**Title Page: Ne1 Freelance Semester Project**

Raushawn Mitchell #94804, Aaron Hazzard #95455

University of Trinidad and Tobago

Information and Communication Technology

Diploma in Software Engineering

SFEN2005

Ms. Kerryann Xavier

Friday 26th May 2023

**Table of Contents**

Preface…………………………………………….…pg 4

Introduction……………………………………….…pg 5

Business Overview and Objectives………….pg 5

Business Overview……………………pg 5

Project Objectives…………………….pg 5

Project Overview………………………….….pg 5

Scope of Project………………………pg 5

Research Methods…………………….pg 5

Requirement Analysis…………………………………pgs 6- 16

Functional Requirements of the Application….pg 6

Non-Functional Requirements…………….…….pg 6

Use Cases and Use Case Descriptors……………..pgs 6-13

Use Cases………………………………….pgs 6-8

Use Case Descriptors……………………...pgs 9-13

Class Diagram………………………………….….pg 14

Sequence Diagrams……………………………...….pgs 15-16

System Design………………………………………...…….pgs 17-19

Layered Architecture…………………………………pg 17

Package Diagram…………………………….pg 17

Component Diagram………………….pg 17

Hardware Architecture

Textual Description………………………………pgs 17-18

Deployment Diagram……………………………pg 18

Communication Diagram…………………………pg 19

Hardware Platform……………………………………………….pg 19

Software Platform…………………………………………………pgs 19-20

Persistence Model……………………………………………….pgs 21-22

ERD Diagram………pg 21

File Structures………pgs 21-22

Data Dictionary……………………………………………….pgs 22-25

System Administration……………………………………….pg 26

Security………………………………………………pg 26

Backup and Restore………………………………….pg 26

Testing……………………………………………………….pgs 26-28

Implementation Plan…………………………………………pgs 28-29

Lessons Learned……………………………………………..pg 30

Technical………………………………………….….pg 30

Managerial…………………………………………...pg 30

Recommendations……………………………………pg 30

User Procedures……………………………………………pgs 30-31

Appendix……………………………………………………pg 31

**Preface**

This assignment serves as an opportunity to delve into the fascinating world of website design. It has been crafted with the aim of exploring key concepts, developing critical thinking skills, and deepening our understanding of website design in the context of our university curriculum.

Throughout this assignment, we will embark on a journey of discovery, analysing various aspects of website design and their significance in software engineering. By engaging in thoughtful research, reflection, and analysis, we aim to broaden our knowledge and gain valuable insights into the subject matter.

I would like to express my sincere appreciation to our teacher for their guidance, encouragement, and expertise in shaping this assignment. Their dedication to our learning journey has been instrumental in creating an enriching academic experience.

Additionally, I extend my gratitude to my fellow classmates for their collaboration, stimulating discussions, and shared enthusiasm in tackling the complexities of website creation. Our collective effort and diverse perspectives have contributed to the depth and richness of this assignment.

I hope this assignment serves as a platform for personal growth, fostering curiosity, and nurturing a passion for lifelong learning. May it inspire us to explore beyond the confines of the classroom and embrace the transformative power of education.

Best regards,

Raushawn

**Introduction**

**Business Overview and Objectives**

***Business Overview***

NE1 Consultancy has a mobile application called NE1 that provides various services for freelancers across Trinidad and Tobago. However, their application is currently down due to failed services and features. Therefore, we decided to take the opportunity to create a Web Application cloned version of their Mobile Application so that they can continue their business.

***Project Objectives***

Our objective with this project is to facilitate our clients and create job opportunities through a medium which we will attempt to create and will benefit their customers and our clients.

**Project Overview**

***Scope of project***

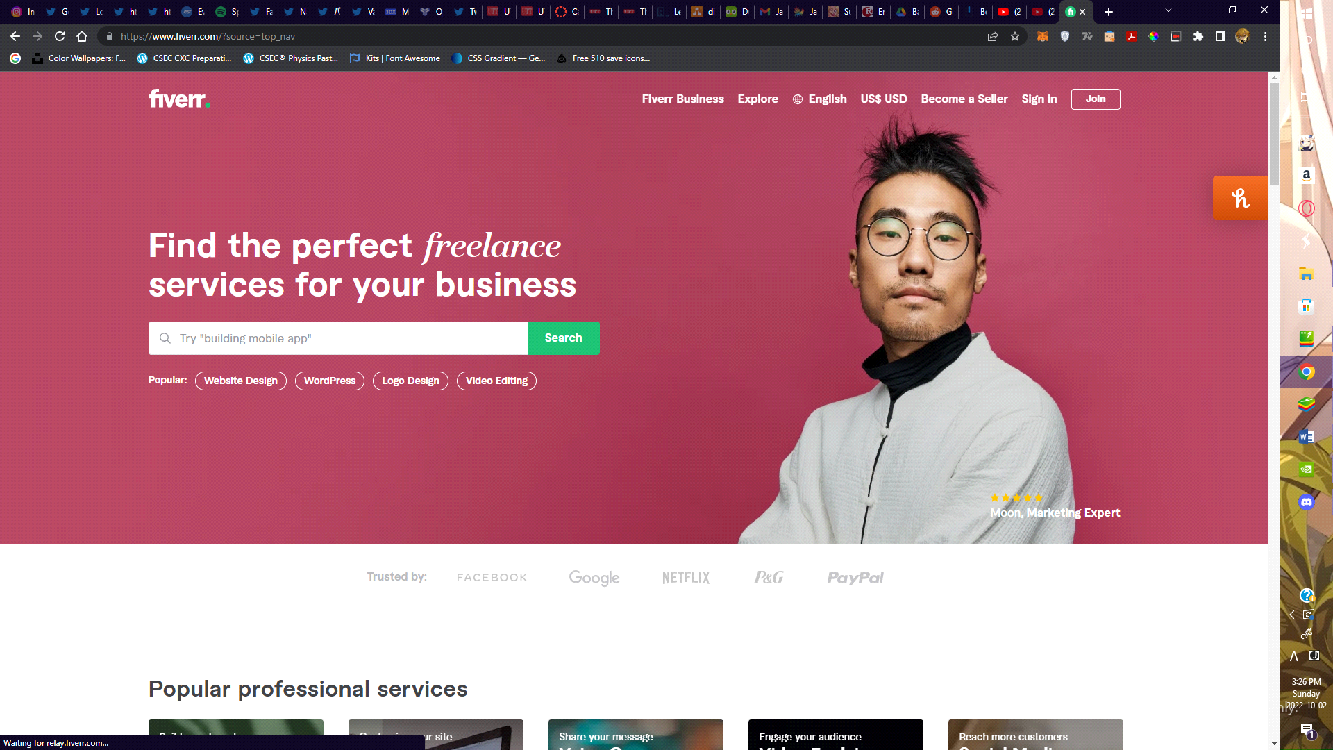
We are only obligated to follow the instructions that this document states, which will be making and designing a website for freelancers can use freely.

***Research methods***

We plan to provide the best and most unique services and options for our clients so that we can dominate the competition and be successful in this field. This will be done through an analysis of free-lancing websites and basing our research on how we could take their product and make it better and more desirable for our clients.

This is a sample website which we will take inspiration from and create a bigger and better product.

<https://www.fiverr.com/>



**Requirement Analysis**

**Functional Requirements of the Application**

• Log in/Sign up System for user profiles

• Freelancers’ details

• Keep track of online transactions

• Unique Sections for each genre of freelancers

• Invoice and distribution of money to clients

**Non-Functional Requirements**

• Minimum of 100TB of data to store

• Any device capable of browsing the web

• 24/7 Customer service support

• Maintenance and Monthly updates

• DBMS (Database Management System)

• The information must be fetched in less than a minute

• Easy to use GUI (Graphical User Interface)

• Daily system backups

**Use Cases and Use Case Descriptors**

***Use Cases***

A picture containing diagram, text, line, circle

Description automatically generated

A picture containing text, diagram, sketch, drawing

Description automatically generated

A picture containing text, sketch, diagram, drawing

Description automatically generated

A diagram of a user profile

Description automatically generated with medium confidence

***Use Case Descriptors***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case Name:** Category Choices | | **ID: NULL** | **Importance Level:** High | |
| **Primary Actor/s: Freelancer, Database** | | | **Use Case Type: NULL** | |
| **Stakeholders and Interests**  User – Freelancer must select a job category to offer their services  Database – Checks for availability of category storage | | | | |
| **Brief Description:** This Uses Cases shows the process that a freelancer goes through if they want to select a job category | | | | |
| **Trigger: NULL** | | | | |
| **Relationships:** | | | | |
|  | 1. Include (Display info based on category 2. Extend (Grand/Deny access) | | |  |
| **Normal Flow of Events:**   1. Navigate to profile. 2. Select become a NE1 Freelancer 3. Setup job profile: 4. Click continue and become a NE1 Freelancer | | | | |
| **Sub Flows: NULL** | | | | |
| **Alternate / Exceptional Flows**   * **Sub Flows:** if a user selects a category that is filled an error prompt appears. | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Use Case Name:** User Profile | | **ID:** | **Importance Level:** High | | |
| **Primary Actor/s: User (Freelancer, Actor)** | | | | **Use Case Type: NULL** | |
| **Stakeholders and Interests**  User – Changes their account information.  Database – Validates and changes based off availability | | | | | |
| **Brief Description:** This Uses Cases shows how a user can interact with the database/website to change their account information | | | | | |
| **Trigger:** NULL | | | | | |
| **Relationships:** | | | | | |
|  | 1. Include (Verification) 2. Extend (name taken, weak password, email taken, unsupported file type, number taken) | | | |  |
| **Normal Flow of Events:**   1. Select user profile. 2. Click edit profile information. 3. Allow user input for requested fields to change (name, password, email, profile picture, phone number) | | | | | |
| **Sub Flows: if either information is taken database will send an error stating that it has been.** | | | | | |
| **Alternate / Exceptional Flows**  1.2 If username is taken prompt username taken, try again.  1.2 If password is too weak prompt weak password, try again.  1.1 If email is taken prompt email taken, try again. | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Use Case Name:** Invoice | | | **ID:** 2 | **Importance Level:** High | |
| **Primary Actor/s:** Client Secondary Actor: Freelancer, Database | | | | | **Use Case Type:** |
| **Stakeholders and Interests**  Client – wants invoice as proof of payment.  Freelancer-wants invoice  Database-provides invoice | | | | | |
| **Brief Description:** This Uses Cases shows the process in the invoice system | | | | | |
| **Trigger: requests of invoice** | | | | | |
| **Relationships:** | | | | | |
|  | Include |  | | | |
| **Normal Flow of Events:**   1. Request invoice by client 2. Database processes information 3. Invoice verification by freelancer | | | | | |
| **Sub Flows:** | | | | | |
| **Alternate / Exceptional Flows:**  2a.  3a. | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Use Case Name:** Refund | | | **ID:** 5 | **Importance Level:** High | |
| **Primary Actor/s:** Client, Freelancer Secondary Actor: Database | | | | | **Use Case Type:** |
| **Stakeholders and Interests**  Administrator – wants to know what class is in session.  Student – wants to know where what day and time a class is scheduled | | | | | |
| **Brief Description:** This Uses Cases shows who will need to see class calendars | | | | | |
| **Trigger: User request** | | | | | |
| **Relationships:** | | | | | |
|  | Extend |  | | | |
| **Normal Flow of Events:**   1. Verifies the user request for refund. 2. Process information 3. Verify based on guidelines. 4. Communicates with freelancer. 5. Freelancer choose to accept or deny | | | | | |
| **Sub Flows:** | | | | | |
| **Alternate / Exceptional Flows:**  2a. If the system deems the request invalid, deny refund. | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Use Case Name:** Input Credentials | | **ID:** 5 | | **Importance Level:** High | |
| **Primary Actor/s:** User, Database | | | | | **Use Case Type:** |
| **Stakeholders and Interests**  User – must input information.  Database – needs to store and access data | | | | | |
| **Brief Description:** This Uses Cases shows how user enters information on system | | | | | |
| **Trigger:** | | | | | |
| **Relationships:** | | | | | |
|  | Include (Verify Credentials) | |  | | |
| **Normal Flow of Events:**   1. Input username 2. Input Password | | | | | |
| **Sub Flows:** | | | | | |
| **Alternate / Exceptional Flows** | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Use Case Name:** Verify Credentials | | **ID:** 5 | **Importance Level:** High | | |
| **Primary Actor/s:** Database | | | | **Use Case Type:** | |
| **Stakeholders and Interests**  User – must input information.  Database – must check that information is correct | | | | | |
| **Brief Description:** This Uses Cases shows how credentials are checked | | | | | |
| **Trigger:** Input Credentials | | | | | |
| **Relationships:** | | | | | |
|  | Deny Access Extends Verify Credentials  Grant Access Extends Verify Credentials | | | |  |
| **Normal Flow of Events:**   1. Checks for username on database 2. Checks if password matches username. | | | | | |
| **Sub Flows:** | | | | | |
| **Alternate / Exceptional Flows**  1.2 If username matches password, grant access.  1.2 If username and password does not match, deny access.  1.1 If username does not exist deny access | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Use Case Name:** Input Information | | **ID:** 5 | | **Importance Level:** High | |
| **Primary Actor/s:** User, Database | | | | | **Use Case Type:** |
| **Stakeholders and Interests**  User – must input information.  Database – needs to store and access data | | | | | |
| **Brief Description:** This Uses Cases shows how account information is entered | | | | | |
| **Trigger:** | | | | | |
| **Relationships:** | | | | | |
|  | Association  Include (Select type of account, enter username and password) | |  | | |
| **Normal Flow of Events:**   1. Create username. 2. Create Password 3. Enter password again. 4. Select type of account | | | | | |
| **Sub Flows:** | | | | | |
| **Alternate / Exceptional Flows** | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Use Case Name:** Verify Password | | **ID:** 5 | **Importance Level:** High | | |
| **Primary Actor/s:** Database | | | | **Use Case Type:** | |
| **Stakeholders and Interests**  User – must input information.  Database – needs to store and access data | | | | | |
| **Brief Description:** This Uses Case shows how a password is verified | | | | | |
| **Trigger:** Input Information | | | | | |
| **Relationships:** | | | | | |
|  | Create Account Extends Verify Password  Display Error Extends Verify Password | | | |  |
| **Normal Flow of Events:**   1. Checks to see if password matches | | | | | |
| **Sub Flows:** | | | | | |
| **Alternate / Exceptional Flows**   * 1. If passwords match, create account.   1.1 If passwords do not match, display error | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Use Case Name:** Store User Information | | **ID:** 5 | | **Importance Level:** High | |
| **Primary Actor/s:** User, Database | | | | | **Use Case Type:** |
| **Stakeholders and Interests**  User – must input information.  Database – needs to store data | | | | | |
| **Brief Description:** This Uses Cases shows how database stores information entered by user | | | | | |
| **Trigger:** | | | | | |
| **Relationships:** | | | | | |
|  |  | |  | | |
| **Normal Flow of Events:**   1. User inputs username, password, and other information 2. Database adds information to the system | | | | | |
| **Sub Flows:** | | | | | |
| **Alternate / Exceptional Flows** | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Use Case Name:** Close Account | | | **ID:** 5 | **Importance Level:** Low | |
| **Primary Actor/s:** User, Database | | | | | **Use Case Type:** |
| **Stakeholders and Interests**  User – must request closure of account.  Database – have access to account and account information | | | | | |
| **Brief Description:** This Uses Cases shows how an account may be closed | | | | | |
| **Trigger:** | | | | | |
| **Relationships:** | | | | | |
|  |  |  | | | |
| **Normal Flow of Events:**   1. User requests closure of account 2. Information is removed from the database. 3. Access is removed from account. 4. Account is deleted | | | | | |
| **Sub Flows:** | | | | | |
| **Alternate / Exceptional Flows** | | | | | |

**Class Diagram**

**A picture containing screenshot, line, design

Description automatically generated**

**Sequence Diagrams**

***Login***

Diagram

Description automatically generated

***Signup***

Diagram

Description automatically generated with low confidence

***Request Refund***

Diagram

Description automatically generated

**System Design**

**Layered Architecture**

***Package Diagram***

A picture containing text, diagram, plan, technical drawing

Description automatically generated

***Component Diagram***

***A picture containing text, line, screenshot, diagram

Description automatically generated***

**Hardware Architecture**

***Textual Description***

The front-end of the site is developed using React, a popular JavaScript library for building user interfaces. React allows developers to create interactive and dynamic UI components, handle user interactions, and efficiently update the UI based on data changes. React uses a virtual DOM (Document Object Model) to efficiently render and update components, resulting in a responsive and seamless user experience.

Node.js is used on the server-side to handle requests and responses. It provides a runtime environment for executing JavaScript outside the browser, allowing developers to build scalable and high-performance server-side applications. Node.js uses an event-driven, non-blocking I/O model, making it well-suited for handling concurrent requests and optimizing server performance.

For data storage, the site uses MongoDB, a NoSQL database that provides flexible and scalable document storage. MongoDB stores data in a JSON-like format called BSON (Binary JSON), which allows for the efficient storage and retrieval of structured and unstructured data. It supports high availability and automatic sharding, enabling horizontal scaling as the site's data grows.

The front-end, developed with React, interacts with the server-side powered by Node.js through RESTful APIs (Application Programming Interfaces). These APIs define the endpoints and data formats for communication between the client and the server. React components make HTTP requests to the Node.js server to fetch data from MongoDB or send data for storage and processing. The server processes these requests, communicates with the MongoDB database, and returns the appropriate response to the client.

Overall, this combination of React, MongoDB, and Node.js enables the development of a fast, scalable, and dynamic web application. React provides an efficient and interactive user interface, Node.js handles server-side logic and communication, and MongoDB ensures flexible and scalable data storage. Together, they form a powerful stack for building modern web applications.

***Deployment Diagram***

A picture containing text, screenshot, rectangle, diagram

Description automatically generated

***Communication Diagram***

A picture containing text, screenshot, font, design

Description automatically generated

**Hardware Platform**

Storage Systems.

**Software Platform**

**Front-End Component:**

React: The front-end component of your site will be built using React, a JavaScript library for building user interfaces. React allows you to create reusable UI components, manage state, and handle user interactions.

**Back-End Component:**

Node.js: The back-end component of your site will be built using Node.js, a JavaScript runtime environment. Node.js enables server-side scripting and allows you to handle server-side logic, routing, and database interactions.

Express.js: Express.js is a popular web application framework for Node.js. It provides a set of tools and features to build robust and scalable web applications. Express.js simplifies the process of handling HTTP requests, routing, and middleware integration.

**Database Component:**

MongoDB: MongoDB is a NoSQL document database that stores data in flexible, JSON-like documents. It is widely used with Node.js for its flexibility and scalability. MongoDB allows you to store and retrieve data efficiently and provides powerful querying capabilities.

**Additional Tools and Libraries:**

Axios: Axios is a popular HTTP client library that allows you to make HTTP requests from your Node.js server to external APIs or services.

Mongoose: Mongoose is an Object Data Modelling (ODM) library for MongoDB and Node.js. It provides a higher-level abstraction for interacting with MongoDB, making it easier to define schemas, , perform CRUD operations, and establish relationships between data models.

**Package Managers**

NPM (Node Package Manager) or Yarn are commonly used package managers in the Node.js ecosystem. They allow you to install, manage, and update the dependencies required by your application. You can use these package managers to install libraries, frameworks, and other modules needed for your React, Node.js, and MongoDB project.

**Interaction Model**

**Persistence Model**

**ERD Diagram**

***A picture containing text, diagram, plan, technical drawing

Description automatically generated***

**File Structures**

‘apiServer/APIs.js’- file to run nodemon on server.

‘apiServer/controllers/’- A directory containing controllers that handle business logic for various operations.

‘apiServer/models/’ - A directory containing data models or schemas for MongoDB collections.

‘apiServer/routes/’- A directory containing route handlers for different API endpoints.

.env – File containing sensitive information like API keys and database connection string.

node\_modules- directory containing all necessary packages to run server.

public/- A directory for static assets, such as images, and client-side JavaScript files.

src/ - Directory for the react app’s source code.

components/ - A directory containing individual page components.

static/ - A directory containing css and images that are not added by a user.

App.js - The root component of the React application.

index.js – The entry point for the react application.

yarn.lock - File containing necessary packages to run application.

**Data Dictionary**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | About Us | | | | | |
| **Description** | Stores information on about us | | | | | |
| **Name** | **Type** | **Field Length** | **Mandatory?** | **Unique?** | **Description** | **Example** |
| information | String |  | yes | no | Information for about us page | This is information. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | Contact | | | | | |
| **Description** | Stores contact information | | | | | |
| **Name** | **Type** | **Field Length** | **Mandatory?** | **Unique?** | **Description** | **Example** |
| name | String |  | yes | no | Name of person sending message. | Raushawn |
| email | String |  | yes | no | Email to contact | mm@mail.com |
| message | String |  | yes | no | Message to be sent | This is a message. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | Credit Card | | | | | |
| **Description** | Stores information on credit card | | | | | |
| **Name** | **Type** | **Field Length** | **Mandatory?** | **Unique?** | **Description** | **Example** |
| user\_id | ObjectId |  | yes | yes | Reference to id in user model | 1 |
| cardNumber | String |  | yes | yes | Credit card number | 0000 0000 0000 0000 |
| expiryDate | String |  | yes | no | Expiry date of credit card |  |
| securityCode | String |  | yes | no | Security code for card |  |
| firstName | String |  | yes | no | First name of user | Shawn |
| lastName | String |  | yes | no | Last name of user | Parris |
| **Model** | Invoice | | | | | |
| **Description** | Stores information on invoice | | | | | |
| **Name** | **Type** | **Field Length** | **Mandatory?** | **Unique?** | **Description** | **Example** |
| client\_id | ObjectId |  | yes | yes | Reference to id in user model | 1 |
| freelancer\_id | ObjectId |  | yes | yes | Reference to id in user model | 1 |
| transaction\_id | String |  | yes | yes | Expiry date of credit card |  |
| date | Date |  | yes | no | Security code for card |  |
| payment\_details | String |  | yes | no | Information on payment | Paid |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | JobCategories | | | | | |
| **Description** | Stores information on job categories | | | | | |
| **Name** | **Type** | **Field Length** | **Mandatory?** | **Unique?** | **Description** | **Example** |
| name | String |  | yes | Yes | Name of job category | Cleaning |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | Jobs | | | | | |
| **Description** | Stores information on jobs | | | | | |
| **Name** | **Type** | **Field Length** | **Mandatory?** | **Unique?** | **Description** | **Example** |
| user\_id | ObjectId |  | yes | yes | Reference to id in user model | 1 |
| title | String |  | yes | yes | Job title | 0000 0000 0000 0000 |
| description | String |  | yes | no | Details on what job entails. | We clean really good. |
| thumbnail | String |  | yes | no | Thumbnail for job | Cats.jpeg |
| price | Number |  | yes | no | Price for job. | 11.50 |
| category | String |  | yes | no | Category job belongs to. | Cleaning |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | Messages | | | | | |
| **Description** | Stores messages sent on site | | | | | |
| **Name** | **Type** | **Field Length** | **Mandatory?** | **Unique?** | **Description** | **Example** |
| message | String |  | yes | no | Reference to id in user model | This is a message. |
| users | Array |  | yes | no | Credit card number | John, Aaron |
| sender | ObjectId |  | yes | no | Reference to id in user model | Aaron |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | Ratings | | | | | |
| **Description** | Stores information on credit card | | | | | |
| **Name** | **Type** | **Field Length** | **Mandatory?** | **Unique?** | **Description** | **Example** |
| jobID | ObjectId |  | yes | yes | Reference to id in jobs model | 1 |
| freelancerID | ObjectId |  | yes | yes | Reference to id in user model | 1 |
| userID | ObjectId |  | yes | yes | Reference to id in user model | 1 |
| ratings | Number |  | yes | no | Job rating | 5 |
| feedback | String | 255 | no | no | First name of user | Shawn |
| date | Date |  | yes | no | Date rating was added |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | ReportJob | | | | | |
| **Description** | Stores information on job reports | | | | | |
| **Name** | **Type** | **Field Length** | **Mandatory?** | **Unique?** | **Description** | **Example** |
| job\_id | ObjectId |  | yes | yes | Reference to id in job model | 1 |
| jobTitle | String |  | yes | no | Title of job | Clean Bros |
| user\_id | ObjectId |  | yes | yes | Reference to id in user model | 1 |
| reason | String |  | yes | no | Security code for card | Inappropriate Content |
| reportCategory | String |  | yes | no | Type of report | Plagiarism |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | Services | | | | | |
| **Description** | Stores information on services | | | | | |
| **Name** | **Type** | **Field Length** | **Mandatory?** | **Unique?** | **Description** | **Example** |
| Title | String |  | yes | yes | Services title | Cataloguing |
| description | String |  | yes | yes | Description of service | Catalogue things for you |
| thumbnail | String |  | yes | no | Image of service | Clean.jpeg |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | Users | | | | | |
| **Description** | Stores information on credit card | | | | | |
| **Name** | **Type** | **Field Length** | **Mandatory?** | **Unique?** | **Description** | **Example** |
| username | ObjectId |  | yes | yes | Reference to id in user model | 1 |
| email | String |  | yes | yes | Credit card number | \*\*@mail.com |
| password | String |  | yes | no | Expiry date of credit card | 12345678 |
| is\_staff | String |  | yes | no | Security code for card | Yes |
| is\_active | String |  | yes | no | Is user online | yes |
| date\_joined | Date |  | yes | no | Date user created account |  |
| jobs | Array of ObjectId |  |  |  | Reference to id in job model |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | userProfile | | | | | |
| **Description** | Stores user profile information | | | | | |
| **Name** | **Type** | **Field Length** | **Mandatory?** | **Unique?** | **Description** | **Example** |
| user\_id | ObjectId |  | yes | yes | Reference to id in user model | 1 |
| profile | String |  | yes | no | User’s profile |  |
| bio | String |  | yes | no | Information about yourself or status | Looking for young entrepreneurs |
| credcard | ObjectId |  | yes | yes | Reference to credit card number in CreditCard model | 0000 0000 0000 0000 |

**System Administration**

**Security**

Denying access to admin pages depending on if user account is staff or not.

Hiding sensitive information from source code by storing it in .env file.

**Backup and Restore**

Database information is backed up on mongo atlas cloud servers.

All code is regularly backed up on GitHub, so it is available if something happens locally.

**Testing**

|  |  |
| --- | --- |
| Test | Date |
| **Done By**: Aaron Hazzard | |
| **Objective(s) of Test**: Ensure Everything Works Properly | |
| **Recommended Conditions**:   * Accessible from a variety of devices, including desktop computers, laptops, tablets, and smartphones. * Responsive, meaning that it should adapt to the size of the device it is being viewed on. * Easy to use, with clear navigation and a user-friendly interface. * Fully Functional | |
| **Input Parameter(s)**:   * User credentials: The username and password of a registered user. * Job posting information: The title, description, and skills required for a job posting. * Freelancer profile information: The skills, experience, and availability of a freelancer. * Payment information: The payment method and amount for a job. * Service information: Services should be inserted, updated and/deleted. * About us information: About us content should be inserted and updated. * Categories: Categories should be inserted or removed based on decision from CEO. | |
| **Test Procedure**:  **Login test:**   * Navigate to login page. * Attempt to log in to the website with valid user credentials. * Attempt to log in to the website with invalid user credentials. * Verify that the user can access the website's features after logging in. * Verify that the user is not able to access the website's features after logging in with invalid credentials.   **Job posting test:**   * Navigate to create job page. * Insert job information. * Create a job posting by pressing the create job button. * Verify that the job posting is displayed on the job page. * Verify that users can view the job posting.   **Ratings:**   * Navigate to jobs page. * Select a job. * Select stars between 1 and 5. * Insert feedback.   **Edit Ratings:**   * Navigate to jobs page. * Select a job. * Select edit review. * Repeat **Ratings test.** * Confirm that ratings were updated by refreshing the page if it doesn’t automatically.   **Payment:**   * Navigate to jobs page. * Select a job. * Select the continue button. * Check validation by selecting confirm & pay with empty fields. * Check validation by selecting confirm & pay with invalid input formats * Verify that the website can process the payment for the job by sending an invoice in their email.   **Service information:**   * Navigate to admin page by logging into admin account. * Navigate to configuration. * View the Services form by clicking the plus (+) icon. * Insert a new service. * Select create service. * Validate by leaving fields empty and ensure that all are required. * Update an existing service by selecting the service title. * Select update. * Validate by leaving fields empty but fields are optional **only if there is at least one data inserted.** * Delete a service by selecting the service title. * Select delete. * Verify that the services are displayed correctly on the website and admin page.   **About us information:**   * Navigate to admin page by logging into admin account. * Navigate to configuration. * Insert new about us content. * Update existing about us content. * Select update. * Verify that the about us content is displayed correctly on the website.   **Categories:**   * Navigate to admin page by logging into admin account. * Navigate to configuration. * Insert a new category. * Remove a category. * Select update. * Verify that the categories are displayed correctly on the website.   **User Profile:**   * Log in to the website with a valid user account. * Click on your name in the top right corner of the website. * Click on the Profile tab. * Enter the new information and click on the Save button. * Verify that the new information is displayed correctly on your user profile page. | |
| * **Expected Result(s)**: * **Login test:** Redirect user to jobs page. * **Job posting test:** Display a success message and redirect user to jobs page. * **Ratings:** Insert ratings to the database and refresh the page to display your ratings. * **Edit Ratings:** Display the hidden ratings form and allow user to update their ratings then refresh the page on submit. * **Payment:** Send invoice to user email. * **Service Information:** Insert, Update, Delete Services and view it dynamically on the home page. * **About us information:** Insert, Update, about us content and view it dynamically on the about us page. * **Categories:** Insert, Update, Delete Categories and view it dynamically on the home, jobs and admin page. * **User Profiles:** Update Information. | |
| **Actual Result(s)**:   * **Login test:** passed. * **Job posting test:** passed. * **Ratings:** passed. * **Edit Ratings:** passed. * **Payment:** passed. * **Service Information:** passed. * **About us information:** passed. * **Categories:** passed. * **User Profile:** passed. | |
| **Change’s Required**:  Access Control is the only change required as not all pages have that implementation | |

**Implementation Plan**

**Objectives**The objective of this implementation plan is to outline the steps that will be taken to implement the website. The website will be implemented in a phased approach, with each phase being completed before the next phase begins.

**Phase 1: Planning**The first phase of the implementation plan will involve planning the implementation of the website. This will include identifying the resources that will be needed, developing a timeline for the implementation, and creating a communication plan.

**Phase 2: Development**The second phase of the implementation plan will involve developing the website. This will include designing the website, developing the website's content, and testing the website. **Phase 3: Testing**The third phase of the implementation plan will involve testing the website. This will include testing the website for functionality, usability, and security.

**Phase 4: Deployment**The fourth phase of the implementation plan will involve deploying the website. This will include making the website available to users.

**Phase 5: Maintenance**The fifth and final phase of the implementation plan will involve maintaining the website. This will include updating the website's content as needed and resolving any issues that may arise.

**Resources**The following resources will be needed to implement the website:

* **Project manager (Aaron)**
* **Web developer (Aaron, Raushawn)**
* **Graphic designer (Aaron)**
* **Content writer (Aaron)**
* **Client (Brandon Best)**

**Communication Plan***The following communication plan has been developed for the implementation of the website:*

* Weekly status meetings will be held with the project manager, web developer, graphic designer, content writer, and client.
* A weekly update will be sent to the client and users to keep them updated on the progress of the website.
* A final report will be submitted to the client and users upon completion of the project.

**Risks**The following risks have been identified for the implementation of the website:

* The website may not be completed on time.
* The website may not be 100% user-friendly.
* The website may not be 100% secure, especially with access control.

**Lessons Learned**

**Technical**

Responsive Design Skills

Planning and Requirement Gathering

**Managerial**

Communication Skills

Risk Management

Project Management

**Recommendations**

Do not procrastinate on completing functionality for website.

Do not take too long to test implemented functionality.

Set deadlines for tasks during process.

**User Procedures**

**Registration**

Visit website registration page.

Fill out registration form with necessary information.

Submit form to register.

**Edit User Profile**

Change information as seen fit.

Upload profile picture.

Save changes to account.

**Explore Available Jobs**

Login to site with your credentials.

Browse through available job listings or search for specific categories, etc.

**Create Jobs**

Click on create job button.

Fill out necessary form.

**Chat with Seller**

Click on job details page.

Click chat now button.

Communicate with seller for information on job they provide.

**Report Job**

Click on job, you find inappropriate.

Click report button.

Fill out form and click submit.

**Payment**

Click on job which you would like to request.

Click on continue button.

Fill in relevant information.

Click confirm and pay.

Invoice will be emailed shortly after.

**Contact Us**

Fill in form with message you would like to send to us.

Click send and message will be sent to us and your email account.

**Appendix**

**Tools Used to Gather User Requirements**

Interview with CEO