amount of 16 hours per week of active development, hence the average of 64 hours per week.

Task Breakdown

The chart is a breakdown of all the tickets created throughout the development of the project. The tasks are all labeled with a level of difficulty, the status of completion and which developer completed the ticket.

Legend			
1 - Size of Task: Small			
2 - Size of Task: Medium			
3 - Size of Task: Large			
X : Not Complete			
✓ : Completed			
O : Partially Completed			
AW - Austin Wong			
WR - Wasi Raza			
CQ - Calvin Qu			
PW - Po-Chia Wei			
Tasks	**Task Size**	**Status**	**Developer**
Writing Charter, Scope, and Requirements Document	2	✓	AW, WR, CQ, PW
Wrting Design Document	2	✓	AW, WR, CQ, PW
Prepartion of Design and Testing Presentation	1	✓	AW, WR, CQ, PW
Client Meeting	1	✓	WR
Creation of Burn-up Chart	1	✓	PW
Creation of Database	3	✓	AW
Creation of Required Tables and Dummy Data	2	✓	AW
Creation of Controller and Model and Route for Each Table	3	✓	AW
Implementing pagination	2	O	AW
Prepartion of MVP Presentation	1	✓	AW, WR, CQ, PW
Creation of Schema	2	✓	AW
Creation of Login	2	✓	CQ, PW
Logging out of account	2	X	
Account recovery	2	X	
Creation of Middleware and Token	2	✓	CQ, PW
Creation of Stripe	3	✓	WR
Setting Heroku Environment	1	✓	WR
Setting Docker Environment	1	✓	WR
Setting Drone Environment	3	✓	WR
Code Review	3	✓	WR
Setting CI/CD Integration	2	✓	WR
Setting Toggl for Time and Task Tracking	1	✓	AW
Route Guard for Different Accounts	2	O	AW
Integration Test and Debug	2	✓	CQ, PW
Unit Test and Debug	2	✓	CQ, PW
Endpoint Test and Debug	2	✓	CQ, PW
Stripe Test and Debug	2	✓	CQ, PW
Writing Endpoints Documentation	2	✓	AW
Prepartion of Final Presentation	1	✓	AW, WR, CQ, PW
Writing Final Document	2	✓	AW, WR

Figure 4: Describes all the tasks, the size of the task, completion status and responsibilities. The legend is provided to show what each value represents.

Final Burnup Chart

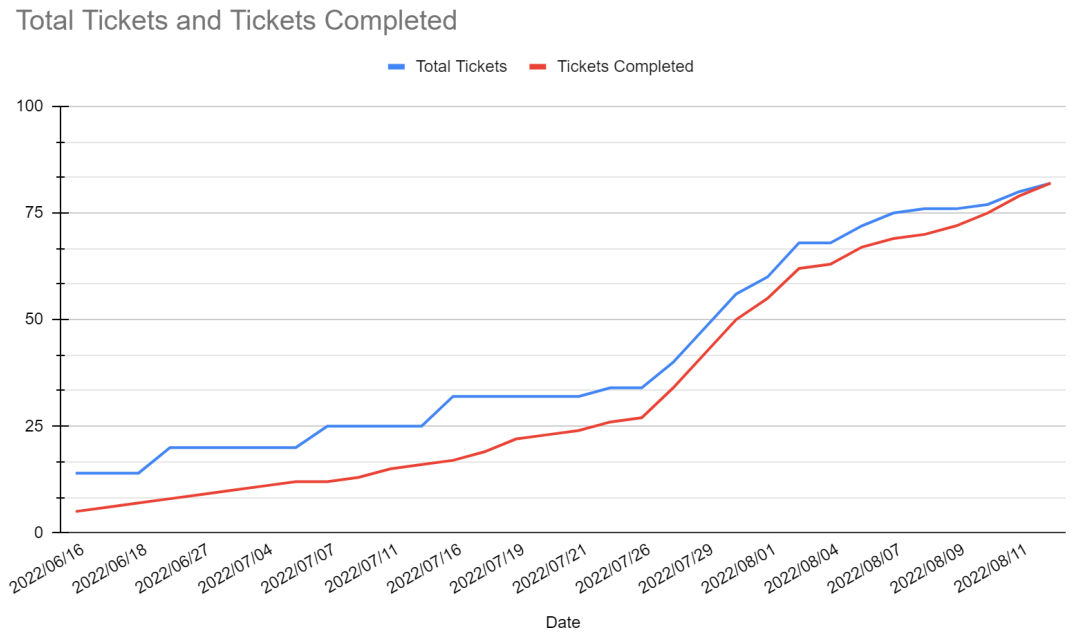

Figure 8: Is a burnup chart showing the progress of ticket completion. The space between the two lines in the middle of the chart is when the team had tickets that were too large, and we were not able to deliver. When it gets more parallel is when we create smaller tickets and are able to finish quicker.

Final Burndown Chart

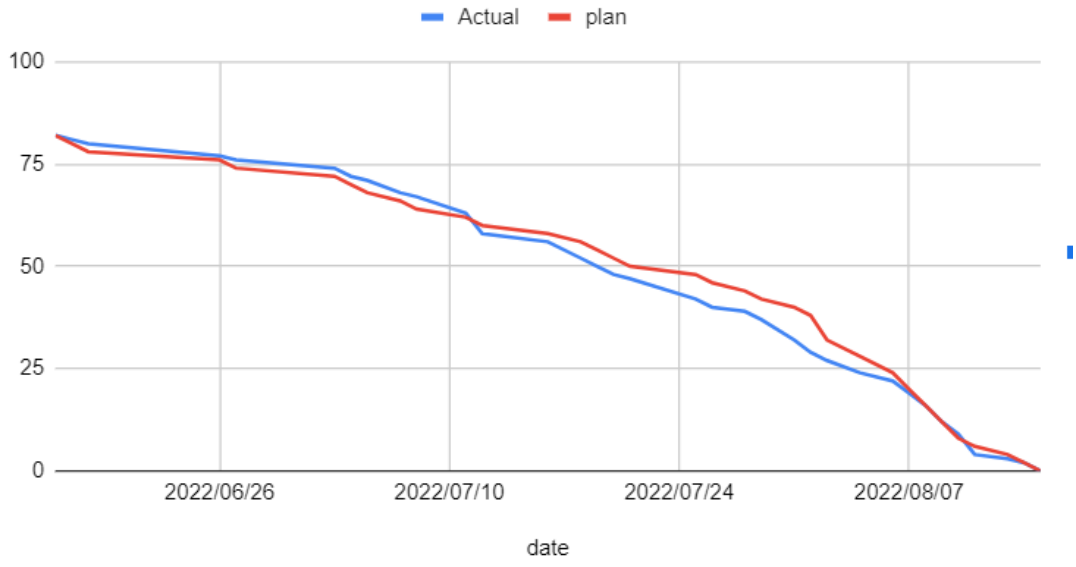

Figure 9: The burndown chart is an inverse of the burnup chart and reflects the same information. It shows the struggles during the beginning of production when the team had larger tickets, and the line separates when smaller tickets were created and were finished quicker.

Delivered Requirements

Searching for ads

The consumer will be able to search for ads. The user will be able to search for an ad using a name or a keyboard, they would then be provided with businesses that match results from the input. The diagrams below show the process on how the system handles the search.

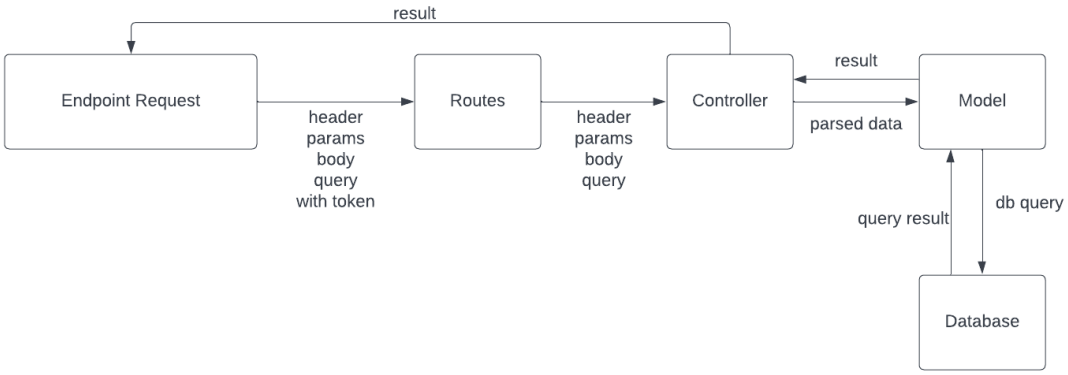

Figure 5: Shows a DFD represents how data flows when searching for an ad

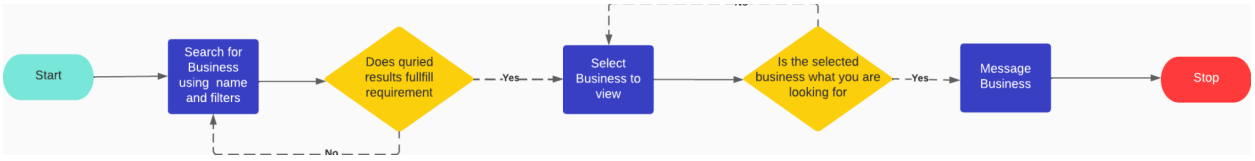

Figure 6: Is a flow chart for a consumer to represent the process of looking for a service. They have the option to search by name or keyword. They can then select a business and send them a message.

Database

A database was created and will be used to store the user data on the app. Because Stripe handles everything regarding payment, payment and payment information will not be stored in the database. The database will also allow manipulation of user data.

Login to an Account

When a user tries to login to their account, the front-end system will send a request to our login endpoint which will connect to the database to ensure the login information matches. If everything passes, the system will send the front-end a token used to login.

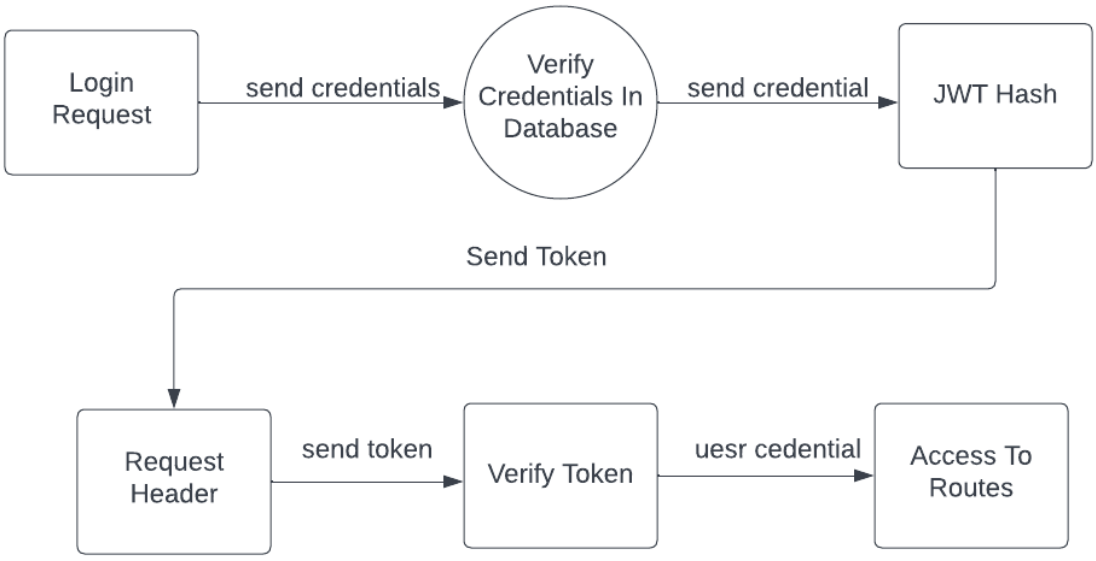

Figure 7: Shows a DFD that represents how the system will handle login and tokens.

Stripe

The system now has a wrapper that interacts with Stripe to allow payments on the app. Due to issues regarding payment, the only payment is done over the app purchasing subscriptions. Stripe will handle and be responsible for all the security regarding the payment and also provide a specified stripe user id for the business and consumer users.

Tombstoning Accounts

The team has decided that if a user wants to delete an account, it will not be permanently deleted and will only be tombstoned. This will prevent further creations of accounts under the same email.

Over the App Messaging

Messaging over the app will be done in a post and receive method. A user will post a message and when the receiving user opens the app they will see the message. The messaging on the app will not be live.

Features that were not fully completed:

Pagination:

Pagination was one feature that was not able to be fully implemented. This feature is present but has not been implemented and connected to the database.

Route Guard:
Route guard was another feature that was fully coded and tested using dummy data; however it has not been connected and implemented using the database.

Features that were not completed:

Logout Feature:
Due to time limitations there was no feature to logout of the account. So, as of now the user will not be able to logout of the account.

Account Recovery:
Due to further time limitations, account recovery was not able to be implemented. Therefore, if a user forgets their login, they would not be able to recover the account.

Chat Feature
There already exists a database and route for chat, however there has not been a feature added to implement everything.

Issues Found While Developing

As there was no front-end integration there was not too much testing that could be done to find any bugs. An issue found is that the route guard data is present, but has not yet been implemented. Therefore currently a global login can be used to access all data.

Tech Stack

MySQL:
Our team decided to use mySQL when developing the database because mySQL is secure and is also reliable when handling data. Also, because our clients are still a smaller company, mySQL is very cheap to maintain.

Node js
We decided to use Node js because of its compatibility. In the scenario the clients ever want to change their framework, Node js is very compatible with other frameworks.

Express Node
Express node is used to create tokens when logging into an account. Upon successful login the token will be sent to the front-end and will then be connected back to the APIs.

Stripe Purchasing API

We chose to use Stripe to handle payments over other APIs due to its low cost. As other APIs take a larger cut of the payment, stripe is the best option to handle payments for a smaller company.

Mocha Tests

Mocha was used to tests the endpoints. This was done by sending data through the body parameters, and queries to the respected route, to simulate a real world request.

Test Plan

Scope

The purpose of this test plan is to ensure we meet all of our functional, non-functional, technical, and user requirements without the introduction of bugs or issues. Our overall test plan will consist of unit testing, code review, performance testing, and bug fixing. Testing will be a continuous process throughout its duration.

Process

Testing will be integrated into the development process of the service. We will begin by developing the actual service. Once completed, we will create a minimum of three unit tests to check service behaviours.

1. To check the expected output with expected input
2. To check the behaviour with unexpected input
3. To check the behaviour with nonexistent input

Other unit tests will be created on a case-by-case basis. Once the system passes all unit tests it will be handed over to the integration manager for code review. The integration manager will check the code to see if the code can be made more efficient, bug-free and will not break or collide with other systems. Once completed, the system will receive test cases to ensure nothing will break the code. If all test cases pass then the system is deemed tested and operational.

Requirements	Type of Test (U: Unit Tests, E: Endpoint Tests, I: Integration Tests)	Pass or Fail(P:Pass, F: Fail)	Contributor (WR:Wasi RAZA, AW:Austin Wong, CQ: Calvin Qu, PW: Po-Chia Wei)
Functional Requirements			
The system will search a business using a collection of keywords and name	U, E, I	P	CQ, PW
The system will be able to delete or tombstone user accounts	U, E, I	P	CQ, PW
The system will be able to recognise an update to account information and update accordingly	U, E, I	P	CQ, PW
The system will be able to recognise the different types of accounts upon login			
The system will handle all transactional data and sales data via the Stripe suite and API	U, E, I	P	WR
Non-Functional Requirements			
Ads will be displayed to the user with pagination			
A route guard is implemented to make sure the correct user is able to use the correct features			
Login credentials are email and password	E	P	CQ, PW
Technical Requirements			
The system will use a Node.js framework to retrieve data from a MySQL database	I	P	CQ, PW
JWT tokens are used for authentication and session handling	E	P	CQ, PW
User Requirements			
Users will be able to view other user's profile	U, E, I	P	CQ, PW
Users are able to log in, log out and recover accounts	E	P	CQ, PW
Consumer users are able to view ads made by business users	U, E, I	P	CQ, PW
Consumers will be able to purchase a subscription via the Stripe API that will change their accounts to Business.	U, E, I	P	WR
Business users are able to create, delete update ads	U, E	P	CQ, PW
A business will be able to have multiple Ads up to 3			
Consumer users and business users are able to communicate over a chat feature			
Admin accounts will add delete update user credentials and information	U, E	P	CQ, PW
Admin accounts are able to access the Stripe information dashboard and view, refund and update transactional data			
There will only exist one HelpYa account at a time			
Helpya account will be able to drop database	U,E	P	CQ, PW

Figure 10: Shows the progress and which tests were done on the software. Included is also whether or not the tests passed and who contributed to the tests.

Tests not Completed

- Recognition of different account types
- Testing database with the ads feature
- Route guard testing with database
- Maximum ads allowed to be made at once
- Only one HelpYa account can be created

Testing environment

The testing environment will be all Windows 10 machines running the latest version of VS Code. A local server will be used to test code on the machine it is being developed before it is handed to the integration manager for further testing. All unit tests will be written in mocha.

Reflection

Key Lessons

Key lessons the team has learned throughout the development of the project was the importance of organization and planning when developing a system. Without proper planning at the

beginning the team was lost, unorganized and did not know who was doing what task. As our project required a lot of planning, it held us back and delayed the production of the system. Also learning the importance of breaking down tasks, doing so made development easier and made the team realize other features had to be implemented. Branching was also an important lesson the team learnt, this caused issues for us a first because when a feature was finished being tested it was not being merged back into master, doing so created merge conflicts.

Next time

If the opportunity were to arise where we can do this project again, there would be a heavy focus on project management at the very beginning. Doing so we can create many different tickets that are broken down so the team can easily delegate tasks. Also with certain tasks, time management was also an issue and caused delays in finishing other tasks. If there was another opportunity, members would finish other smaller tickets first to help progress the development of the project.

Conclusion

As the project comes to a close, based on the client's requirements, the project was a success. All the key requirements were successfully delivered and the clients are satisfied with the results of the project. Although there is still some work needed to implement our software with the clients front-end, with time it can be implemented and be up and running.

Product Delivery

Workflow and Deployment

An agile and scrumban approach was used for the development of this project.

- Weekly scrum meetings were done to discuss tasks that were going to be done
- Review code weekly and ensure the development was successful
- Tickets to show the tasks needing to be done and who was doing the tasks

Before deploying a feature, each portion was tested with unit tests to ensure there were no bugs in the code. The integration lead then reviewed the code to ensure efficiency and then it was merged into our master branch. When developing other features, a new branch from master was made and the process repeated.

Installation

As our project is back-end and involved development with APIs there will not be too much that is needed to be installed. To get the code you would clone to repository using this link:

https://github.com/UBCO-COSC-499-Summer-2022/api-development-and-testing-project1-helpya-services-inc . To install all the necessary libraries you would run the command `npm install` in the same terminal you cloned the repository to.