Deployment and Handover Document

B2D Venture Web Application

Prepared For:

B2D Venture

Prepared By:

FishermanFriends team

Project Team Members:

Wissarut Kanasub

Chaiyawut Thengket

Sukprachoke Leelapisuth

Version:

Version 1.0

Date:

28 November 2024

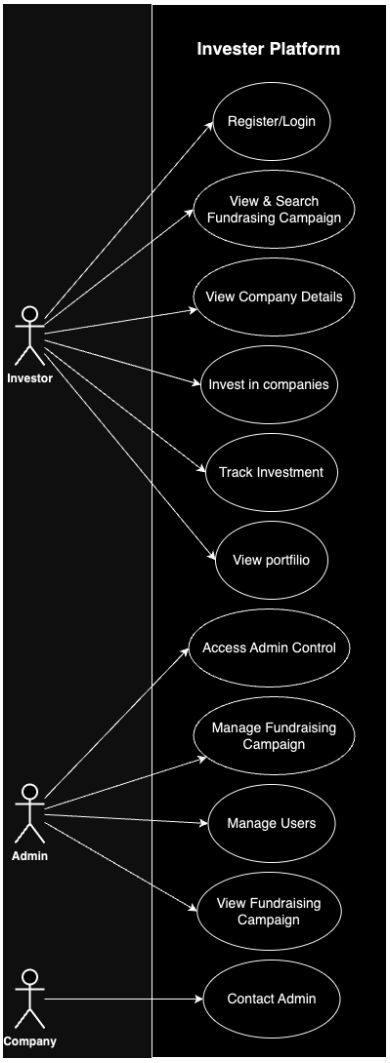
Project Background

In today's rapidly evolving world, innovations and new technologies are being developed at an unprecedented pace, sparking creativity and inspiration among people. However, turning these ideas into reality often requires a crucial resource: funding.

This online investment platform is designed to bridge the gap between startup entrepreneurs with innovative ideas but limited financial resources, and investors who are eager to support them. This platform provides a space where entrepreneurs can showcase their ideas, while investors can easily find opportunities that align with their interests.

Our goal is to create an environment that fosters connections between entrepreneurs and investors, enabling them to collaborate and bring new innovations to life that can positively impact the world.

UseCases Diagram



Breif Use cases

Investor:

- **Register/Login:** Register an account or log into the system.
- **View & Search Fundraising campaign:** Browse or search for available fundraising opportunities.
- **View Company Details:** Read details about specific companies.
- **Invest in Companies:** Select and invest in a company.
- **Track Investments:** Track the history of investments(Statements).
- **View Portfolio:** View overall investment portfolio.

Admin:

- **Access Admin Control Panel:** Access the control panel to manage fundraising and users.
- **Manage Fundraising Campaign:** Create, update, delete, and view fundraising campaigns.
- **View Users:** View user's account details.
- **View Fundraising Statements:** View detailed statements for each fundraising campaign, including user investments, timestamps, and amounts.

Company:

- **Contact Admin:** Contact the Admin to provide information and initiate a fundraising campaign.

Fully-dressed Use Cases

Use Case 1: Register/Login (Investor)

Section	Description
Primary Actor	Investor
Goal	Register an account or log into the system.
Preconditions	- The Investor must have access to the internet. - The system is online and operational.
Main Success Scenario	1. The Investor accesses the registration/login page. 2. The Investor provides their credentials (email and password) or chooses to register by providing the necessary details (name, email, password, etc.). 3. If registering, the system validates the information and creates an account. 4. If logging in, the system verifies the credentials. 5. The system logs the Investor into their account.
Postconditions	- The Investor is authenticated and gains access to the system. - If a new account is created, the account details are stored in the system.
Alternate Flows	- Invalid Credentials (Login): The system displays an error message if the credentials provided are incorrect, allowing the Investor to try again. - Duplicate Email (Registration): If the email is already registered, the system prompts the user to log in instead or use a different email.

Use Case 2: View & Search Fundraising Campaign (Investor)

Section	Description
Primary Actor	Investor
Goal	Browse or search for available fundraising opportunities.
Preconditions	- The Investor must be logged into the system.
Main Success Scenario	1. The Investor navigates to the fundraising campaigns section. 2. The Investor browses through the available campaigns or uses search filters (e.g., industry, amount needed) to find specific campaigns. 3. The Investor selects a campaign to view more details.
Postconditions	- The Investor can see a list of fundraising campaigns that match their criteria.
Alternate Flows	- No Campaigns Found: If no campaigns match the search criteria, the system displays a message indicating no results were found and suggests alternative search options.

Use Case 3: View Company Details (Investor)

Section	Description
Primary Actor	Investor
Goal	Read details about specific companies.
Preconditions	- The Investor must be logged into the system. - The Investor must have selected a fundraising campaign.
Main Success Scenario	1. The Investor selects a company from a fundraising campaign. 2. The system displays the company's profile, including its business model, financials, leadership team, and other relevant details. 3. The Investor reads through the information provided.
Postconditions	- The Investor is informed about the company’s background and financial details.

Use Case 4: Invest in Companies (Investor)

Section	Description
Primary Actor	Investor
Goal	Select and invest in a company.
Preconditions	- The Investor must be logged into the system. - The Investor reads the company's information.
Main Success Scenario	1. The Investor selects the desired fundraising campaign and enters the amount they wish to invest. 2. The system displays the investment details for confirmation. 3. The Investor confirms the investment. 4. The system processes the investment, updating the Investor's account and the fundraising campaign details.
Postconditions	- The investment is recorded. - The fundraising campaign’s progress is updated.
Alternate Flows	- Investment Declined: If the investment cannot be processed for any reason, the system displays an error message and suggests retrying or contacting support.

Use Case 5: Track Investments (Investor)

Section	Description
Primary Actor	Investor
Goal	Track the history of investments.
Preconditions	- The Investor must be logged into the system. - The Investor must have made at least one investment.
Main Success Scenario	1. The Investor navigates to the investment history section. 2. The system displays a list of all previous investments, including details such as the company name, investment amount, date, and status. 3. The Investor reviews the investment history.
Postconditions	- The Investor has a clear record of their past investments.
Alternate Flows	- No Investments Found: If the Investor has not made any investments, the system displays a message indicating no history is available and suggests starting with an investment.

Use Case 6: View Portfolio (Investor)

Section	Description
Primary Actor	Investor
Goal	View the overall investment portfolio.
Preconditions	- The Investor must be logged into the system. - The Investor must have an active investment portfolio.
Main Success Scenario	1. The Investor navigates to the portfolio section. 2. The system displays an overview of the Investor's current investments, including total value and total value for each company. 3. The Investor reviews the portfolio.
Postconditions	- The Investor is informed about the current status of their investments.

Use Case 7: Access Admin Control Panel (Admin)

Section	Description
Primary Actor	Admin
Goal	Access the control panel to manage fundraising and users.
Preconditions	- The Admin must be authenticated with admin privileges.
Main Success Scenario	1. The Admin logs into the system with their credentials. 2. The Admin navigates to the admin control panel. 3. The system grants access to various management functions such as fundraising campaigns and user management.
Postconditions	- The Admin is successfully logged into the control panel with access to all admin features.
Alternate Flows	- Unauthorized Access: If the Admin does not have the necessary privileges, the system denies access and displays an error message.

Use Case 8: Manage Fundraising Campaign (Admin)

Section	Description
Primary Actor	Admin
Goal	Create, update, delete, and view fundraising campaigns.
Preconditions	- The Admin must be logged into the control panel.
Main Success Scenario	1. The Admin selects the option to create, update, delete, or view a fundraising campaign. 2. For creation or updates, the Admin enters or edits the necessary details (e.g., company information, funding goals, deadlines). 3. The system saves the changes and updates the fundraising campaign list. 4. If deleting, the Admin confirms the deletion, and the system removes the campaign.
Postconditions	- The fundraising campaign list is updated according to the Admin's actions.
Alternate Flows	- Deletion Confirmation: If the Admin attempts to delete a campaign, the system may prompt for confirmation to avoid accidental deletions.

Use Case 9: View Users (Admin)

action	Description
Primary Actor	Admin
Goal	View user' accounts in detail or information.
Preconditions	- The Admin must be logged into the control panel.
Main Success Scenario	1. The Admin navigates to the user section. 2. The Admin views the list of registered users.
Postconditions	- User accounts are traversed according to the Admin's actions.

Use Case 10: View Fundraising Statements (Admin)

Section	Description
Primary Actor	Admin
Goal	View detailed statements for each fundraising campaign.
Preconditions	- The Admin must be logged into the control panel.
Main Success Scenario	1. The Admin reach statements section. 2. The system displays a detailed statement, including user investments, timestamps, and amounts. 3. The Admin reviews the statement.
Postcondition	The Admin has access to detailed financial information for each fundraising campaign.
Alternat Flows	No Investments Recorded: If no investments have been made in the campaign, the system indicates that the statement is empty.

Use Case 11: Contact Admin (Company)

Section	Description
Primary Actor	Company
Goal	Contact the Admin to provide information and initiate a fundraising campaign.
Preconditions	- The Company must have access to the system or an external communication channel (e.g., email).
Main Success Scenario	1. The Company accesses the contact form or uses an external method to reach the Admin. 2. The Company provides the necessary details about their fundraising needs. 3. The Admin receives the information and initiates the fundraising process.
Postcondition	The Admin is informed of the Company's request and can begin the process of creating a fundraising campaign.

Scope of Services

1	**Database Design** ● Define database tables, relationships, and keys.
2	**Development Frontend and Backend**
3	**Deploy the web application**

Architecture Design

The Architecture Design Document presented herein offers a comprehensive and authoritative overview of the system architecture for our project. This document serves as a definitive guide, elucidating the underlying structure, relationships, and functionalities of the system for the client's benefit, while simultaneously delineating the development team's strategic approach to delivering a robust and scalable solution.

The document encompasses four critical design diagrams, each providing unique insights into the system's architecture:

1. **ER Diagram (Entity-Relationship Diagram):** The ER Diagram shows how data is organized and connected in the database. It highlights the main entities (like users, campaigns, and payments) and how they relate to each other, giving a clear picture of how the data flows.

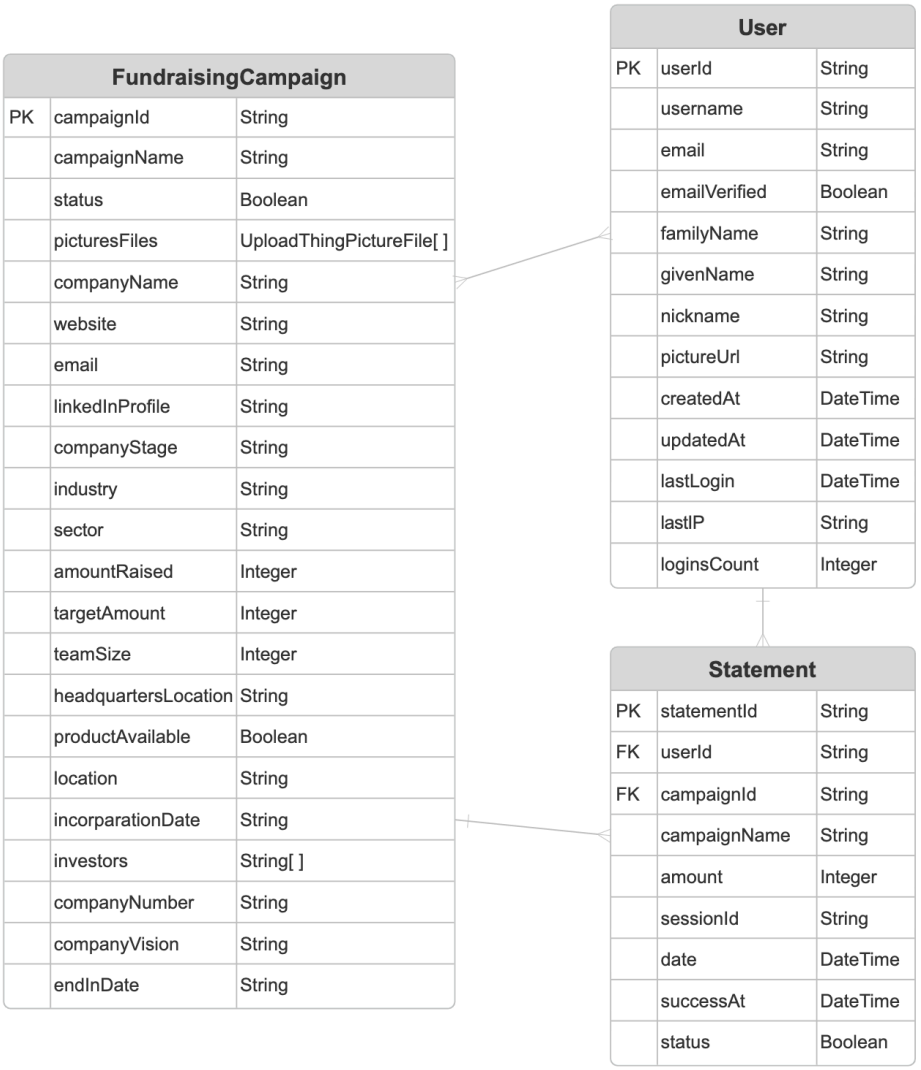
2. **Conceptual Diagram:** This diagram gives a high-level view of the system, showing the main components (like the user interface, backend, and database) and how they work together. It's a simple overview of how everything connects.

3. **UML Class Diagram:** The UML Class Diagram shows the structure of the system, including its main features, attributes, and how different parts of the code interact. It’s a helpful way to understand how the functionality is organized.

4. **Sequence Diagram:** The Sequence Diagram shows step-by-step interactions between different parts of the system for important tasks. It explains how processes happen, such as user actions or system responses, making it easy to follow the flow.

These meticulously crafted diagrams provide a structured and professional approach for reviewing the system's architecture, aligning precisely with the project's delivery plan. The objective is to ensure that the client gains a thorough and unambiguous understanding of each component, its purpose, and the manner in which it will be delivered to meet the project's objectives.

ER Diagram



relationship

- Users can create multiple Statements (One-to-Many).
- FundraisingCampaign can have multiple Statements (One-to-Many).
- Users can participate in multiple FundraisingCampaign, and FundraisingCampaign can have multiple Users participating (Many-to-Many).**Entities and Their Attributes**

FundraisingCampaign (Campaign)

Attribute	Type	Description
campaignId	String (PK)	Unique identifier for each campaign
campaignName	String	The name of the campaign
status	Boolean	The current status of the campaign
picturesFiles	UploadThingPictureFile[]	An array of picture files associated with the campaign
companyName	String	The name of the company running the campaign
website	String	The company's website URL
email	String	The company's contact email address
linkedInProfile	String	The company's LinkedIn profile URL
companyStage	String	The current stage of the company (e.g., startup, growth)
industry	String	The industry the company operates in
sector	String	The specific sector within the industry
amountRaised	Integer	The amount of funds raised in the campaign
targetAmount	Integer	The fundraising goal for the campaign
teamSize	Integer	The number of employees in the company
headquartersLocation	String	The location of the company's headquarters

productAvailable	Boolean	Indicates if the company's product is available
location	String	The location of the company
incorparationDate	String	The date when the company was incorporated
investors	String[]	An array of investor information
companyNumber	String	The company's registration number
companyVision	String	A description of the company's vision or mission
endInDate	String	The end date of the campaign

User

Attribute	Type	Description
userId	String (PK)	Unique identifier for each user
username	String	The user's chosen username
email	String	The user's email address
emailVerified	Boolean	Indicates if the user's email has been verified
familyName	String	The user's family name
givenName	String	The user's given name
nickname	String	The user's nickname
pictureUrl	String	URL to the user's profile picture
createdAt	DateTime	The date and time when the user account was created
updatedAt	DateTime	The date and time when the user account was last updated
lastLogin	DateTime	The date and time of the user's last login
lastIP	String	The IP address of the user's last login
loginsCount	Integer	The number of times the user has logged in

Statement

Attribute	Type	Description
statementId	String (PK)	Unique identifier for each statement
userId	String (FK)	Foreign key linking to the User table
campaignId	String (FK)	Foreign key linking to the FundraisingCampaign table
campaignName	String	The name of the campaign associated with this statement
amount	Integer	The amount invested or pledged in this statement
sessionId	String	A session identifier for this statement
date	DateTime	The date and time when the statement was created
successAt	DateTime	The date and time when the statement was successfully processed
status	Boolean	The current status of the statement

Notes

- The FundraisingCampaign entity combines attributes that might typically be split between a Company and a Campaign entity. This design choice may offer simplicity but could lead to data redundancy if a company runs multiple campaigns.

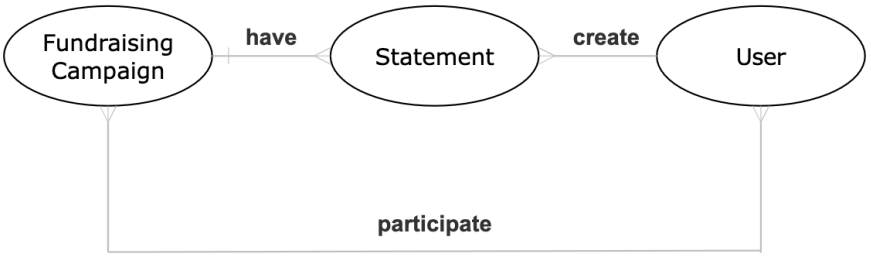
- The User entity includes detailed information about user activity and authentication, which is useful for tracking user engagement and security purposes.

- The Statement entity serves as a junction between Users and FundraisingCampaigns, representing individual investments or pledges. This allows for a many-to-many relationship between users and campaigns.

- Some data types (like UploadThingPictureFile[]) suggest that this schema might be designed for a specific application or framework.

- The use of String types for some fields that might typically be dates (like incorporationDate and endInDate in FundraisingCampaign) could indicate flexibility in date formats or the need for string manipulation before storage or retrieval.

Conceptual Diagram



Overview

This conceptual diagram represents the high-level structure of a fundraising platform's database, focusing on the Many-to-Many relationship between Users and FundraisingCampaigns, mediated by Statements.

Detailed Conceptual Diagram

This diagram illustrates that:

- A FundraisingCampaign can have many Statements
- A User can have many Statements
- A Statement always belongs to one FundraisingCampaign and one User
- The Many-to-Many relationship between FundraisingCampaign and User is realized through the Statement entity

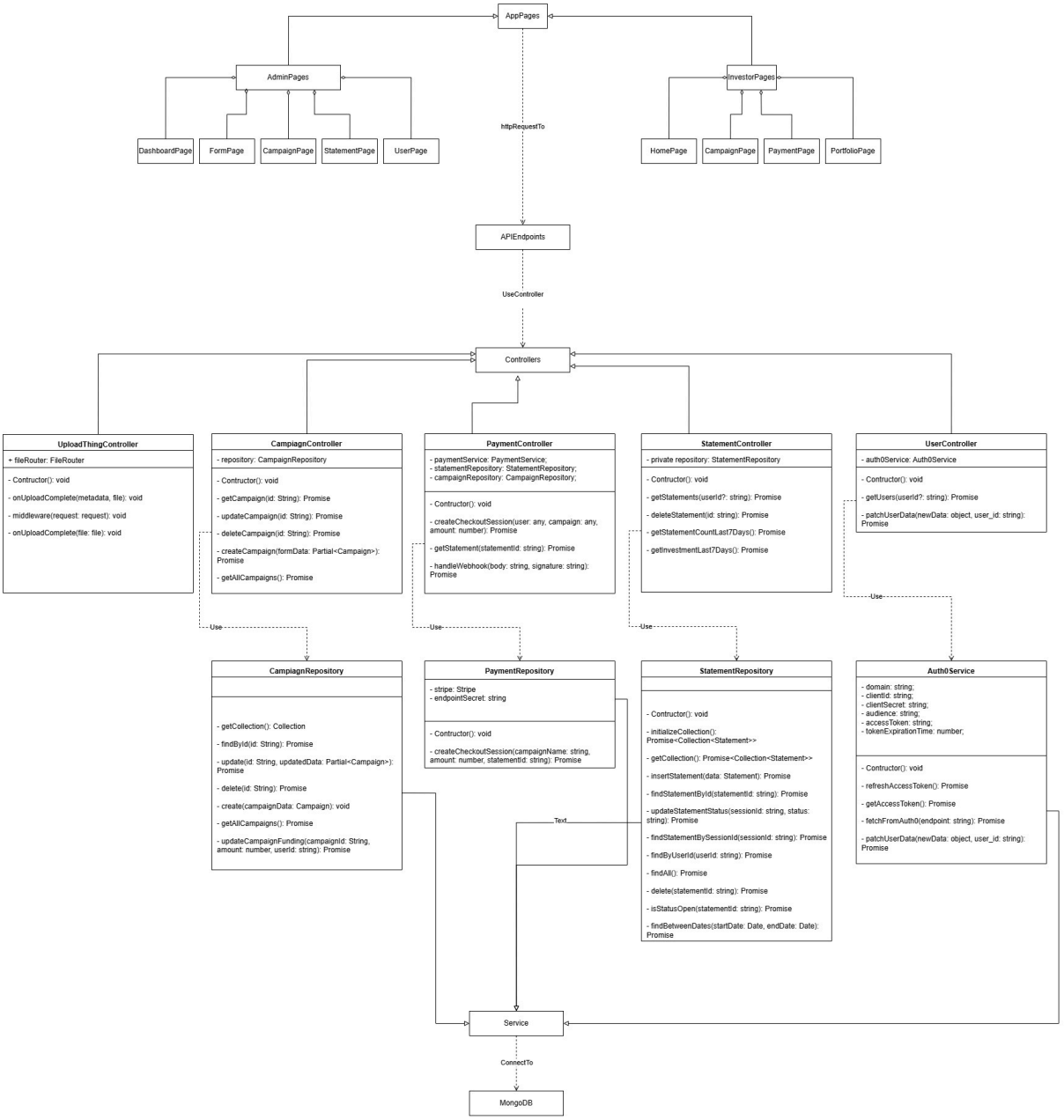
Entity Descriptions

- **FundraisingCampaign:** Represents a fundraising initiative, combining both campaign and company information.

- **User:** Represents individuals interacting with the platform, such as investors or campaign creators.

- **Statement:** Acts as a bridge between Users and FundraisingCampaigns, representing individual investments or pledges.

UML Class Diagram



AppPages:

- AdminPage: Manages admin pages (Dashboard, Form, Campaign, Statement, User).
- InvestorPage: Manages investor pages (Home, Campaign, Payment, Portfolio).

Controllers: Handle API requests and interact with services and repositories:

- UploadThingController: Manages file uploads.
- CampaignController: Manages campaigns (create, update, retrieve, delete).
- PaymentController: Manages payments and sessions.
- StatementController: Handles statements.
- UserController: Manages user data.

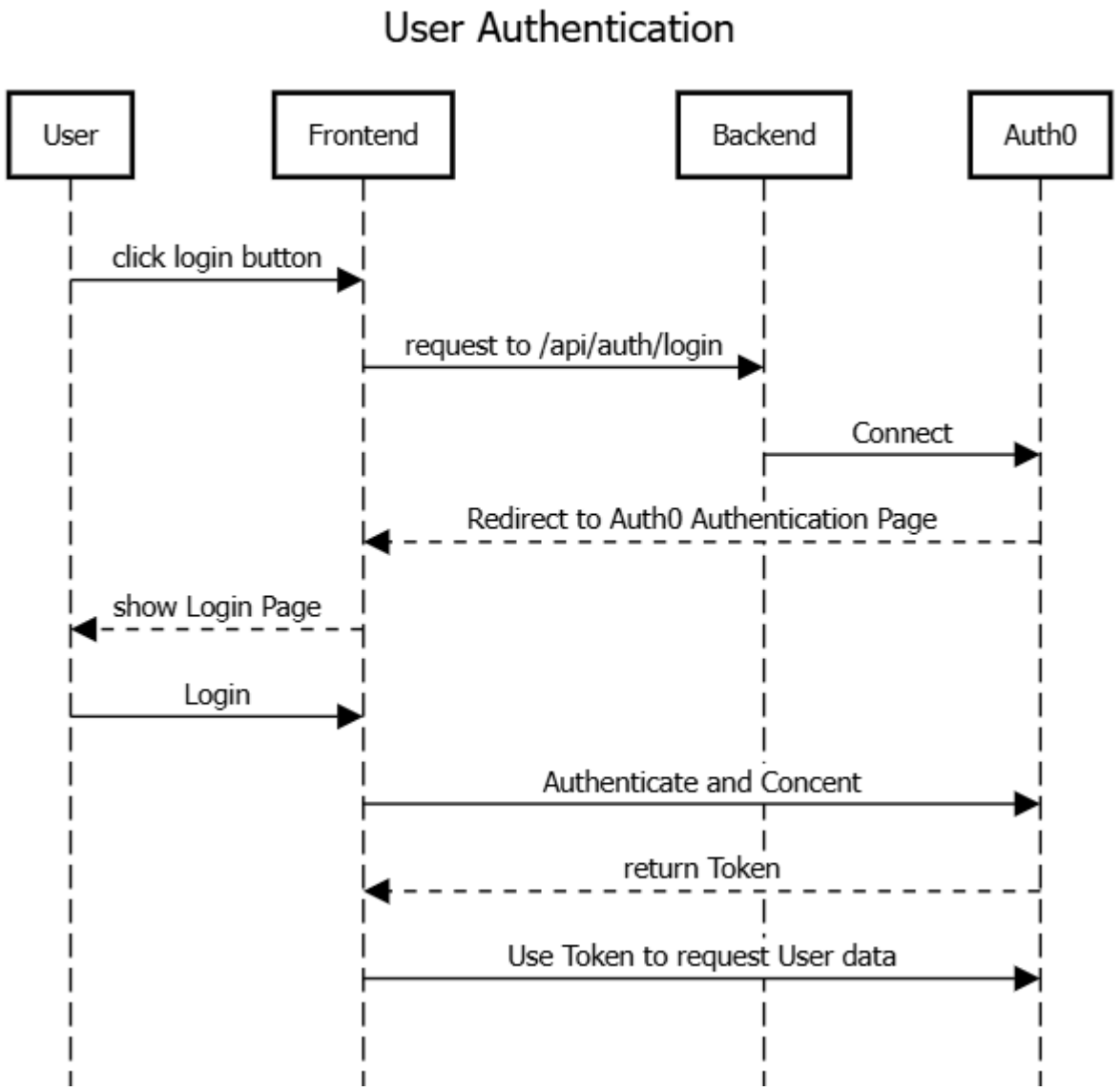
Repositories: Abstract data interactions:

- CampaignRepository: Handles campaign data.
- PaymentRepository: Manages payment data.
- StatementRepository: Manages statement data.
- Auth0 Service: Handles authentication.

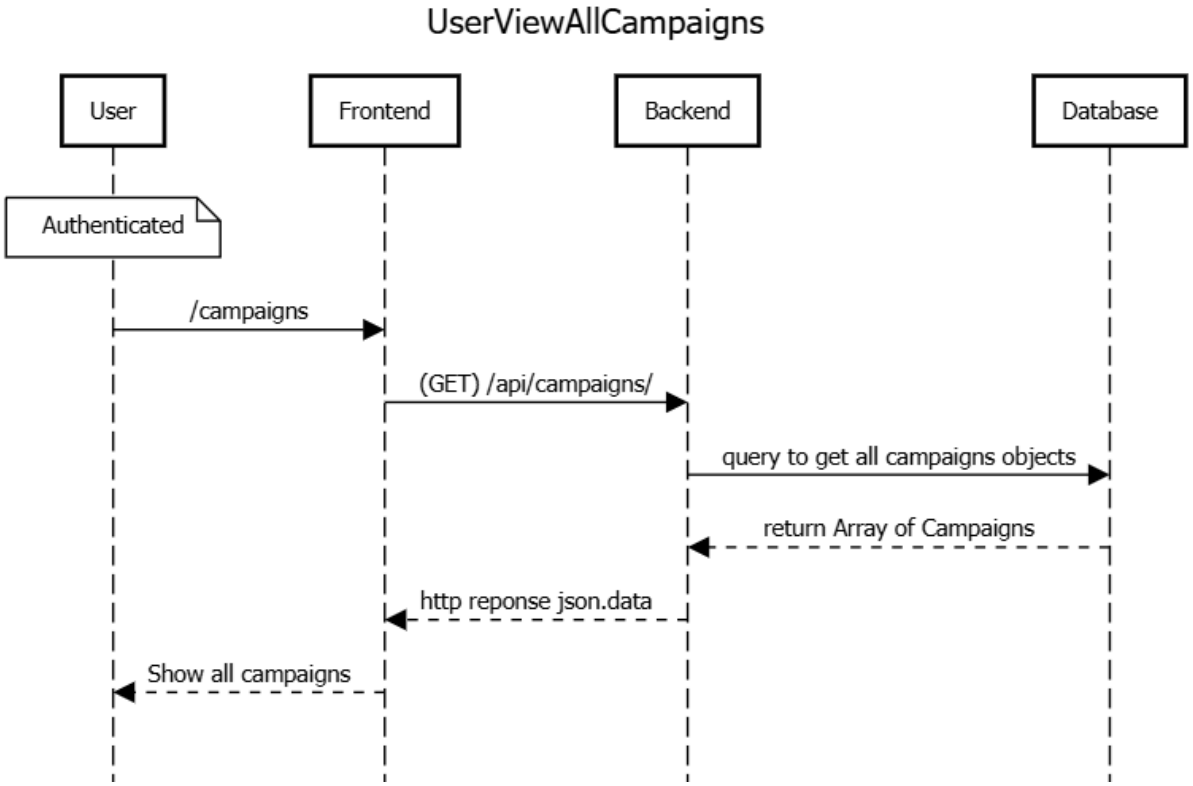
Service Layer: Contains business logic between controllers and repositories.
MongoDB: Data storage used by repositories.

Sequence Diagram

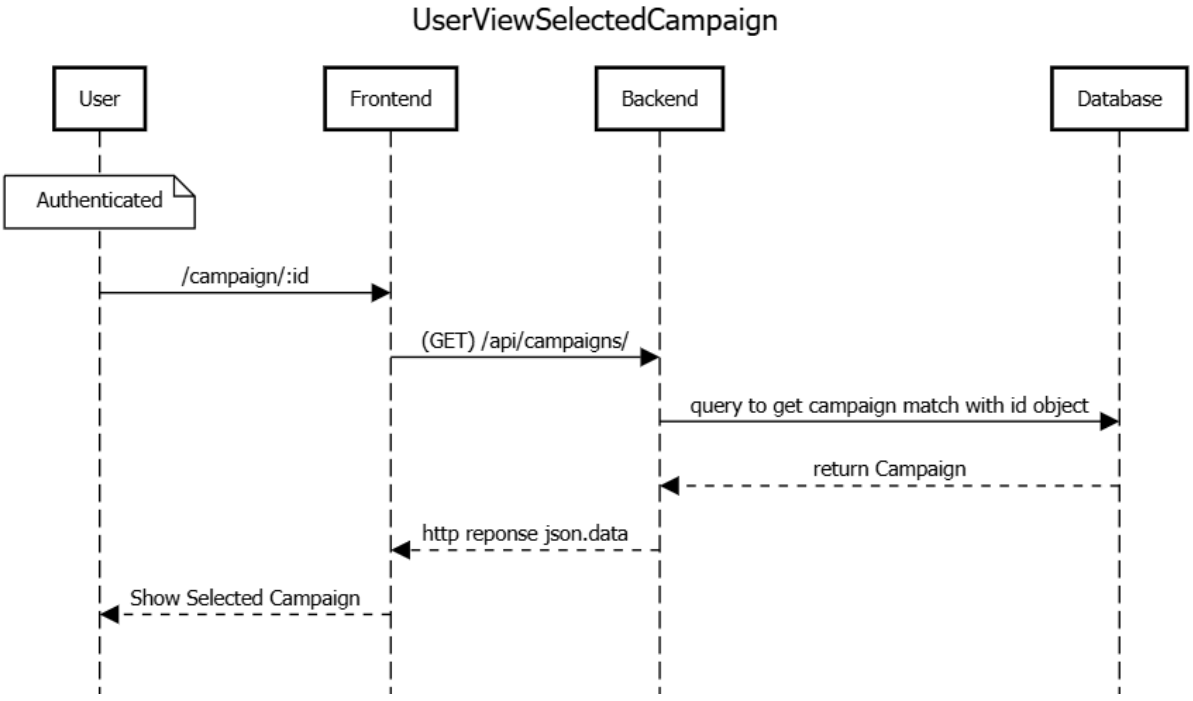
- **Authentication**



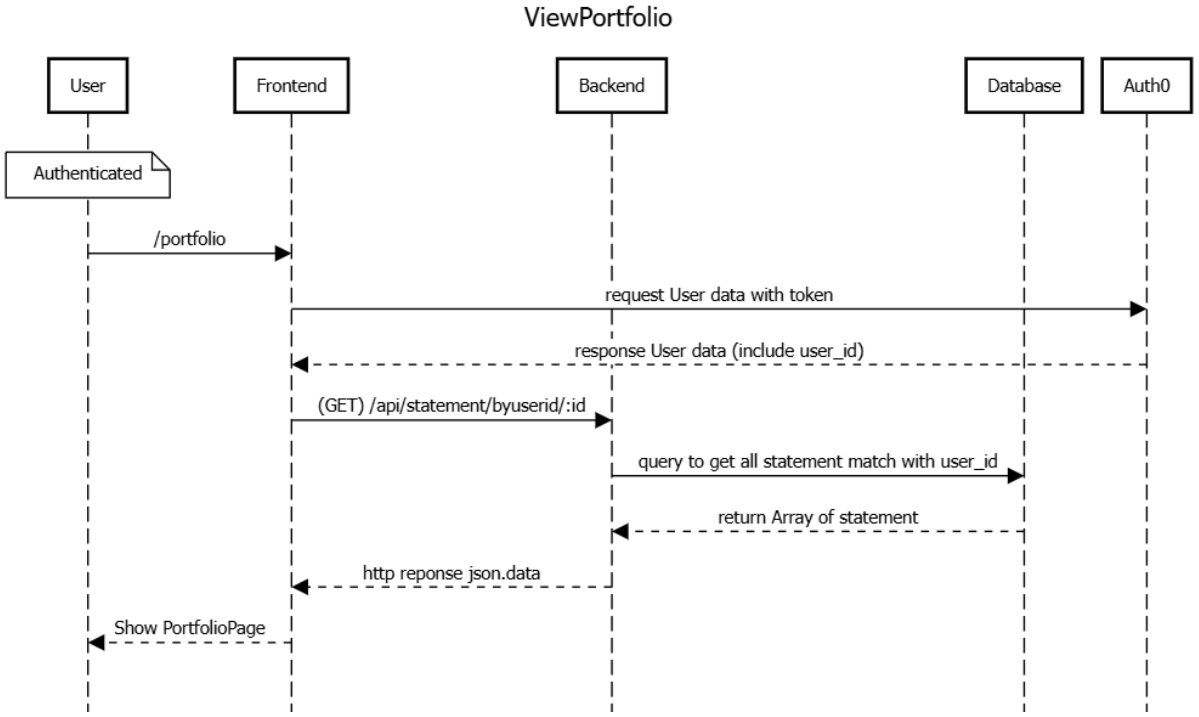
- **View Campaigns Pages**



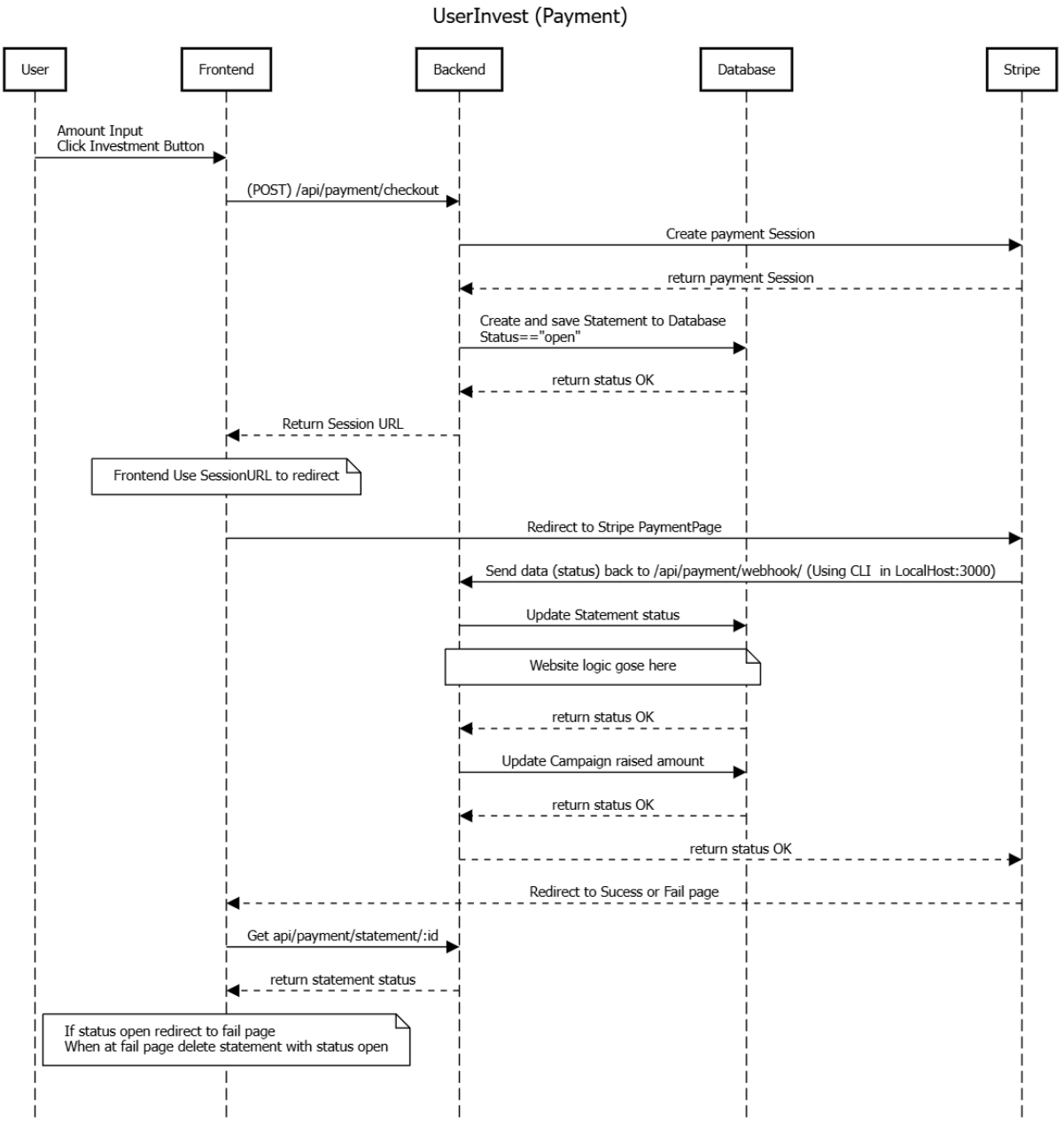
- **ViewSelectedCampaign**



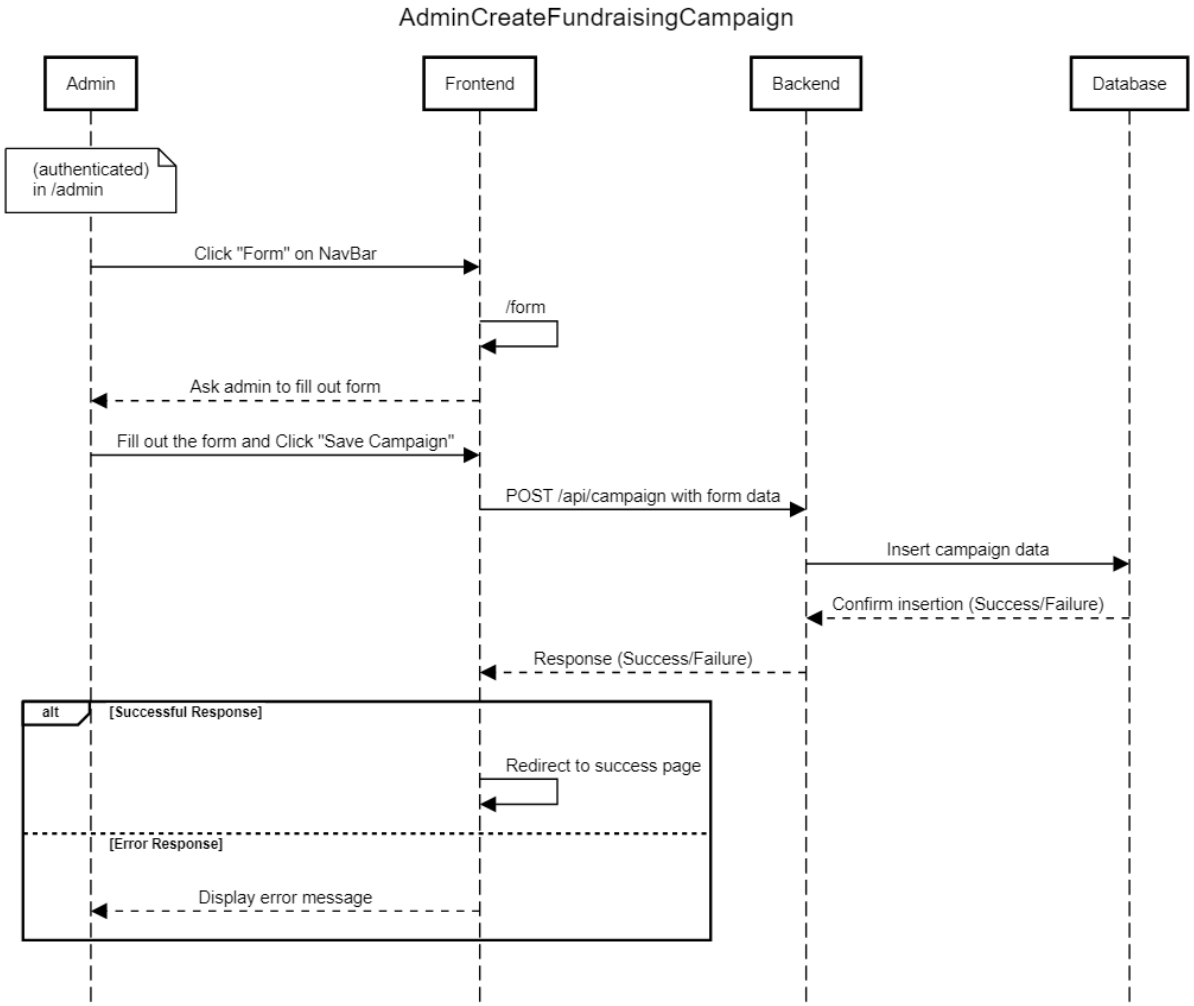
- **ViewPortfolio**



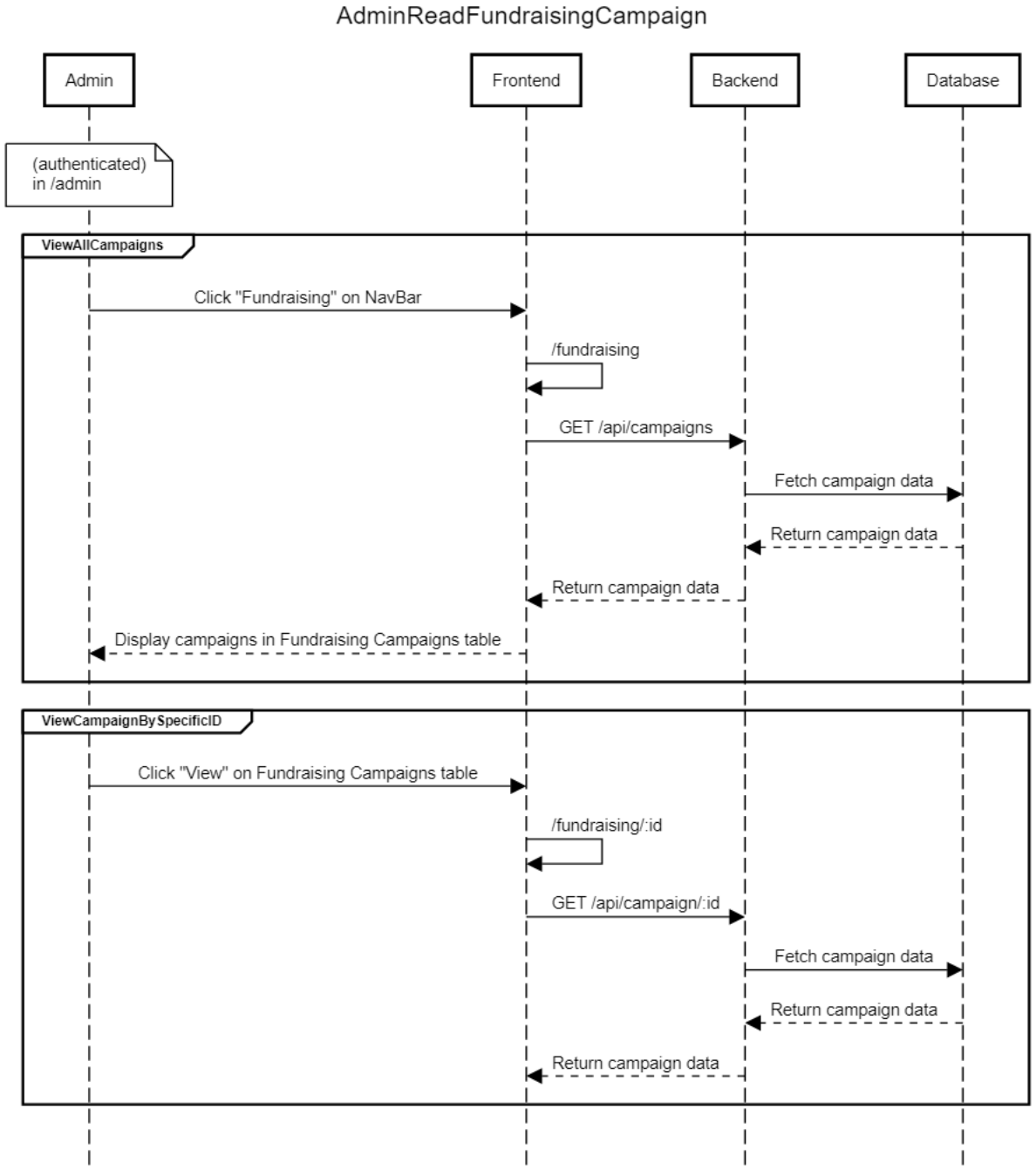
- **Payment**


UserInvest (Payment)
User
Frontend
Backend
Database
Stripe
Amount Input
Click Investment Button
(POST) /api/payment/checkout
Create payment Session
return payment Session
Create and save Statement to Database
Status=="open"
return status OK
Return Session URL
Frontend Use SessionURL to redirect
Redirect to Stripe PaymentPage
Send data (status) back to /api/payment/webhook/ (Using CLI in LocalHost:3000)
Update Statement status
Website logic gose here
return status OK
Update Campaign raised amount
return status OK
return status OK
Redirect to Sucess or Fail page
Get api/payment/statement/:id
return statement status
If status open redirect to fail page
When at fail page delete statement with status open

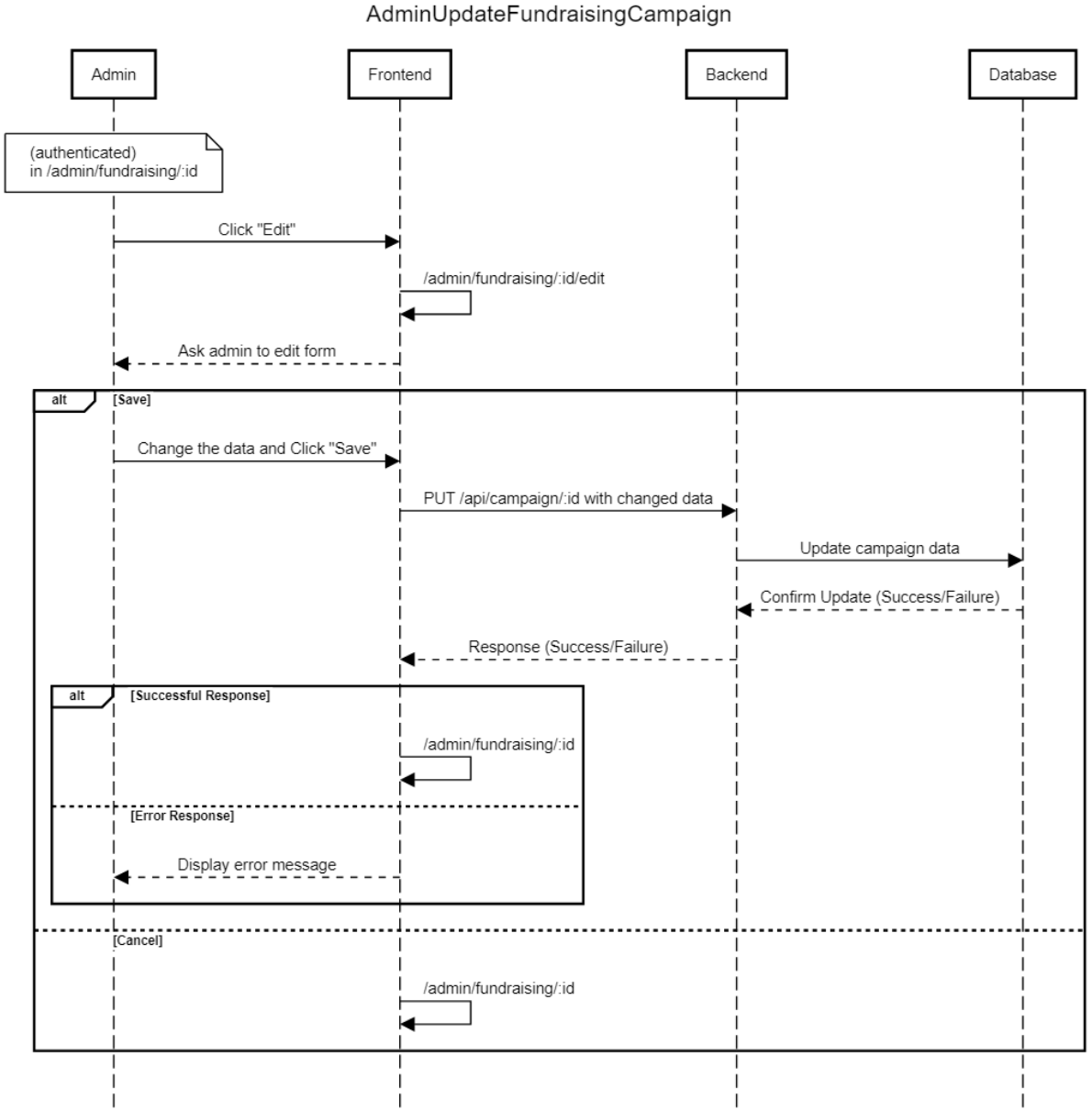
- **AdminCreateFundraisingCampaign**



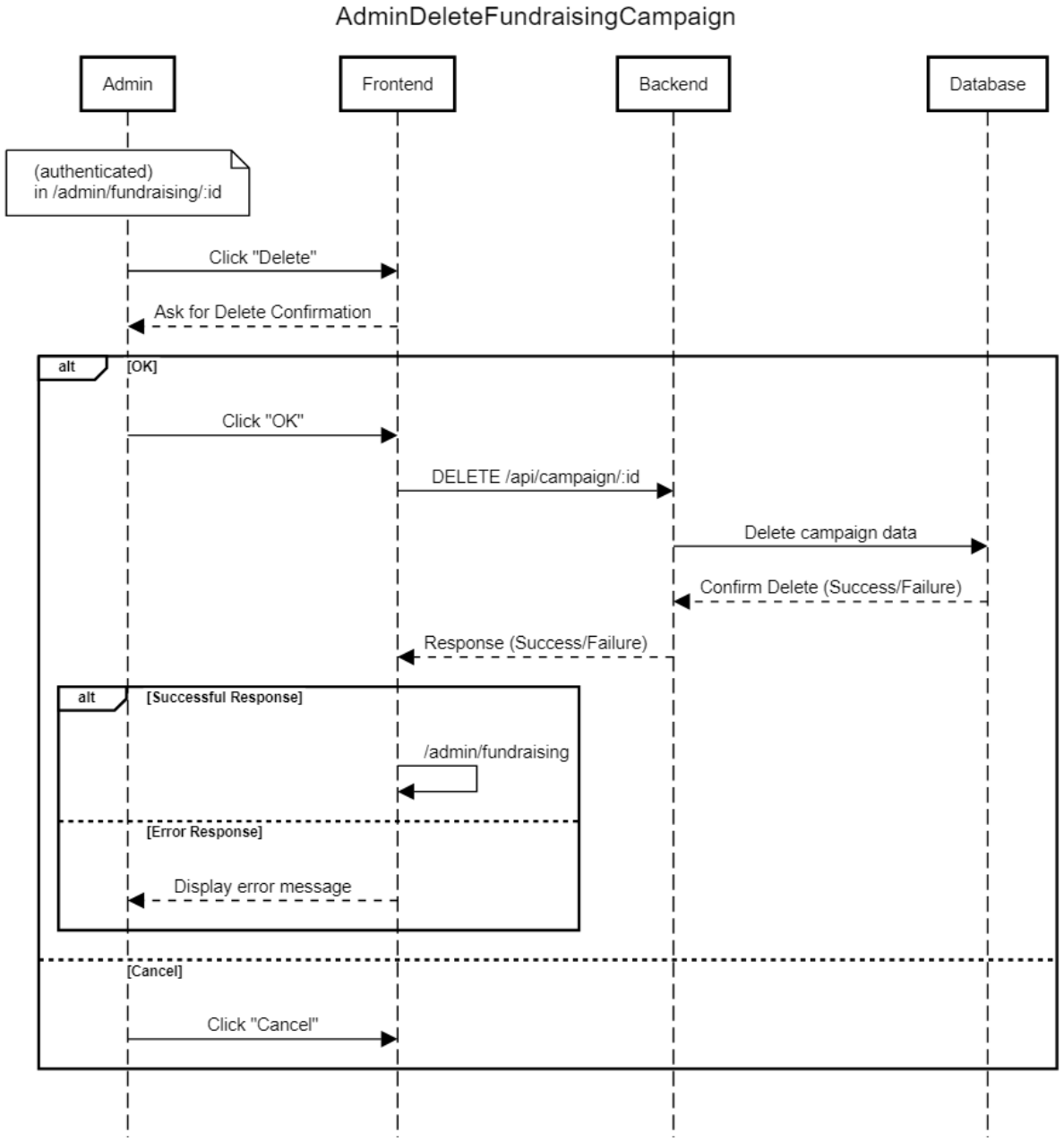
- **AdminReadFundraisingCampaign**



- **AdminUpdateFundraisingCampaign**



- **AdminDeleteFundraisingCampaign**



API Documentation

Overview

This document provides detailed information on all API endpoints in the application, including Campaign, Statement, User, Authentication, Payment, and Upload APIs. Each endpoint description includes the request method, URL, request parameters, expected responses, and sample response data.

1Campaign API Group

1.1 Get All Campaigns

- Endpoint: `../api/campaigns`
- Method: `GET`
- Description: Retrieve a list of all campaign data.
- Response:
  - Status: 200 OK
  - Body: Array of Campaign objects

Sample Response:

```
[
   {
      "_id": "ObjectId",
      "id": "String",
      "name": "string",
      "status": "string",
      "description": "string",
      "pictureFiles": ["UploadThingPictureFile"],
      "companyName": "string",
      "website": "string",
      "founderName": "string",
      "email": "string",
      "linkedInProfile": "string",
      "companyStage": "string",
      "industry": "string",
      "sector": "string",
      "amountRaised": 10000,
      "targetAmount": 50000,
      "teamSize": 10,
      "headquartersLocation": "string",
      "productAvailable": true,
      "location": "string",
      "incorporationDate": "string",
      "investors": ["string"],
      "companyNumber": "string",
      "companyVision": "string",
      "endInDate": "string"
   }
]
```

1.2 Create Campaign

- Endpoint: `../api/campaign`
- Method: `POST`
- Description: Create a new campaign.
- Request Body: JSON object containing campaign details
- Response:
  - Status: 201 Created
  - Message: Success message if the campaign is created successfully

1.3 Get Campaign by ID

- Endpoint: `../api/campaign/<id>`
- Method: `GET`
- Description: Retrieve campaign details by specific ID.
- Response:
  - Status: 200 OK
  - Body: Campaign object

Sample Response:

```
{
   "_id": "ObjectId",
   "id": "String",
   "name": "string",
   "status": "string",
   "description": "string",
   "pictureFiles": ["UploadThingPictureFile"],
   "companyName": "string",
   ...
}
```

1.4 Update Campaign

- Endpoint: `../api/campaign/<id>`
- Method: `PUT`
- Description: Update a campaign by specific ID.
- Request Body: JSON object with updated campaign details
- Response:
  - Status: 200 OK
  - Message: Success message if updated successfully

1.5 Delete Campaign

- Endpoint: `../api/campaign/<id>`

- Method: `DELETE`
- Description: Delete a campaign by specific ID.
- Response:
  - Status: 200 OK
  - Message: Success message if deleted successfully

---

2. Statement API Group

2.1 Get All Statements

- Endpoint: `../api/statements`
- Method: `GET`
- Description: Retrieve all statement data.
- Response:
  - Status: 200 OK
  - Body: Array of Statement objects

Sample Response:


```
[
   {
      "_id": "ObjectId",
      "statement_id": "string",
      "user_id": "string",
      "campaign_id": "string",
      "campaignName": "string",
      "amount": 1000,
      "session_id": "string",
      "date": "string",
      "successAt": "string",
      "status": "string"
   }
]
```

2.2 Get Statement by ID

- Endpoint: `../api/payment/statement/<id>`
- Method: `GET`
- Description: Retrieve a statement by specific ID.
- Response:
    - Status: 200 OK
    - Body: Statement object

2.3 Delete Statement

- Endpoint: `../api/statement/<id>`
- Method: `DELETE`
- Description: Delete a statement by specific ID.
- Response:
    - Status: 200 OK
    - Message: Success message if deleted successfully

2.4 Get Statements by User ID

- Endpoint: `../api/statement/byuserid/<id>`
- Method: `GET`
- Description: Retrieve statements by specific user ID.
- Response:
    - Status: 200 OK
    - Body: Statement object

---

3. User API Group

3.1 Get All Users

- Endpoint: `../api/users`
- Method: `GET`
- Description: Retrieve all user data.
- Response:
    - Status: 200 OK
    - Body: Array of User objects

Sample Response:

```
[
   {
      "user_id": "string",
      "name": "string",
      "email": "string",
      "email_verified": true,
      "family_name": "string",
      "given_name": "string",
      "nickname": "string",
      "picture": "string",
      "created_at": "string",
      "updated_at": "string",
      "last_login": "string",
      "last_ip": "string",
      "logins_count": 10
   }
]
```


3.2 Get User by ID

- Endpoint: `../api/user/<id>`
- Method: `GET`
- Description: Retrieve a user by specific ID.
- Response:
  - Status: 200 OK
  - Body: User object

3.3 Update User Data

- Endpoint: `../api/user/patch`
- Method: `POST`
- Description: Update user data.
- Request Body: JSON object with updated user details
- Response:
  - Status: 200 OK
  - Message: Success message if updated successfully

4. Authentication API Group

4.1 Login

- Endpoint: `../api/auth/login`
- Method: `GET`
- Description: Redirects to login page.

4.2 Logout

- Endpoint: `../api/auth/logout`
- Method: `GET`
- Description: Redirects to logout page.

5. Payment API Group

5.1 Payment Checkout

- Endpoint: `../api/payment/checkout`
- Method: `POST`
- Description: Initiates payment checkout process.
- Response:
  - Status: 302 Redirect
  - Message: Redirects to the payment checkout page

5.2 Payment Webhook

- Endpoint: `../api/payment/webhook`
- Method: `POST`
- Description: Endpoint for Stripe to confirm session updates.
- Response:
  - Status: 200 OK
  - Message: Confirmation of the webhook received

6. Uploadthing API Group

6.1 Upload Pictures

- Endpoint: `../api/uploadthing`
- Method: `POST`
- Description: Uploads pictures to the cloud.
- Request Body: Form-data with image files
- Response:
  - Status: 201 Created
  - Message: Success message if uploaded successfully

Deployment and Handover

Overview:

This document outlines the deployment process and configuration for the **B2DVenture** web application on Vercel, using MongoDB Atlas for the database, Stripe for payment processing, and Auth0 for user authentication. Additionally, it provides instructions for transferring access control to the client so they can manage and maintain the application without redeployment.

Components:

- **Frontend/Backend: NextJs Deployed on Vercel.**
- **Database: MongoDB Atlas (Cloud).**
- **Payments: Stripe.**
- **Authentication: Auth0.**

Vercel Environment Setup:

AUTH0 Variables
1. **AUTH0_SECRET:** A secret key used to sign and encrypt JWT tokens issued by Auth0. It is necessary for secure communication between your application and Auth0. 2. **AUTH0_BASE_URL:** The base URL for your application. It is used to define where Auth0 should redirect after login or logout, typically the frontend application URL (e.g., http://localhost:3000 for local development). 3. **AUTH0_ISSUER_BASE_URL:** The base URL for your Auth0 domain, used for generating authorization and token requests. It's typically in the format https://<your-domain>.auth0.com. 4. **AUTH0_CLIENT_ID:** The Client ID provided by Auth0 to uniquely identify your application. It's used when making authentication requests.

5. **AUTH0_CLIENT_SECRET:** The Client Secret provided by Auth0, used for secure communication between your application and Auth0, especially when requesting tokens.

6. **AUTH0_AUDIENCE:** The identifier of the Auth0 API or service your application is requesting access to. For example, https://dev-juzu8hcucbv4naz4.us.auth0.com/api/v2/ for the Auth0 Management API.

7. **AUTH0_SESSION_AUTO_SAVE:** Determines whether the session data should automatically be saved. A boolean value (true or false).

8. **AUTH0_MANAGEMENT_CLIENT_ID:** The Client ID for the Auth0 Management API. Used for making API calls to manage users and other resources in Auth0.

9. **AUTH0_MANAGEMENT_CLIENT_SECRET:** The Client Secret for the Auth0 Management API, used in conjunction with the management client ID for making secure API calls.

MongoDB Variables

1. **MONGODB_URL:** The connection string used to connect to your MongoDB Atlas database. It includes authentication credentials (username and password) and the database cluster URL.

Stripe Variables

1. **STRIPE_SECRET_KEY:** The secret key for your Stripe account, used to interact with Stripe's API securely. This is required for server-side operations like charging customers or managing subscriptions.
2. **NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY:** The publishable key for your Stripe account, used on the client side to securely interact with Stripe's API without exposing sensitive information.

3. **STRIPE_WEBHOOK_SECRET:** The secret used to verify that webhook events received from Stripe are legitimate and originated from your Stripe account. This is important for security when handling events such as payment success or failure.

Next.js Variables

1. **NEXT_PUBLIC_BASE_URL:** The base URL of your Next.js application. This is used throughout your app to make API requests or for redirecting to the correct page.

UPLOADTHING Variables

1. **UPLOADTHING_TOKEN:** The token used to authenticate requests to the Uploadthing service, allowing the upload and management of files.

Coinbase Variables

1. **COINBASE_COMMERCE_API_KEY:** The API key used for interacting with the Coinbase Commerce API to accept cryptocurrency payments.
2. **COINBASE_COMMERCE_WEBHOOK_SECRET:** The secret used to verify that incoming webhook notifications from Coinbase Commerce are valid and came from Coinbase.

Web application Deployment on Vercel

- **Deployment Steps:**
  - Connect the repository to Vercel.
  - Add required environment variables in Project Settings > Environment Variables on Vercel.
  - Deploy using the Vercel dashboard or CLI command `vercel --prod`.
- **Build Settings:**
  - Build Command: `next build`.
  - Output Directory: .next.

Database Setup with MongoDB Atlas

- **Database Connection:**
  - Create a MongoDB Atlas cluster and configure IP whitelisting.
  - Add MONGODB_URI to Vercel.
- **Access Management for Client:**
  - Grant access to the client by adding them as a user in MongoDB Atlas with required permissions.
  - Alternatively, transfer the MongoDB project to the client's account for full control.

Payment Integration with Stripe

- **Stripe Configuration:**
  - Generate API keys in Stripe Dashboard and add them as environment variables on Vercel.
- **Access Management for Client:**
  - Add the client as a Team Member in the Stripe Dashboard with appropriate permissions.
  - For full control, guide the client in setting up their own Stripe account and transfer settings accordingly.

Authentication Setup with Auth0

- **Auth0 Application Setup:**
  - Set up an application in Auth0 Dashboard with required redirect URIs and environment variables.
- **Access Management for Client:**
  - Add the client as an admin in the Auth0 tenant for management permissions.
  - For full control, transfer the Auth0 tenant or set up a new tenant under the client's account.

Post-Deployment Testing

- **Confirm that all features are functional and client has access to key components:**
  - **Database Access with MongoDB Atlas.**
  - **User Authentication with Auth0.**
  - **Payment Processing with Stripe.**

Rollback Procedures

- **Vercel Rollback:**
  - Roll back to previous versions on Vercel if issues arise.
- **Stripe and Auth0 Rollback:**
  - Stripe transaction management via the client's access to Stripe Dashboard.
  - Auth0 user management via the client's admin permissions in Auth0.

Project Handover

This section provides instructions for transferring control of the deployment, related tools, and the codebase to the client.

1. GitHub Repository Handover

- **Repository Access:**
  - **Add Client as Collaborator:** Add the client as a collaborator to the GitHub repository, so they can access the codebase and make necessary changes. This can be done in the Settings > Collaborators section of the repository.
  - **Transfer Ownership:** If you want to transfer full control of the repository, you can transfer the repository ownership to the client's GitHub account. To do this:
    - Go to the repository settings.
    - Scroll down to the Danger Zone and click Transfer.
    - Enter the client's GitHub username and confirm the transfer.
    - The client will then have full ownership of the repository and be able to manage all settings, collaborators, and code deployments.

2. Vercel Handover

- Add the client as a Collaborator in the Vercel project to allow them to monitor and manage deployments.
- To fully transfer control, migrate the project to the client's Vercel account by transferring ownership in Team Settings.

3. MongoDB Atlas Handover

- **User Access:** Add the client as a user in MongoDB Atlas with access to the database.
- **Project Transfer:** If needed, transfer the MongoDB project to the client's MongoDB Atlas account.

4. Stripe Handover

- **Team Access:** Add the client to the Stripe Dashboard as a Team Member with permissions to manage transactions and settings.
- **Full Transfer:** Guide the client through creating a new Stripe account and provide integration details.

5. Auth0 Handover

- Admin Access: Add the client as an admin in the Auth0 Dashboard.
- **Tenant Transfer:** If full control is required, set up a new tenant under the client's Auth0 account and migrate settings.

6. Coinbase Handover

- **Team Access:** Add the client as a team member in the Coinbase Dashboard with appropriate permissions to manage the integration, view transaction history, and configure settings.
- **Full Transfer:** If full control is required, guide the client through creating a new Coinbase account, and provide them with API keys and integration details for migrating the payment processing setup.