Fufubay, Inc.

Fufubay
Software Requirements Specification

Version 2.0

Group Members: Aiza Tahir, John Royal,
Mohammad Zikrya, Raha Sumya, & Uswa Qamar

Revision History

Date	Version	Description	Authors
10/11/2022	1.0	Initial specification draft.	Aiza Tahir, John Royal, Mohammad Zikrya, Raha Sumya, & Uswa Qamar
11/22/2022	2.0	Phase II design report.	Aiza Tahir, John Royal, Mohammad Zikrya, Raha Sumya, & Uswa Qamar

Table of Contents

Software Requirements Specification

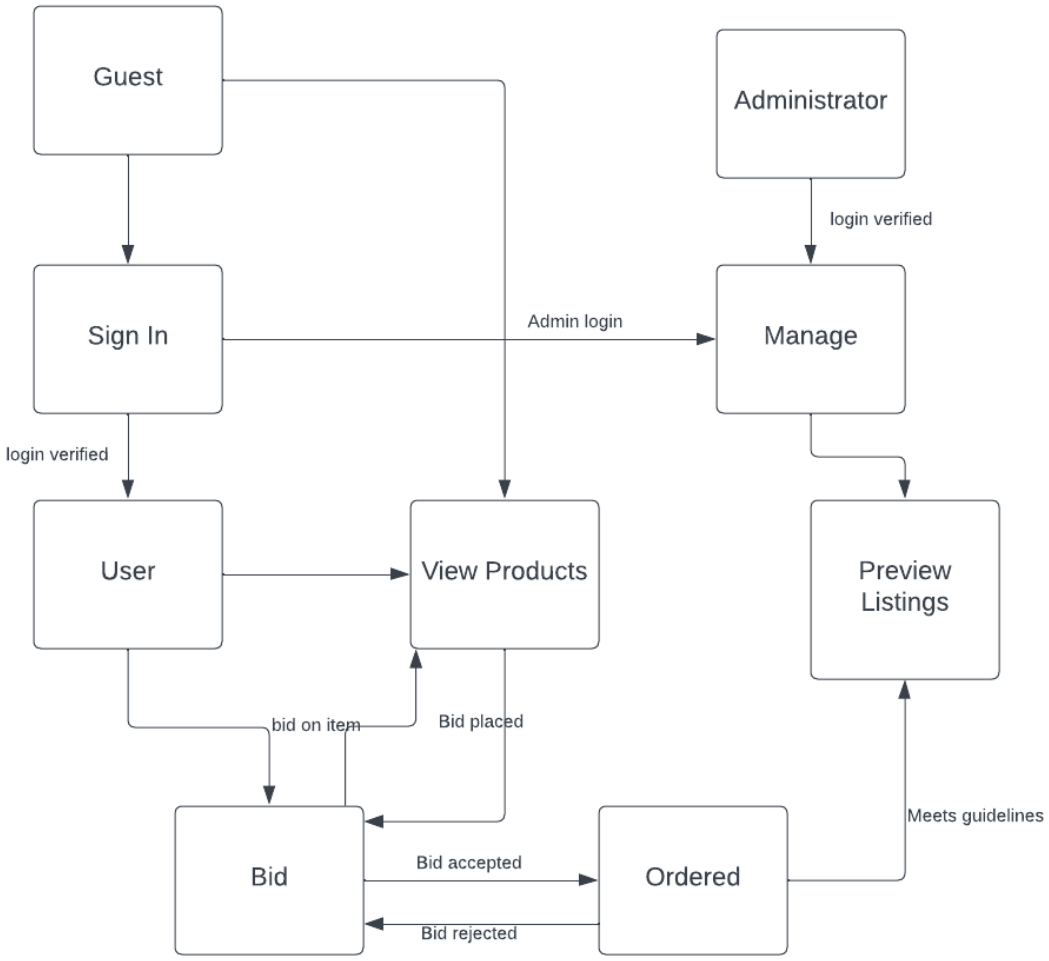
1. Introduction

The purpose of Fufubay is to facilitate the auctioning of products online. It will use a Consumer-to-Consumer business model, which requires an online platform where consumers can communicate and trade anytime. This platform will bring together buyers and sellers in such a way that sellers can list their goods for sale and buyers can place their bids for the products. Both parties can browse through all the items on the list.

The application currently under development is a prototype for an online auction website. As a result, the application will include the fundamentals of such a website, including authentication, product listings, seller reviews, and rudimentary fraud protection. However, the application may not include the full range of features expected of a production e-commerce website, such as frequently-asked questions, customer service communications, shipping, returns, notifications, and product recommendations.

This report provides the requirements and specifications for Fufubay, an online auction website where users can buy and sell products. Users can search through the platform and bid on specific items that are listed for sale. This report covers the purpose, scope, definitions, references, and overview of our application.

1.1. Collaboration Class Diagram



2. Use Cases

2.1. Scenarios & Diagrams

Each use case is depicted in a use-case diagram which takes into account both normal and exceptional scenarios. A collaboration class diagram, along with Petri-net, is provided for each of the following use cases.

2.1.1. Account Status

Upon entering the site, viewers are given the option to sign in or sign up. Admins or users can login by entering their username and password, their login details are authorized by the backend system and they're taken to their dashboard. On the other hand, guests can register to become users for the site. When signing up, they will be asked for their email and to create a username

and password. Once all fields are filled out correctly, their details are sent to an admin to verify to activate their account.

Figure 2.1.1a: Use Case Diagram

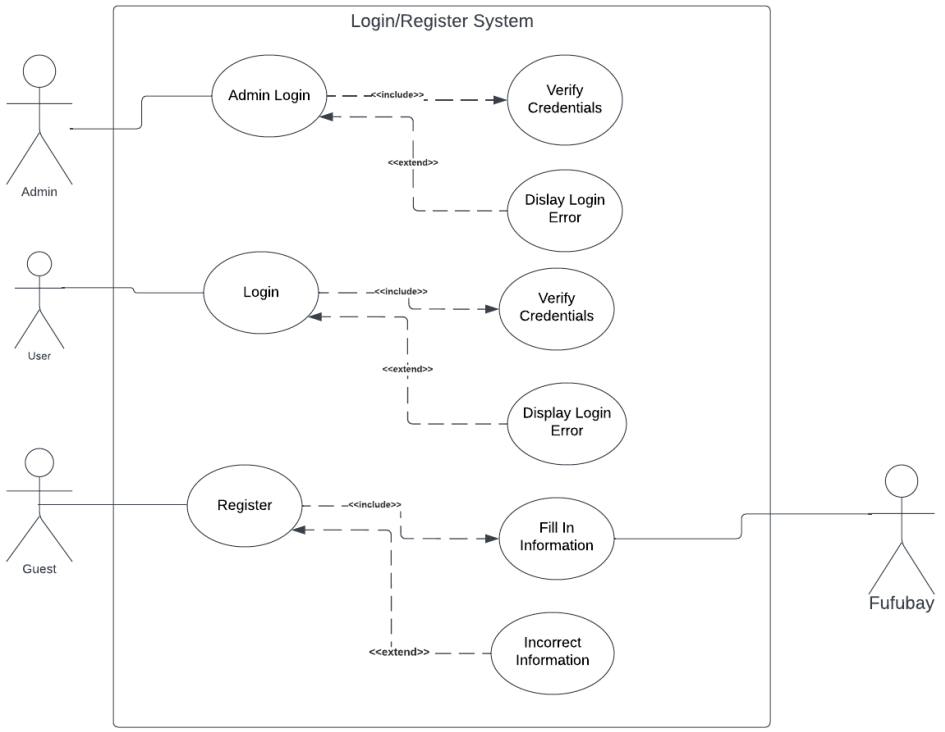


Figure 2.1.1b: Petri-Net Diagram

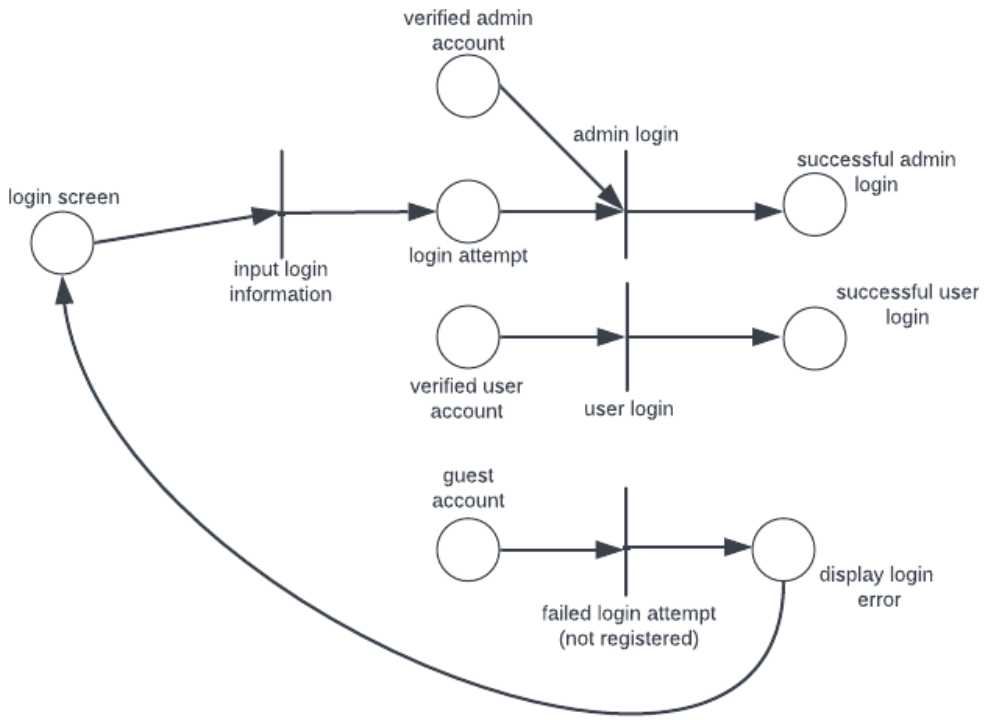


Figure 2.1.1c: Collaboration Class Diagram for Sign In

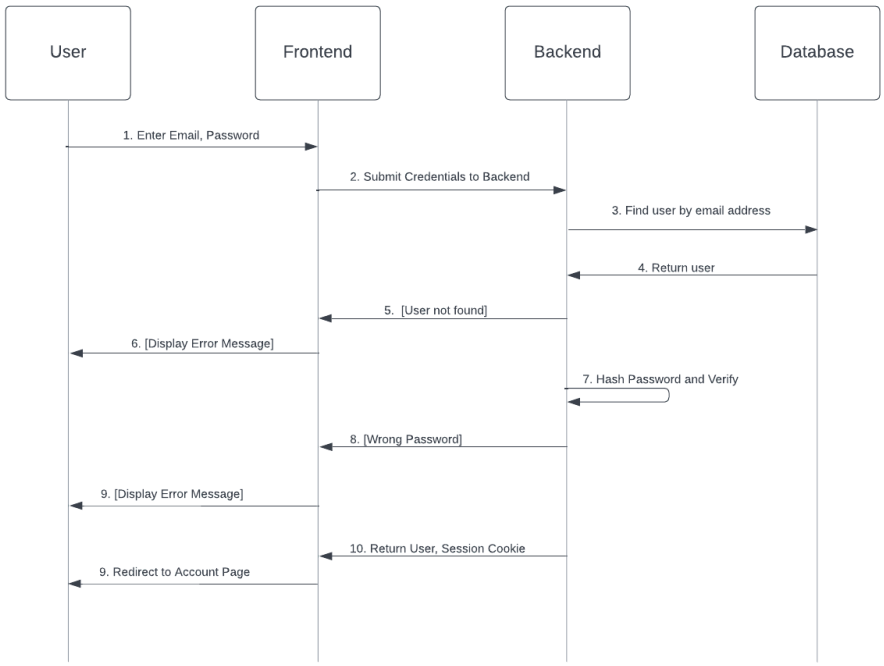
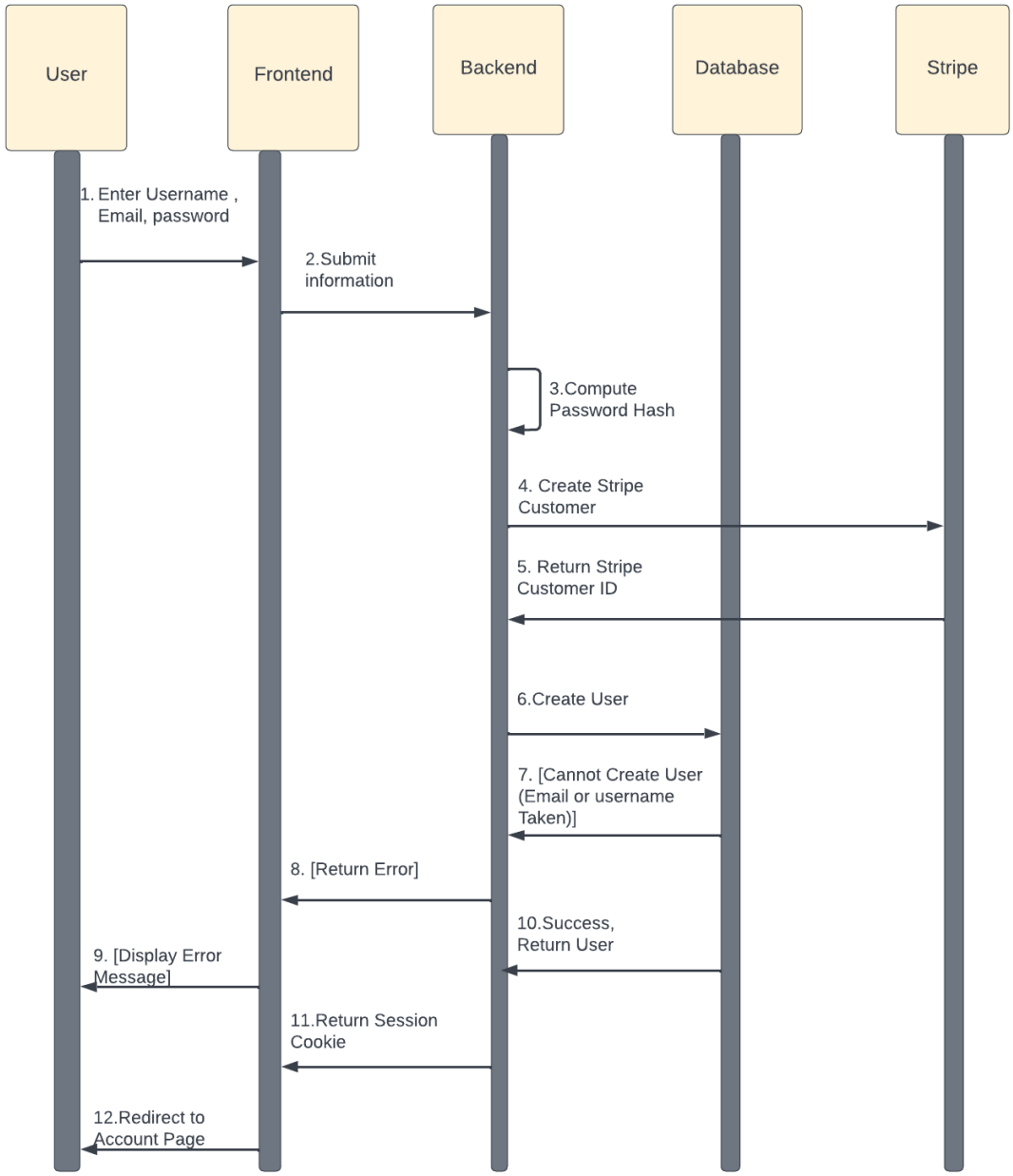


Figure 2.1.1d: Collaboration Diagram for Creating an Account



2.1.2. Product Viewing/Listing

Once a customer searches for a particular product, the system fetches products that match search specifications. If a specific product is not available, a message will appear notifiying the user that the product was not found. Users are also allowed to list products of their own after providing all necessary details about it, which are then also displayed for customers to view and purchase.

Figure 2.1.2a: Use Case Diagram

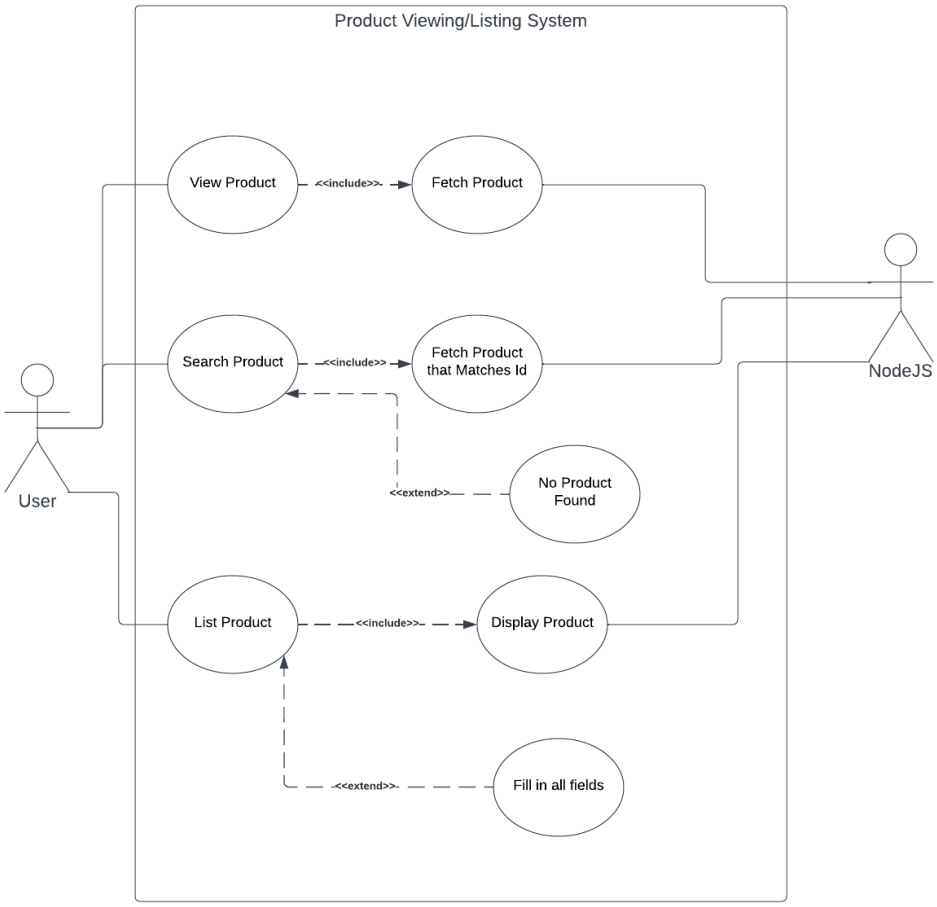


Figure 2.1.2b: Petri-Net Diagram

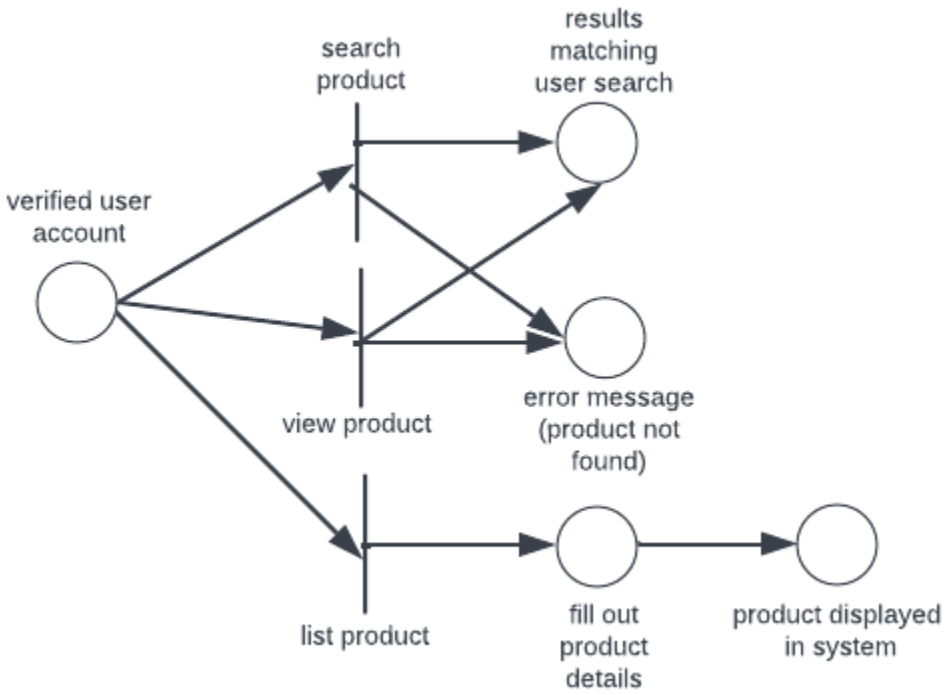
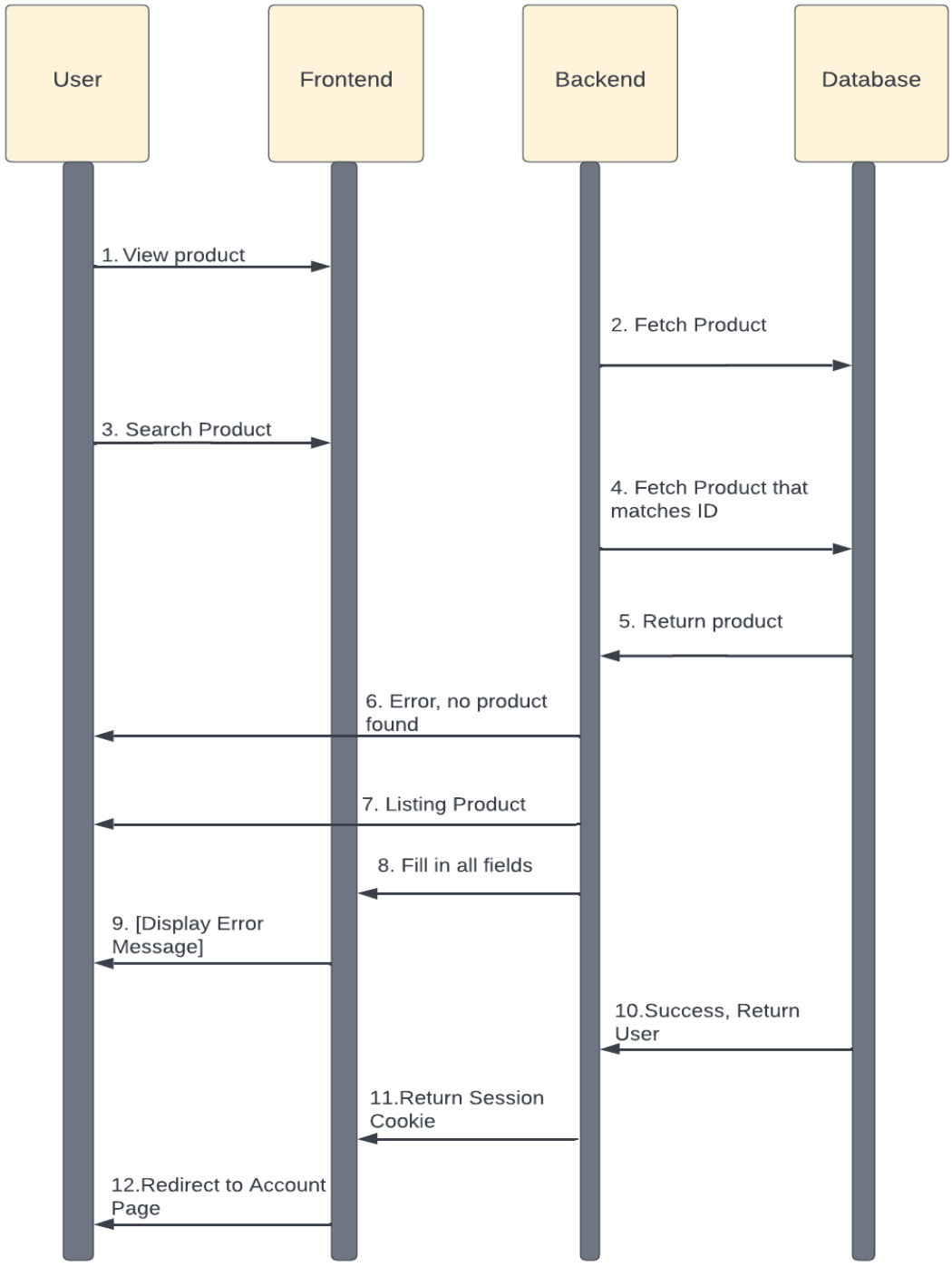


Figure 2.1.2c: Collaboration Class Diagram



2.1.3. Bidding on a Product

Users are permitted to bid on live auctions provided that they are signed in and have added the relevant payment information to their account. Guest users who would like to bid are prompted to create an account. Users without a payment card, address, and phone number linked to their account are redirected to the settings page where they can add that information.

To bid on an auction, the bid amount must be greater than the current highest bid on the product; otherwise, the bid will not be submitted. When submitting the bid, a hold is placed on the user's payment card. If the payment declines, the bid is rejected and the user is notified accordingly. When the auction is over, the winning bidder will be charged for their bid amount, while all other bidders will have the hold released from their payment card.

If the user wins the auction, the user's payment card is charged for their bid amount. For all other users who bid on the auction, the hold is released.

Figure 2.1.3a: Use-Case Diagram

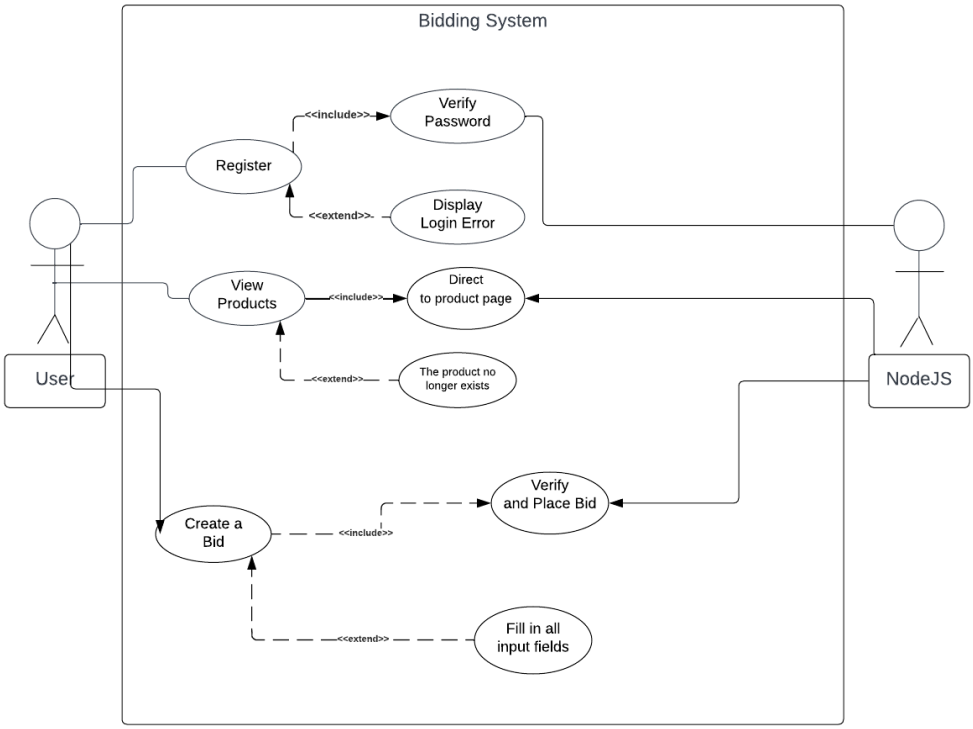


Figure 2.1.3b: Petri-Net Diagram

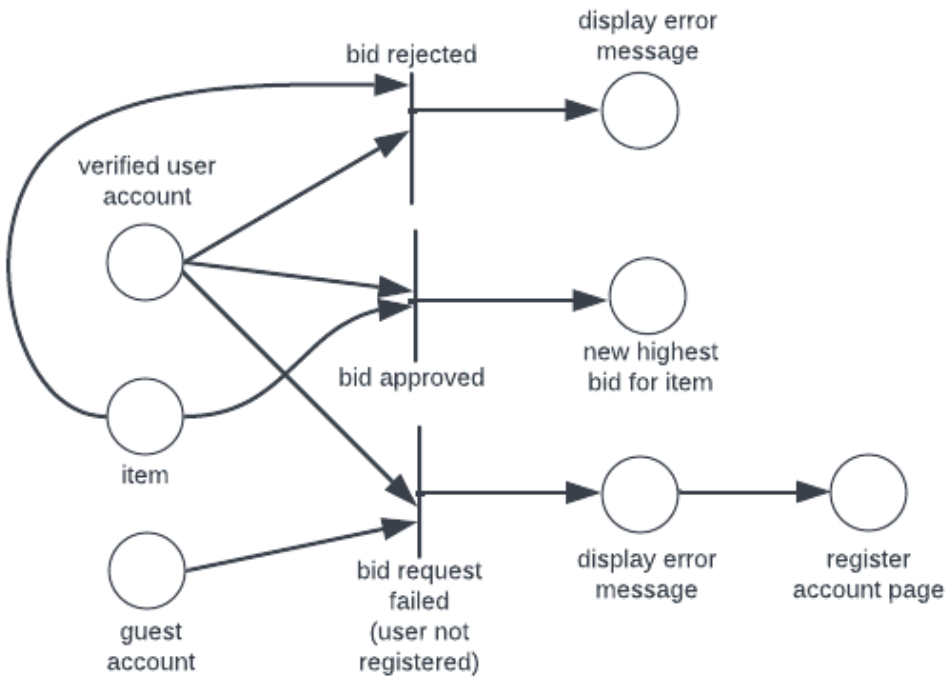
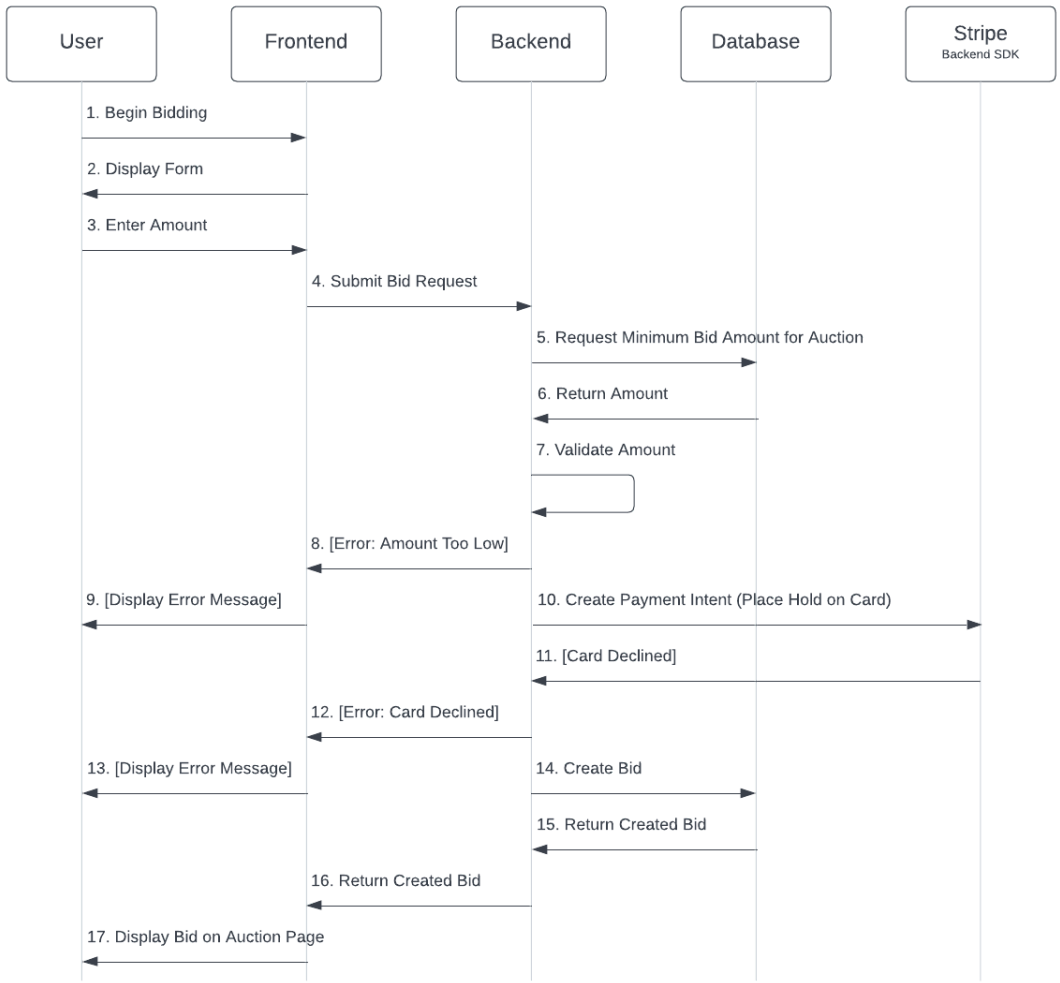


Figure 2.1.3c: Collaboration Diagram



2.1.4. Updating Payment Information

The Update Payment Information use case describes the process of linking an address and payment method to a user's account. This enables the Bid on a Product use case by providing the payment information required to process a bid.

In accordance with industry best practices for handling payment cards and other related information, Fufubay uses Stripe, a payment processing platform, to collect and process payments securely. For security and liability reasons, this information is not stored in our database or processed by our backend.

From the Settings page, users may select the "Update Payment" or "Update Address" button to initiate the update process. The frontend orders the backend to obtain a Setup Intent from Stripe. The backend creates the Setup Intent using the Stripe backend SDK and returns the information to

the frontend. The frontend provides the Setup Intent credentials to the Stripe frontend SDK, known as Stripe.js. Stripe.js then provides an sandboxed form in which users may enter their updated information. When the user is finished entering their information, they will select the “Save” button, ordering Stripe.js to save the updated payment information by “confirming” the Setup Intent. Finally, the form is hidden and the updated information is displayed on the user’s settings page.

Figure 2.1.4a: Use-Case Diagram

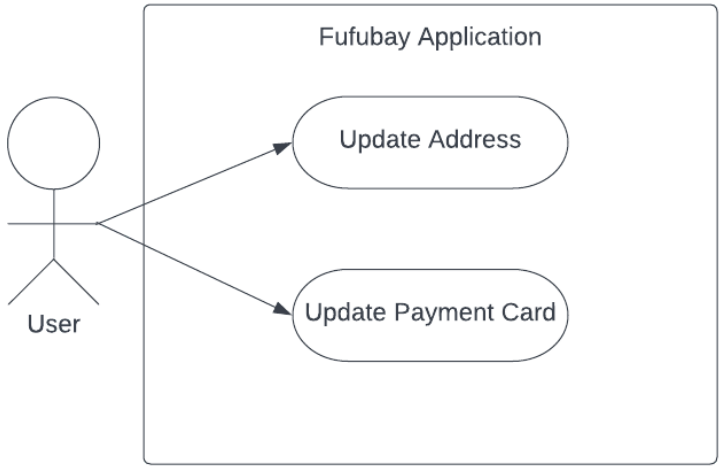
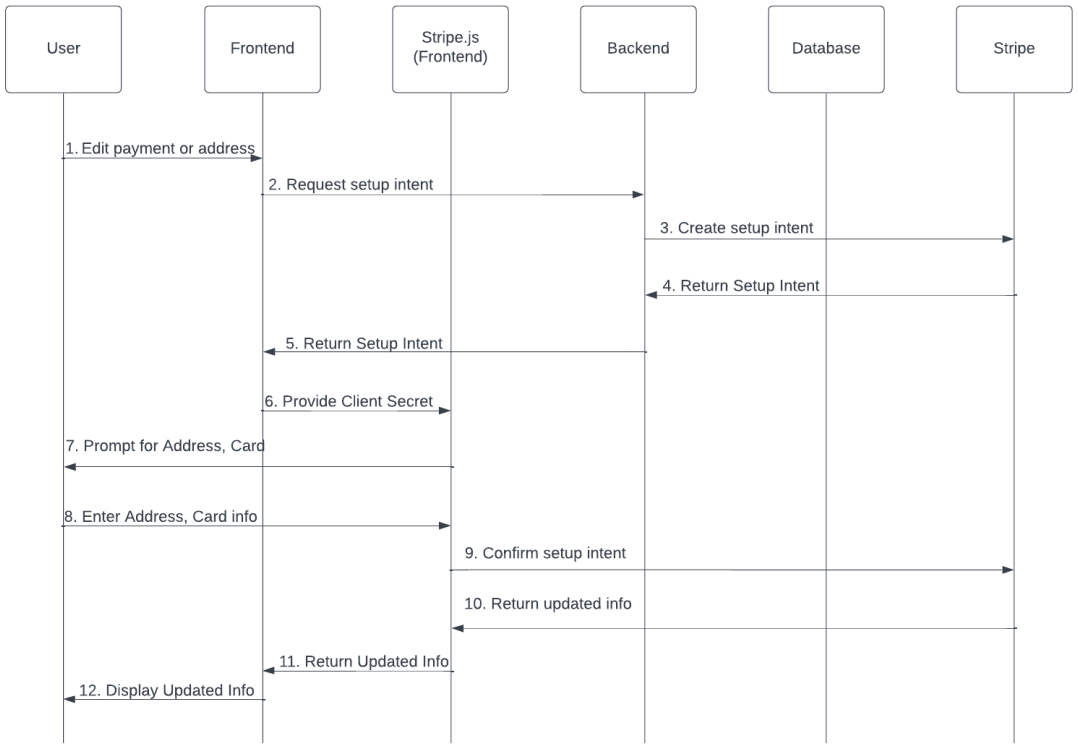


Figure 2.1.4b: Collaboration Diagram



2.1.5. Update Account Information

Registered users are able to view and update their account information, including their profile image, username, email, password, and phone number. To update this information, navigate to the Settings page and select the relevant option, such as “Edit Username” or “Edit Password”. The frontend displays a form for inputting the updated information, then submits the information to the backend so it can be saved in the database.

This use case is separate from updating the user’s address and payment card because of the additional care required to process payment cards.

Figure 2.1.5a: Use-Case Diagram

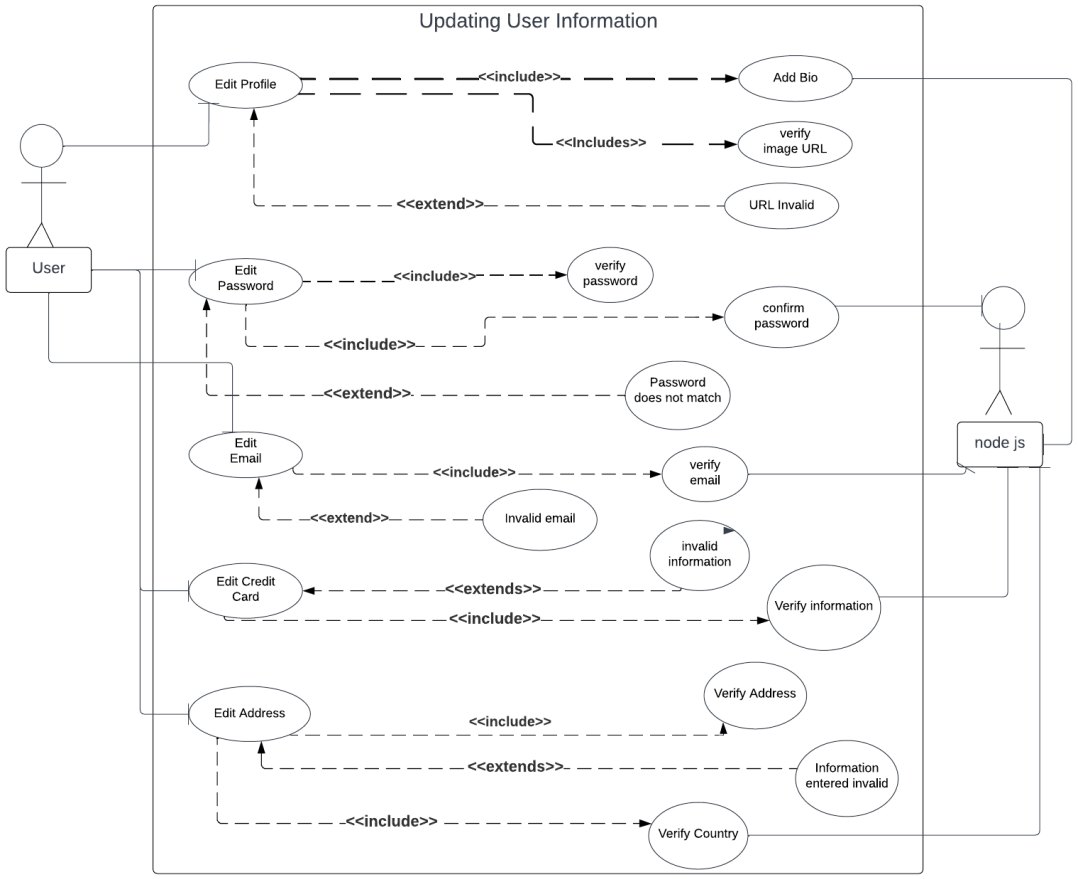


Figure 2.1.5b: Petri-Net Diagram

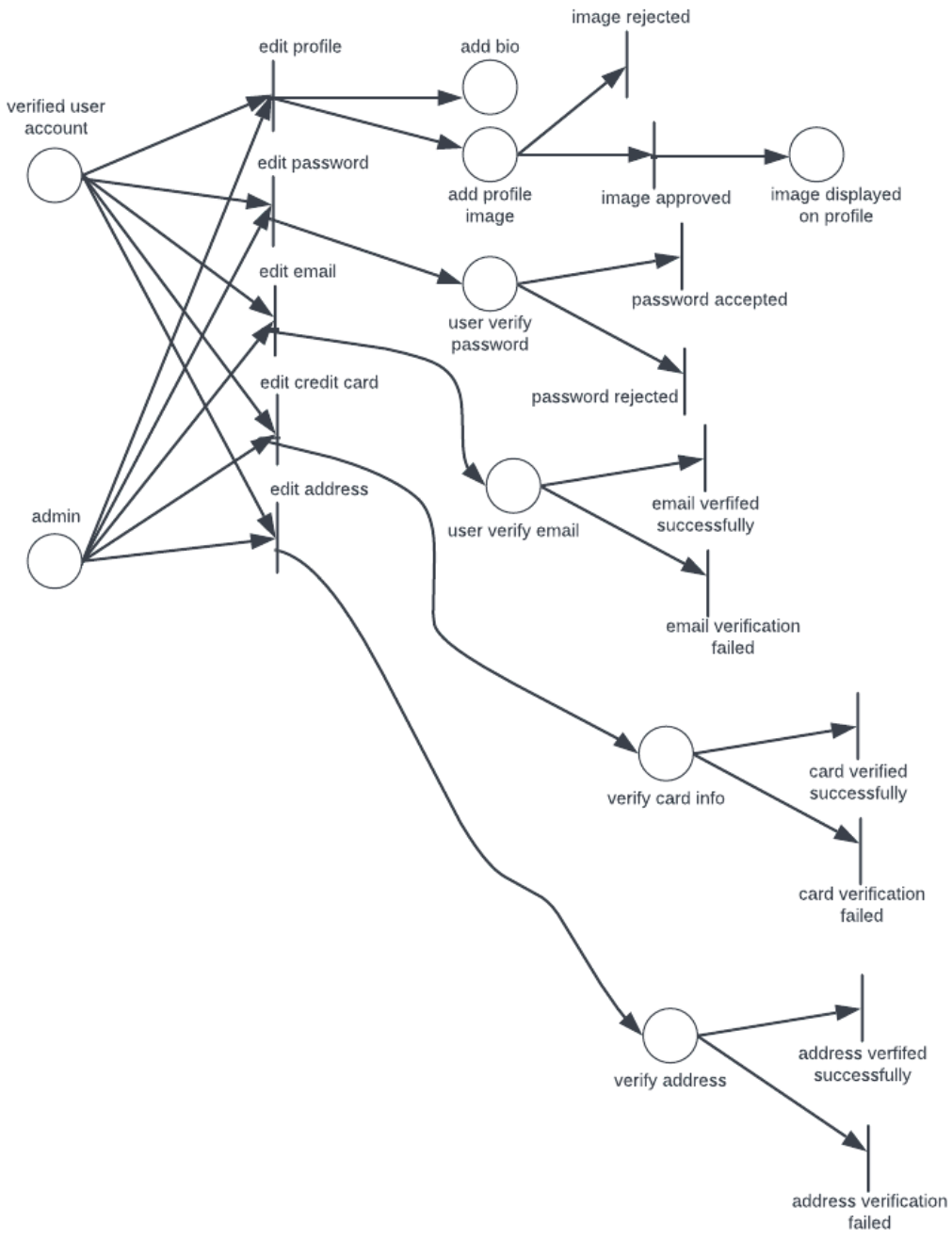
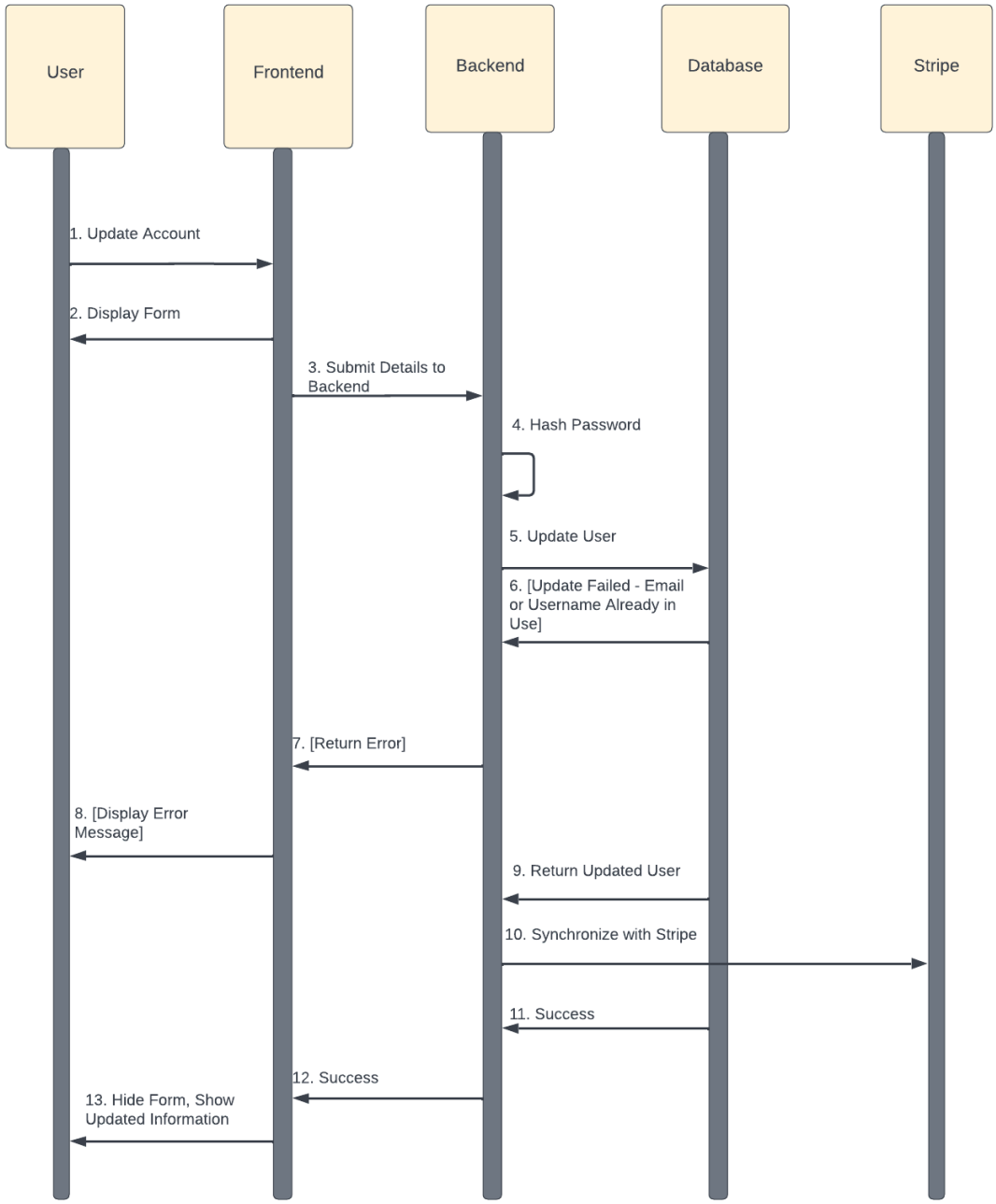
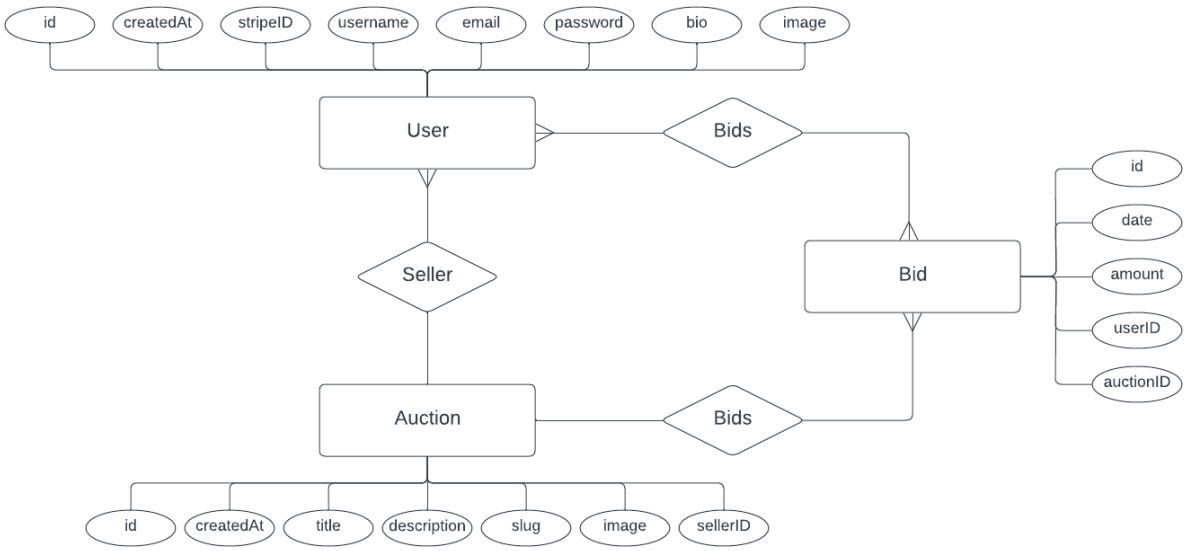


Figure 2.1.5c: Collaboration Class Diagram



3. E-R Diagram



4. Detailed Design

4.1. User Methods

4.1.1. Create an Account

Users have the option to sign up to become users of the account. For the frontend portion of this, in the navbar, they can click on the sign up button that will pop up a modal, to add in their information to become members of the account. For now, once they hit sign up, their credentials will be recorded into the database.

4.1.2. Sign In

Registered users can enter in their user information to access their account. In the navbar, there is a login button that brings up a modal to enter their login details. The backend will scan the database to match their account information. Once their information is matched they are brought to their account page.

4.1.3. Create an Auction

This method is implemented in the frontend using React. When the user clicks Sell With Us, they will be directed to the auction page, where they can fill in the prompts and upload the auction. After the user presses the create button, the frontend communicates to the backend, to add the information to the database and display the listing onto the homepage.

4.2. Backend

4.2.1. GET /api/auctions

This API route is used by the home page to retrieve a list of current auctions in reverse-chronological order. Additionally, if the `sellerID` query parameter is provided, this API route is also used to populate the user's profile page with a list of their auctions. Prisma is an object-relational mapping (ORM) tool that facilitates database interactions. A `res.success()` response is an HTTP response with a 200 status code.

```
async handler (req, res) {
      const where: Prisma.AuctionWhereInput = {}
      if (typeof req.query.sellerID === 'string') {
        where.sellerID = parseInt(req.query.sellerID)
      }
      const auctions = await prisma.auction.findMany({ where,
orderBy: { createdAt: 'desc' } })
      return res.success(auctions)
    }
}
```

4.2.2. POST /api/auctions

This route is used to create a new auction. Users must be authenticated to complete this action. The "slug" data property is a version of the title with spaces and special characters replaced with spaces, for use in public auction URLs (e.g. "Samsung Galaxy S22" converts to "samsung-galaxy-s22"). A `res.unauthorized()` result returns a 401 HTTP status code, while a `res.created()` result returns a 201 status code.

```
async handler (req, res) {
  if (req.user == null) {
    return res.unauthorized()
  }
  const data: Prisma.AuctionCreateInput = {
    title: req.body.title,
    description: req.body.description,
    slug: req.body.title.toLowerCase().replace(/ /g,
'-').replace(/[^\w-]+/g, ''),
    seller: {
      connect: { id: req.user.id }
    }
  }
  const auction = await prisma.auction.create({ data })
  return res.created(auction)
}
```

4.2.3. GET /api/auctions/:id

This is used to retrieve a current auction. Information about the seller is included in the API request. A `res.notFound()` response includes a 404 status code.

```
async handler (req, res) {
  const auction = await prisma.auction.findUnique({
    where: { id: parseInt(req.params.id) },
    include: { seller: true }
  })
  if (auction != null) {
    return res.success(auction)
  } else {
    return res.notFound()
  }
}
```

4.2.4. PUT /api/auctions/:id

This is used to update a current auction. To complete this action, the user must be authenticated using the same account that was used to create the auction.

```
async handler (req, res) {
  const auction = await prisma.auction.findUniqueOrThrow({
    where: { id: parseInt(req.params.id) },
    include: { seller: true }
  })
  if (req.user == null || auction.sellerID !== req.user.id) {
    return res.unauthorized()
  }
  await prisma.auction.update({
    where: { id: auction.id },
    data: Object.assign(auction, req.body)
  })
  return res.success(null)
}
```

4.2.5. DELETE /api/auctions/:id

This is used to delete an auction. This will be used to remove auction listings that have been rejected by an administrator.

```
async handler (req, res) {
  const auction = await prisma.auction.findUniqueOrThrow({
    where: { id: parseInt(req.params.id) }
  })
  if (req.user == null || auction.sellerID !== req.user.id) {
```

```
    return res.unauthorized()
  }
  await prisma.auction.delete({ where: { id: auction.id } })
  return res.success(null)
}
```

4.2.6. GET /api/users

This obtains a list of user accounts. This will be used for administrative purposes.

```
async handler (req, res) {
  const users = await prisma.user.findMany()
  return res.success(users)
}
```

4.2.7. POST /api/users

This is used to create a new user account. The `normalizeEmail()` function ensures that minor errors in an email address entry (e.g. capitalization and whitespace) do not prevent users from authenticating successfully. The `scrypt.hash()` function uses the Scrypt password hashing algorithm to compute a secure hash of the user's password.

```
async handler (req, res) {
  const { username, email, password } = {
    username: req.body.username as string,
    email: normalizeEmail(req.body.email),
    password: await scrypt.hash(req.body.password)
  }
  const stripeCustomer = await stripe.customers.create({
    email
  })
  const user = await prisma.user.create({
    data: {
      username,
      email,
      password,
      stripeCustomerID: stripeCustomer.id
    }
  })
  await req.signIn(user)
  return res.created(user)
}
```

4.2.8. GET /api/users/:id

This is used to obtain a single user profile.

```
async handler (req, res) {
  const id = parseInt(req.params.id)
  const user = await prisma.user.findUnique({
    where: { id }
  })
  if (user == null) {
    return res.notFound()
  }
  return res.success(user)
}
```

4.2.9. PATCH /api/users/:id

This is used to update an existing user account. If the user would like to update their password, the `scrypt.hash()` function is used to compute a hash of the new password. If the user is updating their email address, this change will be communicated to the Stripe backend as well.

```
async handler (req, res) {
  const id = parseInt(req.params.id)
  if (req.user == null || id !== req.user.id) {
    return res.unauthorized()
  }
  if (typeof req.body.password === 'string') {
    req.body.password = await scrypt.hash(req.body.password)
  }
  const user = await prisma.user.update({
    where: { id },
    data: req.body as User
  })
  if (typeof req.body.email === 'string') {
    await stripe.customers.update(user.stripeCustomerID, {
email: user.email })
  }
  await req.signIn(user)
  return res.success(user)
}
```

4.2.10. DELETE /api/users/:id

This is used to delete a user account, either because the user would like to leave the website or because they have been forcibly removed by an administrator. Both the user in the Fufubay database and the corresponding customer record in the Stripe backend are removed. Once the user is deleted, a new account cannot be created with the same email address, effectively banning the user from the website.

```
async handler (req, res) {
  const id = parseInt(req.params.id)
```

```
  if (req.user == null || id !== req.user.id) {
    return res.unauthorized()
  }
  const user = await prisma.user.findUniqueOrThrow({
    where: { id }
  })
  await Promise.all([
    prisma.user.delete({ where: { id } }),
    stripe.customers.del(user.stripeCustomerID)
  ])
  return res.success(null)
}
```

4.2.11. POST /api/auth/sign-in

To authenticate a user, this finds the user in the database by their email address, then uses the `scrypt.compare()` function to verify that the user entered the correct password. If the user is authenticated successfully, the `req.signIn()` helper function is used to create and return a new session cookie that identifies the authenticated user for future requests.

```
async handler (req, res) {
  const { email, password } = req.body as { email: string,
password: string }

  const user = await prisma.user.findUnique({
    where: { email }
  })
  const isAuthenticated = user != null && await
scrypt.compare(password, user.password)

  if (isAuthenticated) {
    await req.signIn(user)
    return res.success(user)
  } else if (user == null) {
    return res.notFound('We couldn’t find that account.')
  } else {
    return res.unauthorized('Wrong password.')
  }
}
```

4.2.12. GET /api/auth/sign-out

This uses the `req.signOut()` helper function to sign out the current user by deleting their session cookie.

```
async handler (req, res) {
  await req.signOut()
```

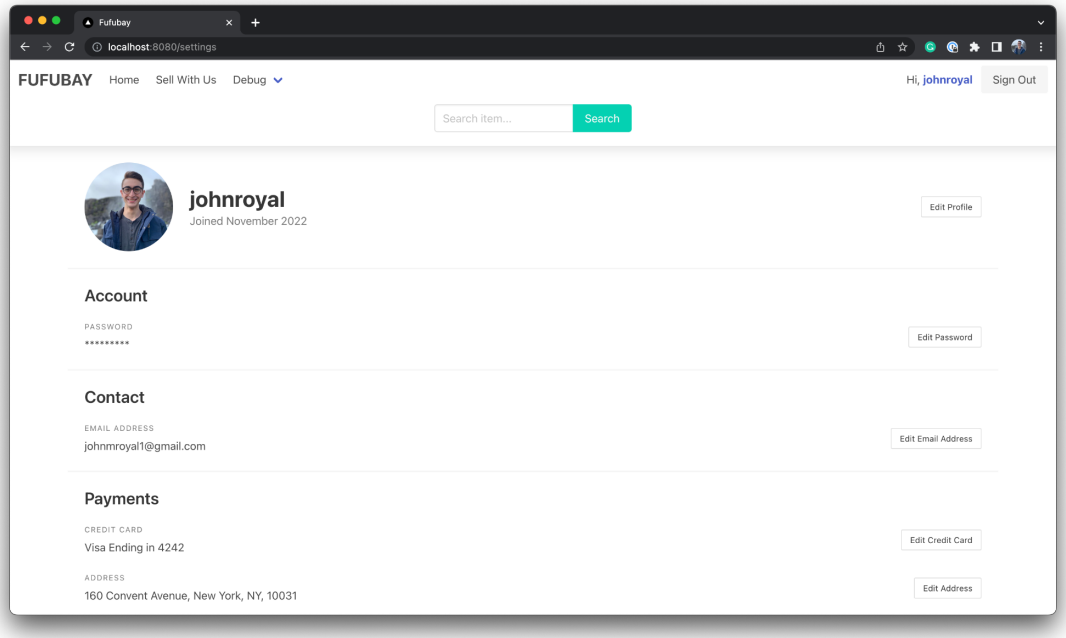
```
  return res.success(null)
}
```

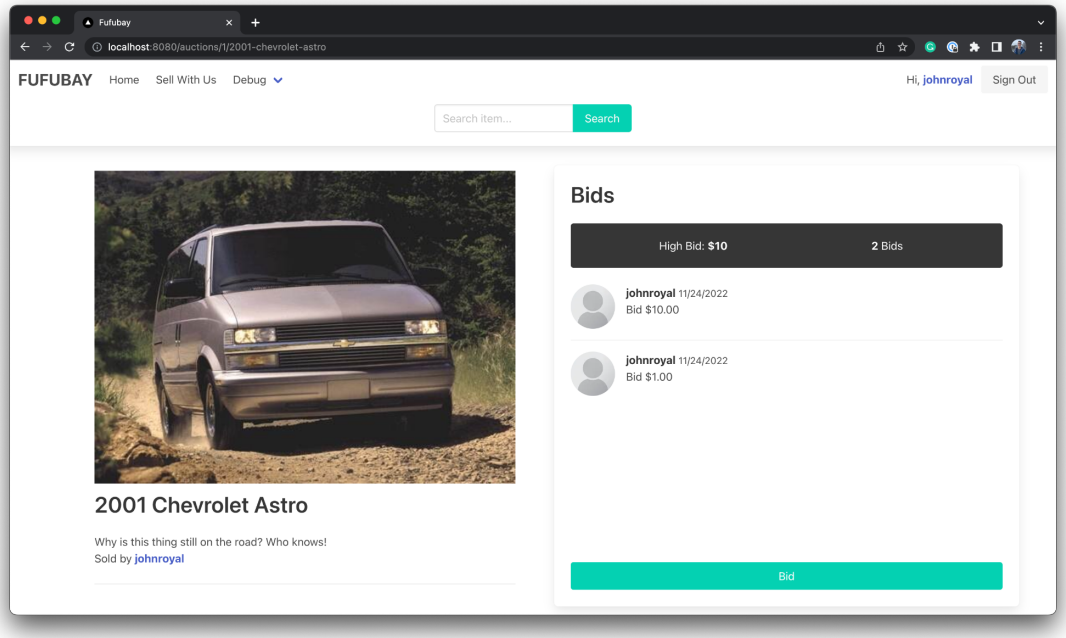
5. System Screens

This is the prototype of the user interface of the online store. This is the format of the home page, auction page, and transactions. For the design of this site, we used the CSS framework Bulma, which allowed us to create a seamless and appealing frontend for the site. In our homepage, we have a navbar that will redirect to the homepage, allow the user to create a new auction, and give the option to sign up or login. There is also a category component that will later filter through the listings. Besides that the home page will display all the listings posted by users.

The auction page will exhibit pictures of the product, the title, a description, and the current bids for the specific item.

Lastly, for the transaction lay out, the user can access their account settings, where they have the options to update their profile, account information, their address, and to add their payment option.


Fufubay
localhost:8080/settings
FUFUBAY
Home
Sell With Us
Debug
Hi, johnroyal
Sign Out
Search item...
Search
johnroyal
Joined November 2022
Edit Profile
Account
PASSWORD
****
Edit Password
Contact
EMAIL ADDRESS
johnmroyal1@gmail.com
Edit Email Address
Payments
CREDIT CARD
Visa Ending in 4242
Edit Credit Card
ADDRESS
160 Convent Avenue, New York, NY, 10031
Edit Address


Fufubay
localhost:8080/auctions/1/2001-chevrolet-astro
FUFUBAY
Home
Sell With Us
Debug
Hi, johnroyal
Sign Out
Search item...
Search
2001 Chevrolet Astro
Why is this thing still on the road? Who knows!
Sold by johnroyal
Bids
High Bid: $10
2 Bids
johnroyal 11/24/2022
Bid $10.00
johnroyal 11/24/2022
Bid $1.00
Bid

6. Group Meetings

Meeting #	Date	Meeting Details	Members Present
1	10/11/2022	Introduction; deciding on programming language(s)	Aiza Tahir, John Royal, Mohammad Zikrya, Raha Sumya, & Uswa Qamar
2	10/26/2022	Brainstorming for Phase I Report	Aiza Tahir, John Royal, Mohammad Zikrya, Raha Sumya, & Uswa Qamar
3	10/31/2022	Final preparation for Phase I Report submission	Aiza Tahir, John Royal, Mohammad Zikrya, Raha Sumya, & Uswa Qamar
4	11/15/2022	Discussed details regarding GitHub and Phase II Report	Aiza Tahir, John Royal, Mohammad Zikrya, Raha Sumya, & Uswa Qamar
5	11/21/2022	Preparation for Phase II Report submission	Aiza Tahir, John Royal, Mohammad Zikrya, Raha Sumya, & Uswa Qamar
6	11/22/2022	Discussed details regarding diagrams and final preparation for Phase II Report submission	Aiza Tahir, John Royal, Mohammad Zikrya, Raha Sumya, & Uswa Qamar

7. GitHub Repository

The address of the GitHub repository for this project: https://github.com/john-royal/Fufubay.