**Valid Treasure For Kunyo Gaming**

**BY**

**Prabhat Baduwal**

**7-2-422-11-2021**

**13114/21**

KIST COLLEGE

A Project Report Submitted to

**Faculty of Management, Tribhuvan University**

in partial fulfillment of the requirements for the degree of

**Bachelor of Information Management (BIM)**

May/2025

**STUDENT DECLARATION**

This is to certify that I have completed the Project entitled “Valid Treasure” under the guidance of “Mr. Sandesh Bohora” in partial fulfillment of the requirements for the degree of **Bachelor of Information Management** at Faculty of Management, Tribhuvan University. This is my original work and I have not submitted it earlier elsewhere.

Date: Signature:

Name:

II

**CERTIFICATE FROM THE SUPERVISOR**

This is to certify that the project entitled “Valid Treasure” is an academic work done by “Prabhat Baduwal” submitted in the partial fulfillment of the requirements for the degree of **Bachelor of Information Management** at Faculty of Management, Tribhuvan University under my guidance and supervision. To the best of my knowledge, the information presented by him/her in the project report has not been submitted earlier.

Signature of the Supervisor

Name: Sandesh Bohora

Designation Date:

III

**Acknowledgements**

I would like to express my deepest gratitude to the following people whose support and guidance made this project possible:

* **Mr. Nabin Adhikari**, my project supervisor, for his invaluable advice, continuous encouragement, and constructive feedback throughout every phase of development and overseeing my progress and providing essential administrative support.
* **Faculty members of the Department of BIM**, for their insightful lectures and for laying the foundations that empowered this work.
* **My family and friends**, for their unwavering encouragement, patience, and understanding during late nights and crunch times.

V

**Abstract**

**Purpose:**

This project presents the design and implementation of a full-stack web application built using React for the frontend, Django for the backend and SQLite for the database with seamless integration of interactive web experience using React Three Fiber.

**Methodology:**

Following an Agile development methodology, the system was iteratively developed in different phases. Key activities included:

* **Requirement analysis:**  Gathering functional requirements (user registration, pages system implementation, admin, chats) and non-functional requirements (security, performance, accessibility).
* **Design**: Crafting a component-based React UI using TailwindCSS and Shadcn, designing SQLite schemas using Django’s ORM, and defining RESTful APIs as well.
* **Implementation**: Developing authentication (JWT-based), page management, user roles, and integrating 3d models within the application to make it interactive.
* **Testing**: Unit tests for backend endpoints, integration tests for user work flows, and user-acceptance testing in real browsers and devices.

**Conclusion:**

The final system supports user signup/login, pages browsing, chat operations, validity of users, page likes and favourites. All major modules passed their respective test suites without errors. This project delivers a suitable platform tailored to the Nepalese market, showcasing best practices in web development and real-world authentication systems.

VI

**Table of contents**

Table of Contents

[Introduction (Chapter I) 13](#_Toc196810964)

[1.1 Background 13](#_Toc196810965)

[1.2 Problem Statement 15](#_Toc196810966)

[1.3 Objectives 16](#_Toc196810967)

[1.4 Review of Related Work and Literature 17](#_Toc196810968)

[1.5 Development Methodology 18](#_Toc196810969)

[1.6 Scope and Limitations 19](#_Toc196810970)

[1.7 Report Organization 20](#_Toc196810971)

[Chapter II: System Development Process 21](#_Toc196810972)

[2.1 Analysis 21](#_Toc196810973)

[2.1.1 Requirement Analysis 21](#_Toc196810974)

[2.1.2 Feasibility Study 22](#_Toc196810975)

[2.1.3 Modeling 23](#_Toc196810976)

[2.2 Design 25](#_Toc196810977)

[2.2.1 User Interface Design 25](#_Toc196810978)

[2.2.2 Database Design 27](#_Toc196810979)

[2.2.3 System Architecture 29](#_Toc196810980)

[2.3 Implementation 30](#_Toc196810981)

[2.3.1 Frontend Implementation 30](#_Toc196810982)

[2.3.2 Backend Implementation 31](#_Toc196810983)

[2.3.3 Database Implementation 31](#_Toc196810984)

[2.3.4 Payment Gateway Integration 32](#_Toc196810985)

[2.3.5 Testing 32](#_Toc196810986)

[Conclusion and Recommendation (Chapter III) 33](#_Toc196810987)

[3.1 Summary 33](#_Toc196810988)

[3.2 Conclusion 33](#_Toc196810989)

[3.3 Recommendations 34](#_Toc196810990)

[References (Chapter IV) 34](#_Toc196810991)

VII

[Appendix (Chapter V) 35](#_Toc196810992)

[Appendix A : Code Snippets 35](#_Toc196810993)

[A.1 - POST /api/validTreasure/pages / (Page Post) 35](#_Toc196810994)

[A.2 - GET /api/validTreasure/products/ (Page List Retrieval) 38](#_Toc196810995)

[A.3 - Schema Definition of User model 40](#_Toc196810996)

[Appendix B : System Context Diagrams and Component tree Diagram 43](#_Toc196811002)

[D.1 System Context Diagram 43](#_Toc196811003)

[D.2 Component Tree Diagram 46](#_Toc196811004)

VIII

**List of Figures**

**List of Abbreviation**

This list includes the full form of short text.

**Abbreviation table:**

|  |  |  |
| --- | --- | --- |
| **S.no.** | **Abbreviation** | **Full Forms** |
| 1 | API | Application Program Interface |
| 2 | BIM | Bachelors of Information Management |
| 3 | DB | Database |
| 4 | DRF | Django Rest Framework |
| 5 | JWT | Json Web Token |
| 6 | SDK | Software Development Kit |
| 7 | UI | User Interface |

# Introduction (Chapter I)

## 1.1 Background

In this day and age where gamers and internet users have grown to a number that cannot be comprehended. This web application provides a platform where people can buy niche products and communicate with the buyers. Products that are mostly related to gaming and such.

Valid Treasure is a platform where people can search for pages which are created by the buyers (a type of user role). The users (buyers in this case) can view the pages and see what products the sellers are offering. The users can like the page and favorite them to maybe visit them quick in the near future. The users (buyers and sellers) are able to chat in a chat room to have a quick overview and product description directly from the seller.

From the administrative, Valid Treasure provides an easy way of use to monitor and view issues reported by users. Since it is a small app built around a specific market, the admins can thoroughly check the issues presented by the users and solve them. One of the important roles of admins is the ban pages that could have fraudulent intent. The app provides experience that the users want which is to be secure and worry less.

The development of Valid Treasure aims to bridge the gap between scammed buyers and sellers who are doubted easily by offering a modern, and secure platform tailored to local needs, leveraging the development stack of React And Django for dynamic and scalable performance. This allows the app to be built using robust system which requires minimal intervention in the sense of scalability and reusability.

## 1.2 Problem Statement

Despite the growing popularity of gaming, niche products are hardly available in pages and we can’t find dedicated web apps for this kind of products. Facing the challenges of providing a streamlined, reliable, and user-friendly online shopping experience. Many traditional games and such lack digital platforms, forcing customers to physically visit stores or use shady ways to buy such products, limiting accessibility and convenience. Additionally, where such sales do exist, they often lack efficient orders, real-time chat updates, and clear data analytics for both users and administrators.

There is a need for a specialized, secure, and efficient web-based solution that allows users to easily browse, purchase, and track such products online, while enabling business owners to manage products, monitor sales, analyze trends, and maintain inventory in a centralized, intuitive manner.

**Valid Treasure** addresses these gaps by offering an integrated platform tailored to both customers and administrators, ensuring a complete, efficient, and reliable online liquor purchasing and management system.

## 1.3 Objectives

**General Objective:**

To design and develop an efficient, user-friendly, and secure web-based application — **Valid Treasure**— that enables customers to browse, purchase, and pay for niche products online, while providing administrators with tools to manage pages, monitor orders, and analyze business performance.

**Specific Objectives:**

* To implement an intuitive web interface for browsing, searching, and purchasing niche products.
* To develop a user dashboard where customers can view their past orders and purchase history.
* To create an admin dashboard that facilitates product management, order status updates, and inventory tracking.
* To visualize pages data and user trends through graphs and charts for administrative analysis.
* To automate bans on pages that have many reports by valid users and have better page management.
* To implement secure authentication and role-based access control for both buyers, sellers and administrators.
* To build a scalable and reliable backend using **Django, SQLite, Django-Rest-Framework**.

## 1.4 Review of Related Work and Literature

In the recent past, the coming up of e-commerce sites has revolutionized the nature of consumers' interactions with marketplaces. The majority of these sites, however, specialize in mass-market offerings with little focus on niche segments, especially in gaming-related offerings such as retro collectibles, hard-to-find trading cards, or independent game merchandise. A lack of specialized platforms has resulted in users relying on general marketplaces like eBay, Facebook Marketplace, and Craigslist that are prone to security issues, a bad user experience, and poor customer support (Smith & Johnson, 2021).

Specialized marketplaces like **StockX** for collectibles and sneakers or **TCGPlayer** for trading cards offer a glimpse into the advantages of specialized marketplaces. Specialized marketplaces focus on authentication, detailed analytics, and user reputation systems. **StockX's** approach towards real-time data visualization and seller verification is especially commendable, which reflects how such an integration enhances trust and the platform itself (Martinez, 2020). Yet, these remedies remain comparatively narrow in scope and tend to be product category-specific without enabling users directly to create personalized selling pages.

Technically, Django Rest Framework (DRF) and React have become dominant in scalable web applications due to their modularity as well as performance. With the component-based UI of React, it is easy to deliver seamless user experience on devices, whereas DRF supports RESTful API development to handle user roles, inventory, and analytics.

Briefly, while a few sites target specialized product sales, there remains a huge void in providing a coherent, secure, and user-focused experience optimized for gaming-related products. Valid Treasure draws best practices from the literature and existing sites, and innovates by means of customized page construction, real-time interaction, and straightforward admin tools to meet the unique requirements of this under-served niche.

## 1.5 Development Methodology

The development of **Valid Treasure** was carried out using an **iterative and incremental methodology**, allowing for continuous refinement throughout the project lifecycle. Given the dynamic nature of feature requirements and design adjustments, a flexible approach was essential to effectively manage changes and improvements.

Key characteristics of the development process included:

* **Requirement Gathering and Analysis**: Initial project requirements were identified through personal observation of existing nice products selling platforms and consideration of potential enhancements such as personalized insights and administrative dashboards.
* **Incremental Development**: The project was broken down into manageable modules (user authentication, user roles, page system and admin work, and admin dashboard functionalities). Each module was developed, tested, and refined individually before moving on to the next.
* **Continuous Testing and Feedback**: After each major component was developed, internal testing was conducted to ensure functionality, usability, and integration with other modules. This allowed early detection and resolution of issues.
* **Adaptability to Changes**: As the project evolved, certain additional features (such as page bans, page likes and favorites, and admin monitoring) were introduced based on feasibility and time constraints, demonstrating a flexible and evolving project scope.

The iterative model ensured that core functionalities were prioritized and implemented early, while additional features were progressively incorporated based on ongoing evaluation and improvement opportunities.

## 1.6 Scope and Limitations

**Scope**

* **Customer Features**
  + User Roles and Authentication: Secure, role-based access for administrators, buyers, and sellers.
  + Page and Product Listings: Sellers have the ability to build dedicated pages for their products. Buyers can search, browse, and favorite these pages.
  + Real-Time Chat: Integration of chat rooms for buyer-seller communication to help clarify product details and build trust.
  + Analytics and Visualization: Charts and graphs show trends in user activity, sales performance, and page popularity.
  + Admin Panel: Report problems are addressed by the administrators, scam pages are blocked, and overall activity is monitored on the platform.
  + User Experience: Buyers can see a list of their most liked pages, and buy based on trust built over time.
* **Technical**
  + Backend and Frontend Stack: Developed on Django and Django REST Framework (DRF) for the backend, React for the frontend, and SQLite for local database storage with minimal load.
  + Authentication and role-based access control for users and admins.

**Limitations**

* **Limited Payment Integration**:  
  The current version may not be integrated with payment gateways like ESewa or Stripe and PayPal and instead employ direct communication between buyer and seller to facilitate payments.
* **Desktop-Only Interface**:  
  The application is optimized for desktop browsers; mobile responsiveness and a dedicated mobile app are not in this version.
* **Not Optimal For Scalability**:  
  SQLite, light but development-supportive and applicable to small-scale deployment, is not ideal for a lot of traffic or concurrent large-scale use.

## 

## 1.7 Report Organization

This report is structured as follows:

* **Chapter I: Introduction** :This chapter provides the background, problem statement, objectives, and an overview of the related work in the field. It also includes the methodology used for development, the scope, and the limitations of the project.
* **Chapter II: System Development Process**: This chapter elaborates on the detailed development process, including the analysis, design, and implementation stages of the **Valid Treasure** web application. It covers requirements gathering, the choice of technologies, system architecture, database design, and the overall system workflow. Additionally, this chapter includes a description of the key features and testing procedures applied throughout the project.
* **Chapter III: Conclusion and Recommendation:** This chapter summarizes the project’s achievements, conclusions drawn from the implementation, and offers recommendations for future improvements and further research in the field of e-commerce web applications.

# Chapter II: System Development Process

## 2.1 Analysis

### 2.1.1 Requirement Analysis

**Functional Requirements**

* **User Authentication & Role Management**
  + Users can register and log in using a secure authentication system.
  + Each user is assigned a role (buyer, seller, or admin) upon registration or admin approval.
  + Secure JWT-based authenticationis implemented for session handling.
* **Seller Pages & Product Listings**
  + Sellers can create customizable pages where they list products.
  + Each page includes a page name**,** description**,** profile image**,** andproduct list.
* **Product Browsing & Favorites (Buyers)**
  + Buyers can search for products or browse seller pages.
  + Buyers can favorite pages for quick access.
* **Admin Dashboard**
  + View past orders and order details.
  + See purchase history visualized as simple graphs.
* **Chat & Communication**
  + Buyers and sellers can initiate chat conversations within the platform.
  + Chats are stored and accessible for reference under user accounts.

**Non-Functional Requirements**

* **Security**: Secure password storage (hashed).
* **Performance**: Page load optimized to handle the 3d model on landing page and handle
* **Usability**: Intuitive navigation, clear labels, and feedback on user actions.
* **Maintainability**: Modular codebase, well-documented APIs.
* **Scalability**: Ability to handle up to 1,000 simultaneous users in future deployments.

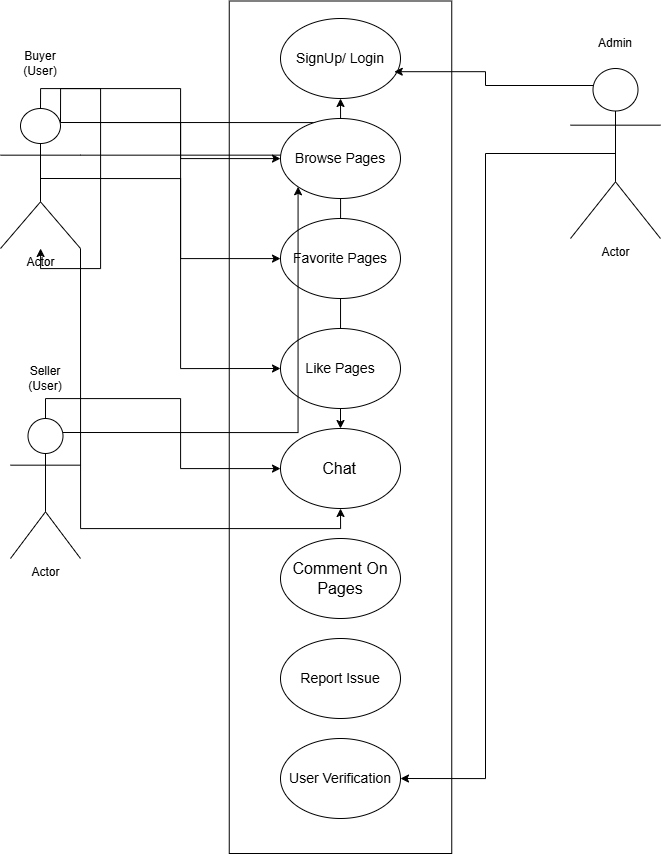
### 2.1.2 Feasibility Study

* **Technical Feasibility**
  + React and Django is well suited for rapid development of dynamic web apps.
  + Developer familiarity with JavaScript and python and React reduced the learning curve.
* **Economic Feasibility**
  + Open-source technologies keep licensing costs at zero.
  + Deployment can be done on low-cost cloud instances or free-tier services during development.
* **Operational Feasibility**
  + The desktop-only web interface matches the target user base (gamers and such customers).
  + Minimal training required—users interact via familiar web controls and clear labels.
* **Schedule Feasibility**
  + Core modules (authentication, user roles, page system) prioritized to fit within the academic timeline.
  + Only one payment gateway (eSewa) was integrated due to time constraints.

### 2.1.3 Modeling

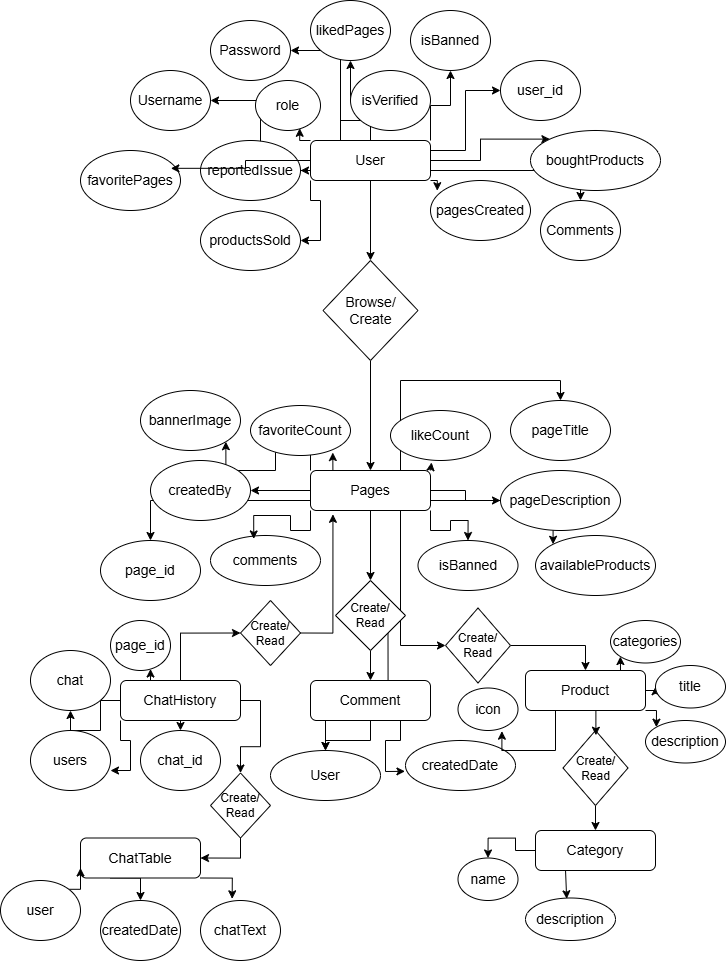
To visualize system functionality and data structure, the following models were created:

* **Use-Case Diagram**  
  Illustrates interactions between **Actors** (Buyers, Sellers, Admin) and **Use Cases** (Login, Browse Pages, Chat, Manage Pages, View Reports etc).

****

**Figure 2.1: Use-Case Diagram.**

* **Entity–Relationship Diagram (Database Schema)**  
  Shows main collections—Users, Products, Orders, Payments—and their relationships (e.g., one Seller→ many Products; each Product→ multiple Category).



**Figure 2.2: Database Schema Diagram.**

## 2.2 Design

### 2.2.1 User Interface Design

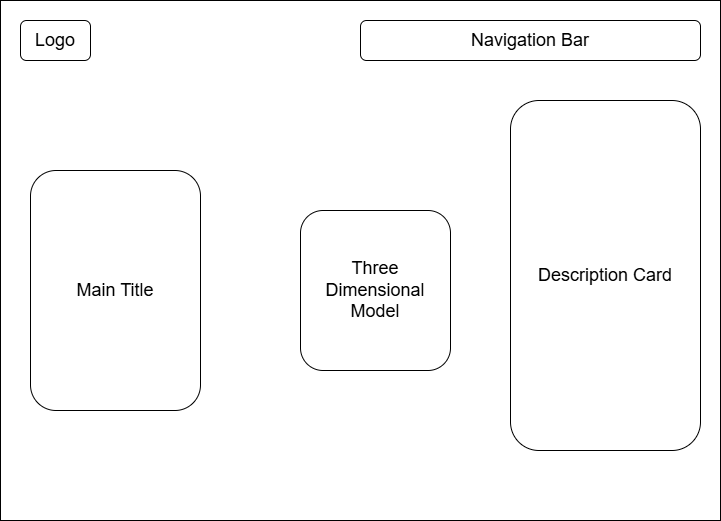
The user interface (UI) design of **Valid Treasure** follows a minimalistic approach, focusing on user ease and streamlined purchasing. Since the web application is not responsive, the layout is optimized for desktop view only. The key pages include:

* **Landing Page**: Features a unique experience with 3d model, animations, and features. It also highlights what the page is for.

****

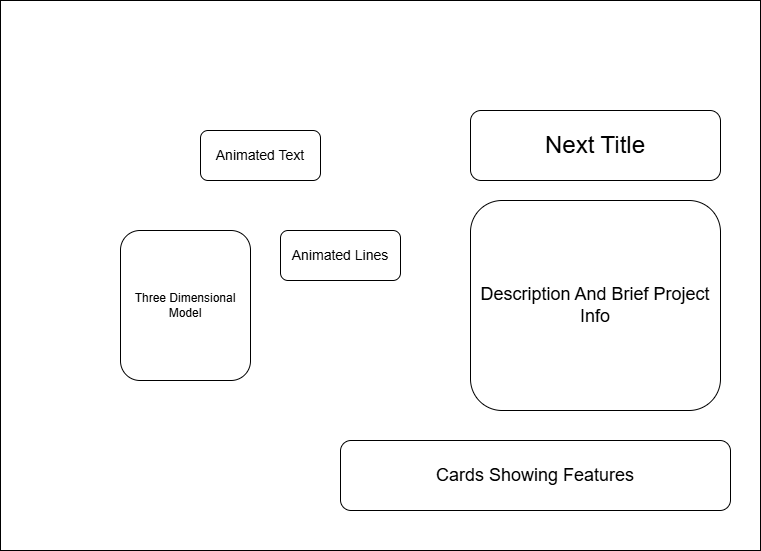
**Figure 2.3: Home Page.**

* **Home Page wireframe**: Displays wireframe of the landing page where the user lands first.

****

**Figure 2.4: Home Page Wireframe.**

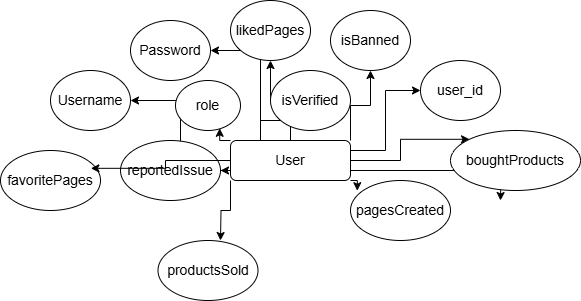
* **Second Page**: Allows users to view about the project, and what they are working with, what they can expect and what features are available.

  
**Figure 2.5: Second Page Wireframe.**

### 2.2.2 Database Design

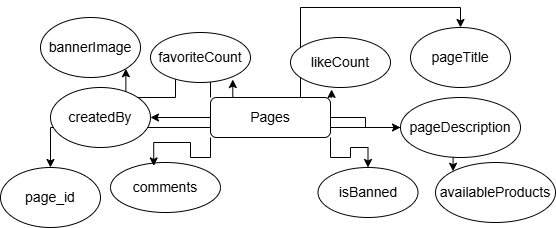
The database for **Valid Treasure** uses SQLite, a SQL database suitable for flexible data storage. The following main collections were defined:

* **Users**: Contains user information (name, password, address, role etc).



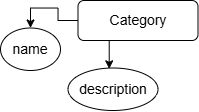
**Figure 2.6: ERD of User Table.**

* **Pages**: Stores page details (banner image, like count, created by, available products etc).



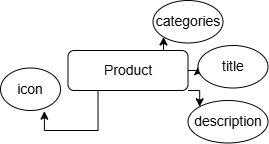
**Figure 2.7: ERD of Page Table.**

* **Category**: Contains details of product category (name, category id, description).

****

**Figure 2.8: ERD of Category Table.**

* **Product**: Stores product details (product id, title, description, icon and category of product).

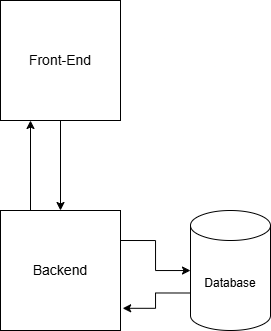


**Figure 2.9: ERD of Product Table.**

### 2.2.3 System Architecture

The **ValidTreasure** system architecture is built using React and Django. The components interact as follows:

* **Frontend (React)**: Provides the user interface for both buyers, sellers and admins, including the page details, user chats, and page management.
* **Backend (Django)**: Handles API requests, manages authentication, and processes through views.
* **Database (SQLite)**: Stores user data, product details, orders, and other information.

****

**Figure 2.10: System Architecture Diagram.**

## 2.3 Implementation

The implementation phase of **Valid Treasure** involved the actual development of the web application using React and Django technology stack. The main objective was to build a fully functional and user-friendly online shopping platform with a simple and secure process through pages and chat integration.

### 2.3.1 Frontend Implementation

The frontend of Valid Treasure was developed using **React.js**.  
Key points:

* Implemented a clean, minimalistic, and categorized product listing layout.
* Enabled users to view pages details and add pages to their favorites.
* Browse pages and other features were built using React's context API for state management.
* Form validations for checkout inputs and login/signup screens were handled on the client-side.
* The frontend was designed primarily for **desktop view**; **responsive design was not a major focus** due to time limitations.
* Styling was implemented using **Tailwind CSS** and **GSAP for animations**.

### 2.3.2 Backend Implementation

The backend server was developed using **Django** and **Django Rest Framework**.  
Main functionalities:

* Authentication system with JWT tokens for secure user login and protected routes.
* API endpoints for page management, product management, and user management.
* Admin functionalities to add, update, and delete pages.
* Validate user status update system to track user status(Verified ? Login : Can’t Login).
* Middleware handling for authentication and role-based access control (admin vs users).

### 2.3.3 Database Implementation

**SQLite** was used as the primary database.

Collections:

* **Users**: Stores user profiles, hashed passwords, and roles (admin/customer).
* **Products Table**: Stores product details like name, description, price, stock, category, and images.
* **Pages Table**: Stores orders placed by users along with product references, order status, and a transaction\_id field updated after successful payment.
* **Category Table**: Stores category details like name, description, slug, id.

### 2.3.5 Testing

Testing in Valid Treasure focused primarily on **manual validation** of core functionalities to ensure the reliability of the application.  
Key areas of testing included:

* **Authentication Flow:**  
  Verifying user registration, login, and role-based access (admin vs normal user).
* **Page Management:**  
  Checking product addition, update, deletion, and correct listing under categories.
* **Chat Operations:**  
  Ensuring smooth adding/removing of items to cart, cart summary accuracy, and transition to checkout.
* **Category Management:**  
  Testing categories of products to be already there, product can be listed by using one of the categories.
* **Comment Management:**  
  The pages contain comments which are crucial to maintain the validity of the page and the seller.
* **Dashboard Analytics:**  
  Admins can view which pages are most used, which has most likes and favorites, and such.

**Note:**  
Formal automated testing (unit testing, integration testing) was **not implemented** in this version due to project timeline limitations. Manual testing was conducted extensively across different devices and browsers to ensure functional consistency.

# Conclusion and Recommendation (Chapter III)

## 3.1 Summary

This chapter detailed the **System Development Process** of Valid Treasure, covering its complete journey from analysis to implementation.  
The **analysis phase** focused on understanding user requirements, defining key use cases, and establishing the database schema.  
The **design phase** involved creating preliminary wireframes and outlining the system architecture connecting frontend and backend components.  
In the **implementation phase**, the application was developed using the React and Django stack, integrating essential features such as page browsing, user operations, page management, and dashboard analytics for administrators.  
**Manual testing** was conducted throughout the project to validate functional correctness.  
Overall, the chapter highlights the practical application of planning, designing, building, and verifying a modern web-based social and to be e-commerce platform or system tailored to the niche products.

## 3.2 Conclusion

Valid Treasure was developed to provide a **dedicated shopping platform** with availability of niche products and such.  
Throughout the project, the goal was to build an intuitive system that caters to both **customers** (for easy product browsing and purchasing) and **administrators** (for efficient management and analysis of pages and users).

Key achievements include:

* A **functional customer interface** enabling pages browsing, page management, and chat interface.
* An **admin dashboard** offering real-time data on top-selling pages, banning of pages, via charts.
* A **streamlined backend architecture** utilizing Django and SQLite for modular and scalable development.
* **Secure product handling** with admin monitoring and an option to report issue if any of the users face difficulties.

The application successfully meets its primary objectives despite **certain constraints**:

* **Responsiveness** was not implemented, limiting optimal mobile usability.
* **Scalability Issues** if using the default Django SQLite database.
* **Basic wireframing and schema visualization** were utilized, with some planned advanced diagrams omitted.

Nevertheless, the system provides a **solid foundation** for a fully operational web application and demonstrates effective practical use of React and Django technologies.

## 3.3 Recommendations

While Valid Treasure achieves its core functionality, several improvements can be considered for future development:

* **Implement responsive design** to ensure a seamless experience across devices (mobile, tablet, desktop).
* **Integrate payment gateways** (e.g., Khalti, Esewa, Stripe, bank cards) for offering customer flexibility.
* **Expand the dashboard analytics** by including metrics like user engagement, repeat purchase rates, and sales forecasting.
* **Enhance security features** such as two-factor authentication for both customers and admins.
* **Improve UI/UX** with better personalization features like "Recommended Products" based on purchase history.
* **Conduct automated testing** (unit, integration, and end-to-end) to ensure better reliability as the platform scales.
* **Maintain complete ERD and documentation** from early stages in future projects to speed up development and validation.

# References (Chapter IV)

Facebook. (2024). *React – A JavaScript library for building user interfaces*. <https://reactjs.org/>

Django Software Foundation. (2024). *Django documentation*. <https://docs.djangoproject.com/en/stable/>

TroisJS. (2024). *React Three Fiber Documentation*. <https://docs.pmnd.rs/react-three-fiber/>

eSewa. (n.d.). *eSewa Digital Wallet Nepal*. Retrieved April 2025, from <https://esewa.com.np>

React Documentation. (n.d.). *React – A JavaScript library for building user interfaces*. Retrieved Jan 2025, from <https://react.dev>

Framer. (2024). *GSAP (GreenSock Animation Platform)*. <https://greensock.com/gsap/>

# Appendix (Chapter V)

## Appendix A : Code Snippets

### A.1 - POST /api/validTreasure/pages / (Page Creation)

class PageList(APIView):

    permission\_classes = [IsAuthenticated]

    def get(self, request):

        if not request.user.is\_admin:

            return Response({"error": "You are not an admin"}, status=status.HTTP\_403\_FORBIDDEN);

        halls = page.objects.all()

        serializer = PageSerializer(halls, many=True)

        return Response(serializer.data)

    def post(self, request):

        if not request.user.is\_admin:

            return Response({"detail:", "You are not an admin." }, status=status.HTTP\_403\_FORBIDDEN)

        serializer = PageSerializer(data=request.data)

        if serializer.is\_valid():

            serializer.save()

            return Response(serializer.data, status=status.HTTP\_201\_CREATED)

        return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

    def delete(self, request):

            if not request.user.is\_admin:

                return Response({"detail": "You are not an admin."}, status=status.HTTP\_403\_FORBIDDEN)

            try:

                hall = Page.objects.get(hall\_id=request.data.get('hall\_id'))

            except Page.DoesNotExist:

                return Response({"detail": "Hall not found."}, status=status.HTTP\_404\_NOT\_FOUND)

            page.delete()

            return Response({"detail": "Hall deleted successfully."}, status=status.HTTP\_204\_NO\_CONTENT)

### A.2 - GET /api/validTreasure/product/ (Product List Retrieval)

//get all products

class ProductList(APIView):

    permission\_classes = [IsAuthenticated]

    def post(self, request):

        if not request.user.is\_admin:

            return Response({"detail": "You are not an admin."}, status=status.HTTP\_403\_FORBIDDEN)

        serializer = ProductSerializer(data=request.data)

        if serializer.is\_valid():

            serializer.save()

            return Response(serializer.data, status=status.HTTP\_201\_CREATED)

        return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

    def get(self, request):

        shows = Product.objects.all()

        serializer = ProductSerializer(shows, many=True)

        return  Response(serializer.data, status=status.HTTP\_200\_OK)

    def delete(self, request):

        if not request.user.is\_admin:

            return Response({"detail": "You are not an admin."}, status=status.HTTP\_403\_FORBIDDEN)

        try:

            movies = Product.objects.filter(movie\_id=request.data.get('movie\_id')).first()

        except Product.DoesNotExist:

            return Response({"detail": "Movie not found."}, status=status.HTTP\_404\_NOT\_FOUND)

        movies.delete()

        return Response({"detail": "Movie deleted successfully."}, status=status.HTTP\_204\_NO\_CONTENT)

### A.3 - Schema Definition of User model

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

    name:{

        type:String,

        required:true,

        trim:true

    },

    email:{

        type:String,

        required:true,

    },

    password:{

        type:String,

        required:true,

    },

    address:{

        type:String,

        required:true,

    },

    answer:{

        type:String,

        required:true,

    },

    phone:{

        type:String,

        required:true,

    },

    role:{

        type:Number,

        default:0

    }

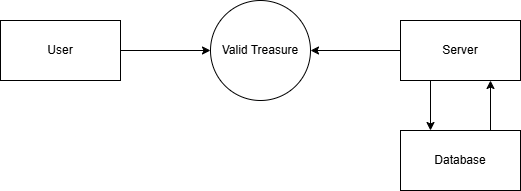
},{timestamps:true});

const userModel= new mongoose.model('user',userSchema);

module.exports = userModel;

## Appendix B : System Context Diagrams and Component tree Diagram

### D.1 System Context Diagram

****

**Fig no : D.1.1**

### D.2 Component Tree Diagram

├── RouterProvider

│ ├── LandingPage

│ ├── Auth

│ │ ├── Login

│ │ └── Register

│ ├── AboutPage

│ ├── NotFound

│

├── BuyerDashboard (Protected)

│ ├── BrowsePages

│ │ ├── SearchBar

│ │ ├── PageCard

│ │ └── CategoryFilter

│ ├── Favourites

│ ├── Orders

│ │ ├── OrderCard

│ │ └── PurchaseHistoryChart ( Chart.js or Recharts)

│ ├── 💬 ChatRoom

│ │ ├── MessageList

│ │ └── MessageInput

│

├── SellerDashboard (Protected)

│ ├── MyPage

│ │ ├── PageSettings

│ │ ├── ProductList

│ │ │ ├── ProductCard

│ │ │ ├── AddProductForm

│ │ │ └── EditProductModal

│ ├── Orders

│ │ ├── OrderManagement

│ │ └── OrderStatusUpdate

│ ├── Analytics

│ │ └── SalesChart

│

├── AdminDashboard (Protected)

│ ├── Reports

│ │ ├── ReportedPagesList

│ │ └── BanActions

│ ├── ManageUsers

│ ├── AdminAnalytics

│

├── Shared Components

│ ├── Navbar

│ ├── Sidebar (Responsive)

│ ├── ProductCard

│ ├── ToastNotifications (via React Toastify)

│ ├── AnimationsWrapper (GSAP hooks/components)

│ ├── CanvasScene (React Three Fiber)

│ └── AxiosInstance (API handler with interceptors)