VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

KUO NHAN DUNG – 520H0619
LE GIA PHU – 520H0401

EZTICKET: A TICKET MANAGEMENT PLATFORM BASED ON MERN STACK APPROACH

INFORMATION TECHNOLOGY PROJECT

SOFTWARE ENGINEERING MAJOR

HO CHI MINH CITY, 2024

VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

KUO NHAN DUNG – 520H0619
LE GIA PHU – 520H0401

EZTICKET: A TICKET MANAGEMENT PLATFORM BASED ON MEARN STACK APPROACH

INFORMATION TECHNOLOGY PROJECT

SOFTWARE ENGINEERING MAJOR

Instructor

Lecturer. Duong Huu Phuc

HO CHI MINH CITY, 2024

ACKNOWLEDGMENTS

We would like to thank **Mr. Duong Huu Phuc** who has guided us through our project and given us notes for our report.

There are some errors within our report, we hope that this does not affect the overall report. Please give us suggestions in order for us to improve our report.

Ho Chi Minh City, March 3rd 2024.
Author
(Sign and write your full name)

Kuo Nhan Dung,

Le Gia Phu

WORKS COMPLETED
AT TON DUC THANG UNIVERSITY

We would like to assure that this report is our own research and under the guidance of **Mr. Duong Huu Phuc**. The research contents and results in this report have not been published in any format before. The table data for analysis and evaluations collected from different sources by the authors are stated in the references section.

Additionally, the project uses comments, reviews and figures from other authors and organizations with annotations of origin.

If any plagiarism is detected, we would like to take full responsibility for the content of our project. Ton Duc Thang University is not related to copyright violations caused by our report in the process of implementation (if any).

Ho Chi Minh City, March 3rd 2024.

Author

(Sign and write your full name)

Kuo Nhan Dung,

Le Gia Phu

EZTICKET: A TICKET MANAGEMENT PLATFORM BASED ON MERN STACK APPROACH

ABSTRACT

Our project goal is to conduct research and develop a management system with the aims to manage events and tickets created by event organizers and purchased by customers. Our approach for this management system is to use MERN (MongoDB, ExpressJS, ReactJS and NodeJS) stack technology. MongoDB is used as a database to store data in JSON format, ReactJS is used as a front-end server to handle and render data, NodeJS and ExpressJS are used as back-end server to process requests and data.

The development of our project has resulted in a modern, user-friendly management system and platform with security in managing and purchasing events and tickets. Our project seeks to give a general understanding in developing a management system using MERN stack and implementing user data analyzation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

MERN	MongoDB, ExpressJS, ReactJS and NodeJS
ODM	Object Data Model
SQL	Structured Query Language
No SQL	No Structured Query Language
ERD	Entity – Relational Diagram
GUI	Graphical User Interface
CRUD	Create, Read, Update and Delete

CHAPTER 1. INTRODUCTION

1.1 Introduction

When conducting research for our project, we encountered several problems involving platform analysis and system processes such as:

- The scarcity of platforms that functions as an open marketplace and treating event tickets as products.
- Platforms built based on old technology or the absent of modern multi-platform payment methods like Paypal and Stripe.
- Platforms which cater exclusively to certain types of event tickets or organizations.

As a result, developing a ticket management platform with the approach of using MERN stack technology will help to provide a marketplace that supplies a vast variety of event tickets; modern experience for the end users; an effortless maintenance cycle and technology integration for developers. Furthermore, this project implementation will contribute to resolving some issues, including:

- Easy technology implementation and maintenance: Javascript and JSON are the programming language and format used for our MERN stack approach and using package manage system like **`npm`** or **`yarn`** help to create a seamless development process.
- Management efficiency: Provides effective events and tickets management functions that helps track detailed information, number of purchased tickets, reducing complexity in the organization process and assisting user data analyzation.
- Modern user experience: Applying modern packages, library and techniques into building layouts that can help improve user navigation when using the platform.

During our research, there are numerous ticket platforms but each of them is missing certain features. So, through our analyzation we will gloss over the pros and cons of each platform in order to reflect on our project:

- Ticketbox[1]:
  - Pros: Provides users with common e-commerce layouts with a modern and friendly user interface. Supports multiple payment methods.
  - Cons: Has difficulties when handling with large events and events that have a specific category. Are more built toward IDECAF theater.
- Ticket Spice[2]:
  - Pors: Modern and easy to use navigation layouts. Supports large events. Allows organizers to have customization page.
  - Cons: Located overseas and supports only a handful of large stadiums only. Prompts users and organizers to subscribe to an annual fee payment instead of a contractual payment.
- Stub Hub[3]:
  - Pros: Platforms has modern layouts and easy to navigate routes. Has a wide range of ticket categories and event categories.
  - Cons: Is mainly supported in the USA and sport events. Has more customization and supports for large events, limited supports for small to medium events.

[1] https://ticketbox.vn/

[2] https://www.ticketspice.com/

[3] https://www.stubhub.com/

- EventZilla[4]:
    - Pros: Provides user friendly layouts and clear navigation menu. Has great event website customization for organizers with easy-to-read instructions. Has a wide variety of events.
    - Cons: Focuses more on small and medium events. Although EventZilla supports a wide variety of events, most of its event are in a single theme.
- SplashThat[5]:
    - Pros: Has a great customization page for event organizers. Supports in-house data analytics and marketing.
    - Cons: Has problems when handling large events. It's not suitable for specific industries

Based on the analysis of the pros and cons of existing ticket platforms, we will orient our development towards addressing these following issues:

- Enhance support and flexibility for managing diverse events: Creating a platform that can support a variety of events, from artistic performances to sports and educational events.
- Optimizing payment process and experimenting with different business model: Quick and simple payment process for customers; reducing transaction and service fees.
- Divide services into separate modules: Implement changes without effecting users' experience when using the platform.
- Implement encryption and JWT: Protect user and purchase information; shield protect data within requests and processes.

[4] https://www.eventzilla.net/us/home

[5] https://splashthat.com/

During our analysis phase, we have set our goals to use MongoDB, ExpressJS, ReactJS and NodeJS technology stack and other tools to assist in the development stages.

- MongoDB:
  - Reasons: Our data has many nested fields and they are constantly changing during development, which is why we choose NoSQL for the flexibility and scalability in database and schema design, NoSQL is able handle large volume of data and fields at high speed. MongoDB is chosen because it is the most used database for NoSQL, it is easy to setup and connect; MongoDB has a good GUI to visualize and manage databases, schemas and fields.
  - Pros: Data is stored in human-readable a JSON format. Supports complex queries and can handle large datasets efficiently.
  - Cons: Queried data post-processing are not as robust as some relational database systems.
- ReactJS:
  - Reasons: We want our customers to have a fast and seamless user experience and ReactJS is the most popular library that can support that with server-side rendering, fast rendering and single-page application. ReactJS is written using Javascript, which helps to unify our programming language from front-end to back-end. ReactJS has reusable components which we utilized to speed up the development process.
  - Pros: The virtual DOM optimizes website performance. Easily integrates with other libraries and frameworks.
  - Cons: Requires a solid understanding of ReactJS to fully leverage its features. Build time can take longer to complete.

- NodeJS and ExpressJS:
  - Reasons: Using NodeJS and ExpressJS to unify our back-end programming language with the front-end by using Javascript. To assist in developing an API server to serve data, we use NodeJS and ExpressJS because they are lightweight, has multiple assisting node packages and a large community to support.
  - Pros: Written using Javascript which can be easily integrates with other libraries and middleware. Lightweight and suitable for developing API services.
  - Cons: Lacks a unified structure, may lead to complexity in developing large applications. Not suitable for tasks requiring multi-threaded processing.
- Postman:
  - Reasons: Postman is used for testing our created API services and creating API documentations and examples.
  - Pros: Postman is simple to install and use. Postman can be operated with a client or in the browser. Creating API documentations effortlessly.
  - Cons: Postman functions are limited offline. API documentation format is simple and plain.
- SUD-Libs[6]:
  - Reasons: We developed and published our own node package to handle emailing, payment and encryption process because some node packages have a large folder and file size but we only need a few functions within them.
  - Pros: Lightweight, can be easily downloaded and imported to multiple different projects.

[6] https://www.npmjs.com/package/sud-libs

  - Cons: Support specific functionalities. Need to be updated and patched regularly to fix bugs.

1.2 Objectives and scope

Using the MERN stack approach, we aim to focus on resolving these difficulties:

- A ticket management platform with a diverse variety of tickets and categories.
- Modern layouts to help users with their experience.
- API services for easy development, management and maintenance of source code.
- Provide tools to help staffs analyze customer data and customer purchase data.

Our ticket management platform creates a marketplace for a variety of ticket categories with modern layouts, which enhance user experience. It also with support for seamless technology integration and tools for analyzing user purchase information.

The scope of our project will focus on developing these functionalities with the specified functions and limitations include:

- Event management: Create, edit, and delete event information, including descriptions, venues, dates, times, and representative images.
- Ticket management: Create, edit, and delete ticket information, including descriptions, date and time.
- Ticket sales and payments: Allow online ticket purchases and provide secure payment methods.
- User management: Register, login, and manage user personal information.
- Basic reporting and statistics: Provide basic reports on revenue, the number of tickets sold, and basic statistics.

1.3 Practical meaning

The successful development of ticket management platform using the MERN stack approach will help contributing to:

- Providing a platform that can support a variety of ticket categories and a seamless process for creating events for event organizers.
- Creating a platform with security in purchasing process and support multiple payment methods.
- Gathering data for statistical reports and strategic decision-making to category potential bad actors toward the system.

1.4 Report structure

CHAPTER 1: INTRODUCTION

CHAPTER 2: SYSTEM ANALYSIS AND DESIGN

CHAPTER 3: SYSTEM IMPLEMENTATION

CHAPTER 4: SYSTEM DEMONSTRATION

CHAPTER 5: SUMMARY AND FUTURE WORK

CHAPTER 2. SYSTEM ANALYSIS AND DESIGN

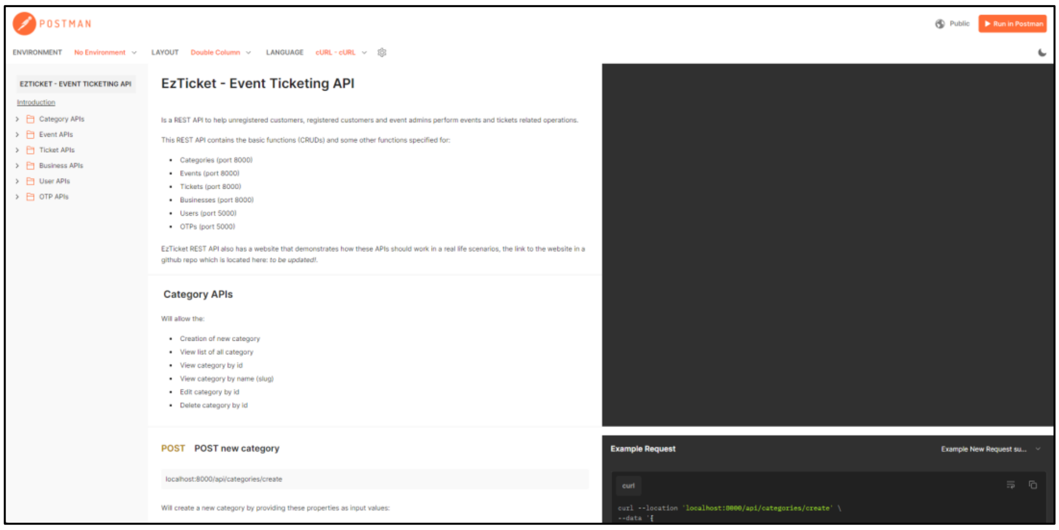
2.1 System description

The ticket management platform will be called "EzTicket" with the approach of using MERN stack as the development technology. EzTicket is a platform that allows for the management of events and tickets for event organizers; and a marketplace to purchase a variety ticket category for customers.

The project uses MERN stack technology which includes: MongoDB as the NoSQL database, storing data in JSON format, ReactJS is used as the front-end server to render data sent from the back-end server, NodeJS and ExpressJS are used to develop the back-end server and handle requests.

The project is divided into separate front-end and back-end server, with a third server being developed in Python using Flask to handle statistical user purchasing data used for analysis. The data transferred between servers, is using JSON format, it is encrypted using AES-256 encryption algorithm and the encryption keys are stored within the development environment for security.

Postman is used to test and save examples of response data sent from the back-end server. It is also used to create an API documentation, please follow this link to view the API document: https://tinyurl.com/EzTicketAPI.



ENVIRONMENT No Environment LAYOUT Double Column LANGUAGE cURL - cURL

EZTICKET - EVENT TICKETING API
Introduction
Category APIs
Event APIs
Ticket APIs
Business APIs
User APIs
OTP APIs

EzTicket - Event Ticketing API

Is a REST API to help unregistered customers, registered customers and event admins perform events and tickets related operations.

This REST API contains the basic functions (CRUDs) and some other functions specified for:

- Categories (port 8000)
- Events (port 8000)
- Tickets (port 8000)
- Businesses (port 8000)
- Users (port 5000)
- OTPs (port 5000)

EzTicket REST API also has a website that demonstrates how these APIs should work in a real life scenarios, the link to the website in a github repo which is located here: *to be updated!*.

Category APIs

Will allow the:

- Creation of new category
- View list of all category
- View category by id
- View category by name (slug)
- Edit category by id
- Delete category by id

POST POST new category

localhost:8000/api/categories/create

Will create a new category by providing these properties as input values:

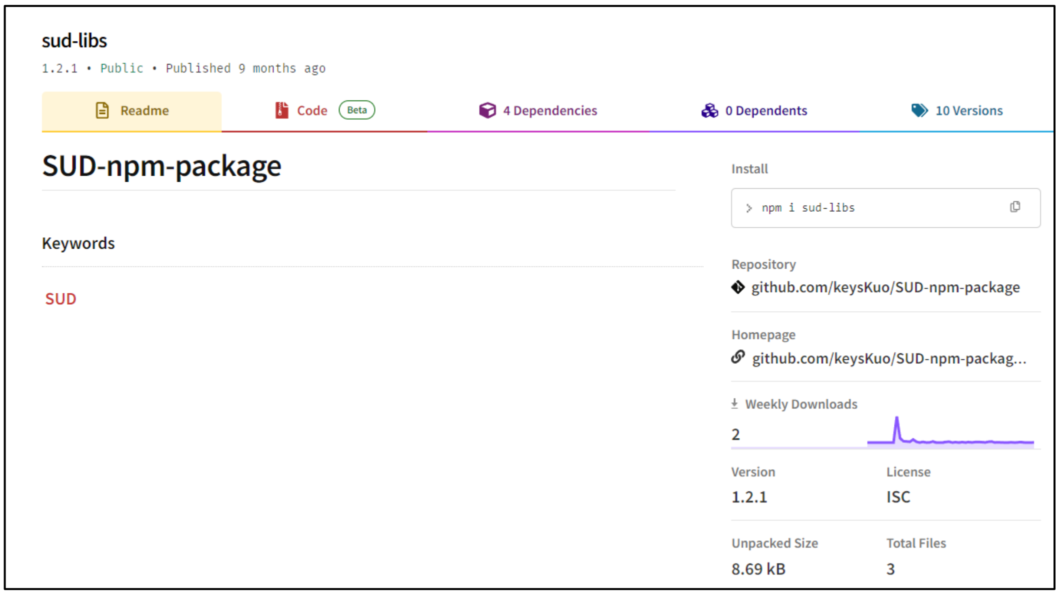
Example Request Example New Request su...

curl

```
curl --location 'localhost:8000/api/categories/create' \
--data '{
```

Figure 2.1. EzTicket API documentation by Postman.

This project also uses a custom node package that we developed and published called “sud-libs” package to handle email and payment methods. It is a lightweight package used to handle simple emailing and payment process requests, please follow this link https://www.npmjs.com/package/sud-libs.



sud-libs

1.2.1 • Public • Published 9 months ago

Readme | Code Beta | 4 Dependencies | 0 Dependents | 10 Versions

SUD-npm-package

Keywords

SUD

Install

```
> npm i sud-libs
```

Repository
github.com/keysKuo/SUD-npm-package

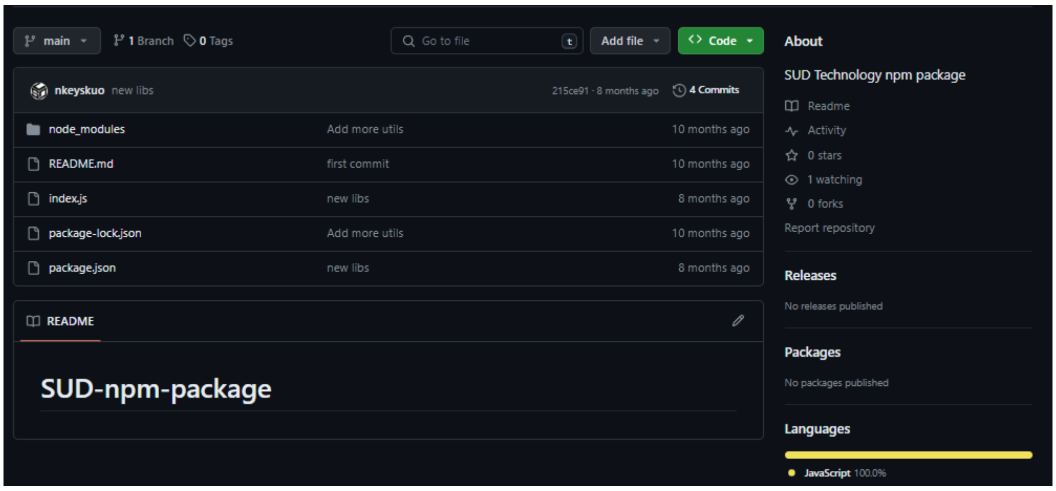
Homepage
github.com/keysKuo/SUD-npm-packag...

Weekly Downloads
2

Version	License
1.2.1	ISC

Unpacked Size	Total Files
8.69 kB	3

Figure 2.2.The “sud-libs” package on npmJS website.



main | 1 Branch | 0 Tags | Go to file | Add file | Code

nkeyskuo new libs — 215ce91 · 8 months ago — 4 Commits

node_modules	Add more utils	10 months ago
README.md	first commit	10 months ago
index.js	new libs	8 months ago
package-lock.json	Add more utils	10 months ago
package.json	new libs	8 months ago

README

SUD-npm-package

About

SUD Technology npm package

Readme

Activity

0 stars

1 watching

0 forks

Report repository

Releases

No releases published

Packages

No packages published

Languages

JavaScript 100.0%

Figure 2.3. The “sud-libs” Github page.

We use multiple choice and interview type questions to build the description, functional and non-functional requirements of the system (see Appendix A and Appendix B).

Functional requirements for EzTicket:

- Event management: Basic CRUD functions events with details such as description, date/time and image. Once an event is created it is in the reviewing stage and is published once it is reviewed.
- Ticket management: Basic CRUD functions new tickets with details such as description and date/time. Enable users to purchase the tickets and view in purchase history.
- User management: Register accounts and login to track events and purchase tickets. Manage personal information, transaction and purchasing history.
- Ticket purchasing process and payments: Allow customers to purchase tickets online through a user-friendly interface, if customers take too much time to complete the purchasing process, remove that customer and set the ticket to available state. Support multiple secure and convenient payment methods.
- Search and filter: Allows user to search for events or search for events by categories.
- Reporting and statistics: Provide basic reports on tickets revenue. Using user purchasing data analyze and create comparison for purchasing patterns.

Non-functional requirements for EzTicket:

- Security: Implement security measures to prevent network attacks and information exploitation and during payment process.
- Scalability: System have to be scalable in order to support new features and functionalities with fast response time.
- Reliability and error handling: System must have minimal downtime and error message should be clear and informative for troubleshooting.

2.2 Use-case diagram

2.2.1 Ticket Management System API Use-case Diagram

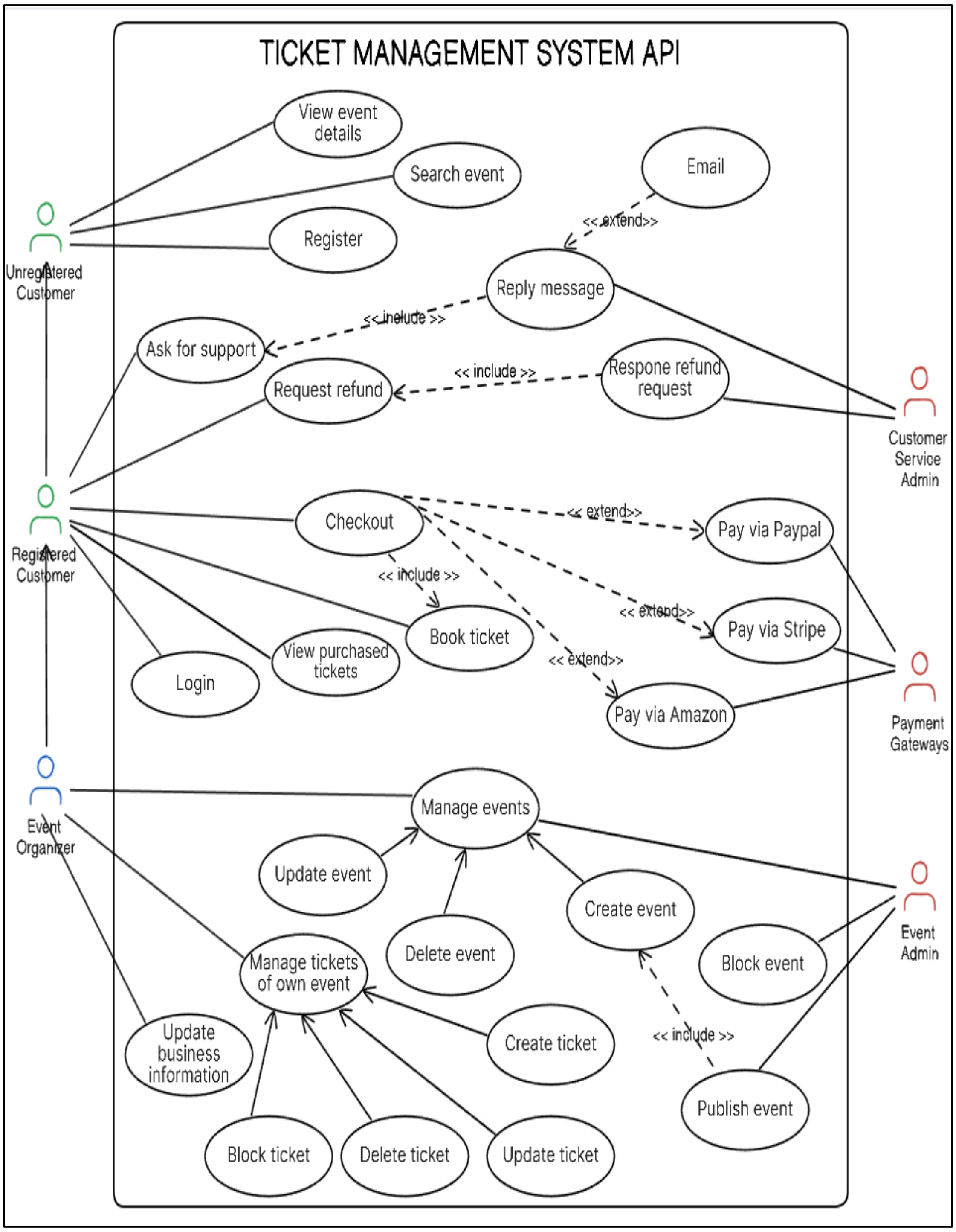


Figure 2.4. Ticket Management System API Use-case Diagram.

Ticket Management System use-case diagram has 6 actors:

- Actors for the staffs: Customer Service Admin and Event Admin.
- Actors for the users: Unregistered Customer, Registered Customer and Event Organizer.
- Actors for the system: Payment Gateway.

Registered Customer inherited use cases from Unregistered Customer. Event Organizer inherited use cases from Registered Customer.

2.2.2 Ticket Management System API Use-case Description

Table 2.1. Response Refund Request Use-case Description

Use-case: Response Refund Request UCID: 1	
Primary Actor	- Customer Service Admin
Stakeholders and Interest	
Brief Description	- Enable Customer Service Admin to response to refund request made by customers who have bought tickets
Pre-condition	- A request refund needs to be made by a customer who has bought a ticket
Trigger	- Customer Service Admin clicks on a refund ticket sent by the Customer
Relationship	- Response Refund Request <<*include*>> Request Refund
Flow of Event	1. Customer Service Admin logs into their account 2. Customer Service Admin accesses their Refund Request Ticket Dashboard 3. Customer Service Admin views Refund Request details

	4. Customer Service Admin responses to Refund Request Ticket

Table 2.2. Request Refund Use-case Description

Use-case: Request Refund **UCID: 2**	
Primary Actor	- Registered Customer
Stakeholders and Interest	- Event Organizer
Brief Description	- Enable Registered Customer (and Event Organizer) to make a refund request to their purchased tickets
Pre-condition	- Registered Customer (and Event Organizer) needs to have purchased a ticket - The purchased ticket's expiration date is still available
Trigger	- Registered Customer (and Event Organizer) clicks on a refund next to their tickets
Relationship	- Request Refund <<*include*>> Response Refund Request - Request Refund <<*association*>> Registered Customer
Flow of Event	1. Registered Customer logs into their account 2. Registered Customer check their purchased ticket dashboard 3. Registered Customer clicks refund next to the ticket
Alternative Flows	1. Event Organizer logs into their account

	2. Event Organizer check their purchased ticket dashboard 3. Event Organizer clicks refund next to the ticket
Exceptional Flows	1. Registered Customer logs into their account 2. Registered Customer check their purchased ticket dashboard 3. If Registered Customer ticket's expiration date is not available, the refund button will not be available

Table 2.3. Reply Message Use-case Description

Use-case: Reply Message **UCID: 3**	
Primary Actor	- Customer Service Admin
Stakeholders and Interest	
Brief Description	- Enable Customer Service Admin to Reply to message made by Registered Customer (and Event Organizer)
Pre-condition	- Registered Customer (and Event Organizer) need to send a support ticket
Trigger	- Customer Service Admin click the reply button
Relationship	- Reply Message <<*include*>> Ask for Support - Reply Message <<*extend* >> Email - Request Message <<*association*>> Customer Service Admin

Flow of Event	1. Customer Service Admin logs into their account 2. Customer Service Admin checks their support ticket dashboard 3. Customer Service Admin clicks reply message next to the ticket

Table 2.4. Email Use-case Description

Use-case: Email **UCID: 4**	
Primary Actor	- Customer Service Admin
Stakeholders and Interest	
Brief Description	- Enable Customer Service Admin to Reply to email message made by Registered Customer (and Event Organizer)
Pre-condition	- Registered Customer (and Event Organizer) need to send a support ticket
Trigger	- Customer Service Admin click the reply by email button next to the support ticket
Relationship	- Email <<*extend* >> Reply Message
Flow of Event	1. Customer Service Admin logs into their account 2. Customer Service Admin checks their support ticket dashboard 3. Customer Service Admin clicks reply message via email next to the ticket

Table 2.5. Ask for Support Use-case Description

Use-case: Ask for Support **UCID: 5**	
Primary Actor	- Registered Customer
Stakeholders and Interest	- Event Organizer
Brief Description	- Enable Registered Customer (and Event Organizer) to make a support request
Pre-condition	- Customer needs to be a Registered Customer or Event Organizer
Trigger	- Registered Customer (and Event Organizer) clicks on “request support” button
Relationship	- Ask for Support <<*include*>> Reply Message - Ask for Support <<*association*>> Registered Customer
Flow of Event	1. Registered Customer logs into their account 2. Registered Customer navigates to their support dashboard 3. Registered Customer creates a “request support” ticket in the dashboard
Alternative Flows	1. Event Organizer logs into their account 2. Event Organizer navigates to their support dashboard 3. Event Organizer creates a “request support” ticket in the dashboard

Table 2.6. Checkout Use-case Description

Use-case: Checkout UCID: 6	
Primary Actor	- Registered Customer
Stakeholders and Interest	- Event Organizer
Brief Description	- Enable Registered Customer to create a checkout for their purchases
Pre-condition	- Registered Customer (and Event Organizer) needs to have book a ticket
Trigger	- Registered Customer (and Event Organizer) clicks checkout after finishing booking their tickets
Relationship	- Checkout <<*association*>> Registered Customer - Checkout <<*include*>> Book Ticket - Checkout <<*extend* >> Pay via Paypal - Checkout <<*extend* >> Pay via Stripe - Checkout <<*extend* >> Pay via Amazon
Flow of Event	1. Registered Customer logs into their account 2. Registered Customer select their tickets 3. Registered Customer book their tickets 4. Registered Customer create a checkout for their booked ticket(s)
Alternative Flow	1. Event Organizer logs into their account 2. Event Organizer select their tickets 3. Event Organizer book their tickets 4. Event Organizer create a checkout for their booked ticket(s)
Exceptional Flow	1. Registered Customer logs into their account

	2. Registered Customer select their tickets 3. Registered Customer book their tickets 4. Registered Customer create a checkout for their booked ticket(s) 5. If the Registered Customer took too much time to complete the checkout process, the booked ticket will be cancelled and the Registered Customer needs to perform the process again

Table 2.7. Pay via Paypal Use-case Description

Use-case: Pay via Paypal **UCID: 7**	
Primary Actor	- Payment Gateway
Stakeholders and Interest	- Registered Customer - Event Organizer
Brief Description	- Provide a Paypal payment gateway for the customers to interact with the system and Paypal
Pre-condition	- There must be a checkout created in the system - A stable connection between the system and the Paypal payment gateway
Trigger	- Registered Customer (and Event Organizer) invokes the checkout method
Relationship	- Pay via Paypal <<*association*>> Payment Gateway - Pay via Paypal <<*extend* >> Checkout

Flow of Event	1. Registered Customer logs into their account 2. Registered Customer select their tickets 3. Registered Customer book their tickets 4. Registered Customer create a checkout for their booked ticket(s) 5. Registered Customer selects Paypal as payment method
Alternative Flow	1. Event Organizer logs into their account 2. Event Organizer select their tickets 3. Event Organizer book their tickets 4. Event Organizer create a checkout for their booked ticket(s) 5. Event Organizer selects Paypal as payment method

Table 2.8. Pay via Stripe Use-case Description

Use-case: Pay via Stripe **UCID: 8**	
Primary Actor	- Payment Gateway
Stakeholders and Interest	- Registered Customer - Event Organizer
Brief Description	- Provide a Stripe payment gateway for the customers to interact with the system and Stripe
Pre-condition	- There must be a checkout created in the system - A stable connection between the system and the Stripe payment gateway

Trigger	- Registered Customer (and Event Organizer) invokes the checkout method
Relationship	- Pay via Stripe <<*association*>> Payment Gateway - Pay via Stripe <<*extend* >> Checkout
Flow of Event	1. Registered Customer logs into their account 2. Registered Customer select their tickets 3. Registered Customer book their tickets 4. Registered Customer create a checkout for their booked ticket(s) 5. Registered Customer selects Stripe as payment method
Alternative Flow	1. Event Organizer logs into their account 2. Event Organizer select their tickets 3. Event Organizer book their tickets 4. Event Organizer create a checkout for their booked ticket(s) 5. Event Organizer selects Stripe as payment method

Table 2.9. Pay via Amazon Use-case Description

Use-case: Pay via Amazon **UCID: 9**	
Primary Actor	- Payment Gateway
Stakeholders and Interest	- Registered Customer - Event Organizer

Brief Description	- Provide an Amazon payment gateway for the customers to interact with the system and Amazon
Pre-condition	- There must be a checkout created in the system - A stable connection between the system and the Amazon payment gateway
Trigger	- Registered Customer (and Event Organizer) invokes the checkout method
Relationship	- Pay via Amazon <<*association*>> Payment Gateway - Pay via Amazon <<*extend*>> Checkout
Flow of Event	1. Registered Customer logs into their account 2. Registered Customer select their tickets 3. Registered Customer book their tickets 4. Registered Customer create a checkout for their booked ticket(s) 5. Registered Customer selects Amazon as payment method
Alternative Flow	1. Event Organizer logs into their account 2. Event Organizer select their tickets 3. Event Organizer book their tickets 4. Event Organizer create a checkout for their booked ticket(s) 5. Event Organizer selects Amazon as payment method

Table 2.10. Book Ticket Use-case Description

Use-case: Book Ticket **UCID: 10**	
Primary Actor	- Registered Customer
Stakeholders and Interest	- Event Organizer
Brief Description	- Enable Registered Customer (and Event Organizer) to book a ticket
Pre-condition	- Event of the tickets must be publicly published - Tickets must be available
Trigger	- Registered Customer (and Event Organizer) select the tickets in the event
Relationship	- Book Ticket <<*association*>> Registered Customer - Book Ticket <<*include*>> Checkout
Flow of Event	1. Registered Customer logs into their account 2. Registered Customer selects an event 3. Registered Customer selects a ticket 4. Registered Customer books their tickets
Alternative Flow	1. Event Organizer logs into their account 2. Event Organizer selects an event 3. Event Organizer selects a ticket 4. Event Organizer books their tickets

Table 2.11. Seach Event Use-case Description

Use-case: Search Event **UCID: 11**	
Primary Actor	- Unregistered Customer
Stakeholders and Interest	- Event Organizer
Brief Description	- Enable Unregistered Customer and Registered Customer (and Event Organizer) to search for events
Pre-condition	- Event of the tickets must be publicly published
Trigger	- Unregistered Customer and Registered Customer (and Event Organizer) types in the search bar
Relationship	- Search Ticket <<*association*>> Unregistered Customer
Flow of Event	1. Unregistered Customer navigates to website 2. Unregistered Customer types into the search bar 3. Unregistered Customer searches for events
Alternative Flow	1. Event Organizer navigates to website 2. Event Organizer types into the search bar 3. Event Organizer searches for events

Table 2.12. View Purchased Ticket Use-case Description

Use-case: View Purchased Ticket **UCID: 12**	
Primary Actor	- Registered Customer
Stakeholders and Interest	- Event Organizer
Brief Description	- Enable Registered Customer (and Event Organizer) view their purchased tickets
Pre-condition	- Registered Customer (and Event Organizer) needs to have successfully book a ticket
Trigger	- Registered Customer (and Event Organizer) clicks on purchased ticket dashboard
Relationship	- View Purchased Ticket <<*association*>> Registered Customer
Flow of Event	1. Registered Customer logs into their account 2. Registered Customer selects "purchased ticket" dashboard
Alternative Flow	1. Event Organizer logs into their account 2. Event Organizer selects "purchased ticket" dashboard
Exceptional Flow	1. Registered Customer logs into their account 2. Registered Customer selects "purchased ticket" dashboard 3. If there are not any tickets that have been successfully purchased, the dashboard will return an empty list

Table 2.13. Login Use-case Description

Use-case: Login **UCID: 13**	
Primary Actor	- Registered Customer
Stakeholders and Interest	- Event Organizer
Brief Description	- Enable Registered Customer (and Event Organizer) to log into their account
Pre-condition	- Registered Customer (and Event Organizer) needs to have created an account
Trigger	- Registered Customer (and Event Organizer) clicks on log in
Relationship	- Login <<*association*>> Registered Customer
Flow of Event	1. Registered Customer navigates to the website 2. Registered Customer enters their information 3. Registered Customer clicks login
Alternative Flow	1. Event Organizer navigates to the website 2. Event Organizer enters their information 3. Event Organizer clicks login
Exceptional Flow	1. Customer navigates to the website 2. Customer enters their information 3. If their credential is incorrect the website will display an error

Table 2.14. Register Use-case Description

Use-case: Register **UCID: 14**	
Primary Actor	- Unregistered Customer
Stakeholders and Interest	- Event Organizer
Brief Description	- Enable Unregistered Customer (and Event Organizer) to register their account
Pre-condition	
Trigger	- Unregistered Customer (and Event Organizer) clicks on register
Relationship	- Register <<*association*>> Unregistered Customer
Flow of Event	1. Unregistered Customer navigates to the website 2. Unregistered Customer enters their information 3. Unregistered Customer clicks register
Alternative Flow	1. Event Organizer navigates to the website 2. Event Organizer enters their information 3. Event Organizer clicks register
Exceptional Flow	1. Customer navigates to the website 2. Customer enters their information 3. If their credential already existed the website will display an error

Table 2.15. View Event Detail Use-case Description

Use-case: View Event Detail UCID: 15	
Primary Actor	- Unregistered Customer
Stakeholders and Interest	- Event Organizer
Brief Description	- Enable Unregistered Customer (and Event Organizer) to view the details of a published event on the website
Pre-condition	- An event must be published on the website
Trigger	- Unregistered Customer (and Event Organizer) clicks on “view event detail”
Relationship	- View Event Detail <<*association*>> Unregistered Customer
Flow of Event	1. Unregistered Customer navigates to the website 2. Unregistered Customer click an event 3. Unregistered Customer click the detail of an event
Alternative Flow	1. Event Organizer navigates to the website 2. Event Organizer click an event 3. Event Organizer click the detail of an event

Table 2.16. Manage Event Use-case Description

Use-case: Manage Event **UCID: 16**	
Primary Actor	- Event Organizer
Stakeholders and Interest	- Event Admin
Brief Description	- Enable Event Organizer (and Event Admin) to manage events (create, update and delete event)
Pre-condition	- Event Organizer (and Event Admin) needs to be logged in - Events (maybe) already created
Trigger	- Event Organizer (and Event Admin) clicks on "manage event" dashboard
Relationship	- Manage Event <<*association*>> Event Organizer - Manage Event <<*association*>> Event Admin - Manage Event <<*generalization*>> Update Event - Manage Event << *generalization*>> Create Event - Manage Event << *generalization*>> Delete Event
Flow of Event	1. Event Organizer navigates to the website 2. Event Organizer logs into their account 3. Event Organizer clicks "manage event" dashboard
Alternative Flow	1. Event Admin navigates to the website 2. Event Admin logs into their account

	3. Event Admin clicks “manage event” dashboard

Table 2.17. Block Event Use-case Description

Use-case: Block Event **UCID: 17**	
Primary Actor	- Event Admin
Stakeholders and Interest	
Brief Description	- Enable Event Admin to block events that have been published due to guidelines violation
Pre-condition	- A published event has violated guidelines
Trigger	- Event Admin clicks the block button next to an event
Relationship	- Block Event <<*association*>> Event Admin
Flow of Event	1. Event Organizer navigates to the website 2. Event Organizer logs into their account 3. Event Organizer clicks the block button next to an event

Table 2.18. Publish Event Use-case Description

Use-case: Publish Event **UCID: 18**	
Primary Actor	- Event Admin
Stakeholders and Interest	
Brief Description	- Enable Event Admin to publish events that are validated with the guidelines
Pre-condition	- An event has been created

Trigger	- Event Admin clicks the publish button next to an event
Relationship	- Publish Event <<*association*>> Event Admin - Publish Event <<*include* >> Create Event
Flow of Event	1. Event Organizer navigates to the website 2. Event Organizer logs into their account 3. Event Organizer clicks the publish button next to a created event

Table 2.19. Update Business Information Use-case Description

Use-case: Update Business Information UCID: 19	
Primary Actor	- Event Organizer
Stakeholders and Interest	
Brief Description	- Enable Event Organizer to update their business information
Pre-condition	- Event Organizer needs to have created an account
Trigger	- Event Organizer change their information and click on “save information”
Relationship	- Update Business Information <<*association*>> Event Organizer
Flow of Event	1. Event Organizer navigates to the website 2. Event Organizer enters their information 3. Event Organizer clicks login

	4. Event Organizer change their business information 5. Event Organizer save their new business information
Exceptional Flow	1. Event Organizer navigates to the website 2. Event Organizer enters their information 3. Event Organizer clicks login 4. Event Organizer change their business information 5. If the new information of Event Organizer is invalid an error will be displayed

Table 2.20. Manage Ticket Use-case Description

Use-case: Manage Ticket UCID: 20	
Primary Actor	- Event Organizer
Stakeholders and Interest	
Brief Description	- Enable Event Organizer to manage tickets for the event (create, update, delete and block tickets)
Pre-condition	- Event Organizer needs to be logged in - Events already created
Trigger	- Event Organizer clicks on “manage ticket” dashboard of an event
Relationship	- Manage Ticket <<*association*>> Event Organizer - Manage Ticket <<*generalization*>> Update Ticket

	- Manage Ticket << *generalization*>> Create Ticket - Manage Ticket << *generalization*>> Delete Ticket - Manage Ticket << *generalization*>> Block Ticket
Flow of Event	1. Event Organizer navigates to the website 2. Event Organizer logs into their account 3. Event Organizer clicks “manage event” dashboard 4. Event Organizer selects an event 5. Event Organizer click “manage ticket” for the selected event

2.3 System database diagram

2.3.1 Ticket Management System Entity-Relationship Diagram

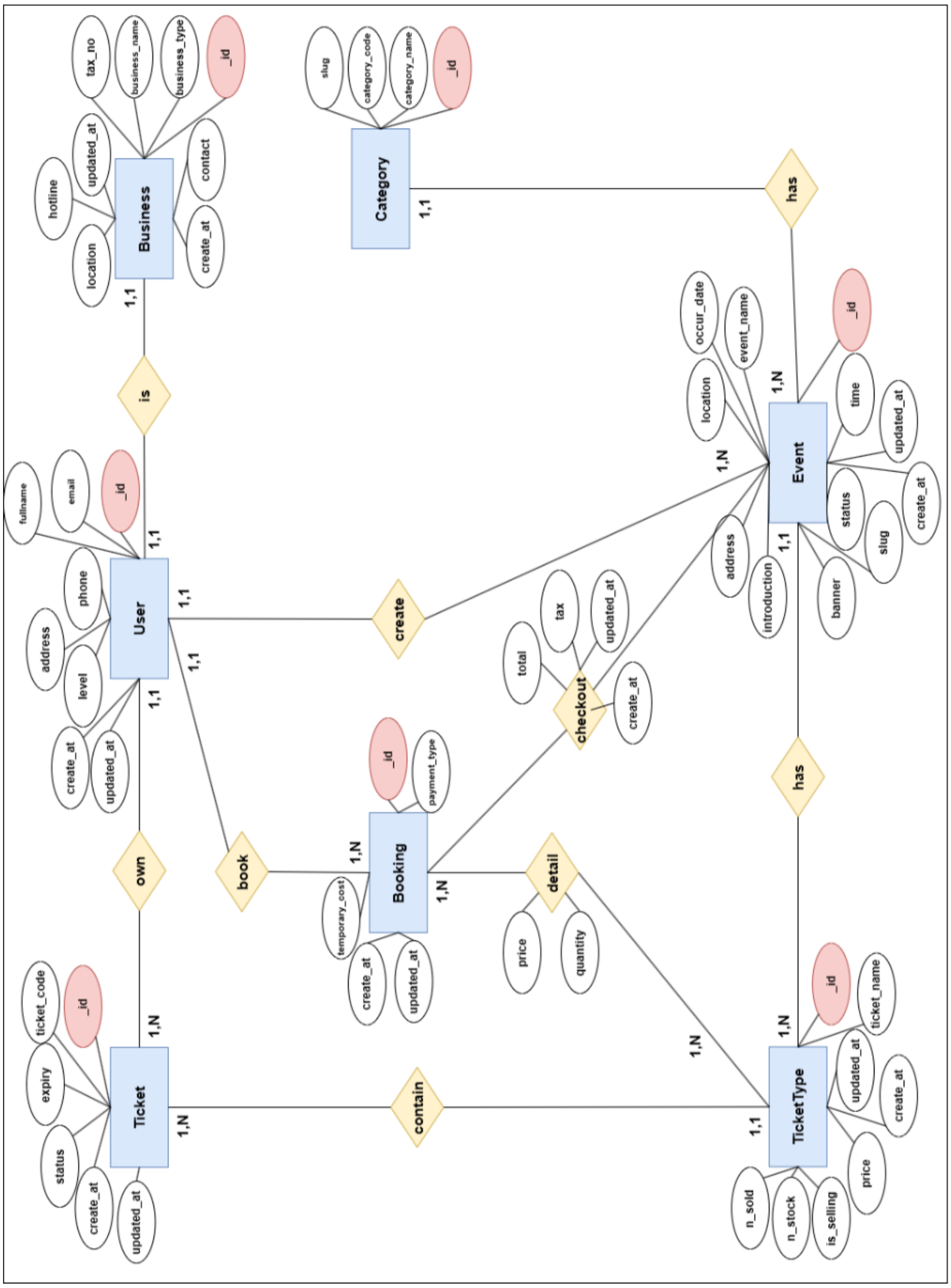


Figure 2.5. Ticket Management System Entity-Relationship Diagram.

Ticket Management Entity-Relationship Diagram has 7 entities:

- User (**_id**, email, fullname, phone, address, level, create_at, updated_at).
- Ticket (**_id**, ticket_code, expiry, status, create_at, updated_at).
- Business (**_id**, business_type, business_name, tax_no, contact_hotline, location, create_at, updated_at).
- Category (**_id**, slug, category_code, category_name).
- Booking (**_id**, payment_type, temporary_cost, create_at, updated_at).
- TicketType (**_id**, ticket_name, n_sold, n_stock, is_selling, price, create_at, updated_at).
- Event (**_id**, event_name, occur_date, location, address, introduction, banner, status, slug, time, create_at, updated_at).

Ticket Management ERD relationship between the entities:

- A User can own 1 or multiple Ticket(s). A User is 1 Business. A User can book 1 or multiple Booking(s). A User can create 1 or multiple Event(s).
- An Event can be checked out by 1 or multiple Booking(s). An Event can have 1 or multiple TicketType(s). An Event can be created by 1 User. An Event can be in 1 Category.
- A TicketType contains 1 or multiple Ticket(s). A TicketType can be detailed 1 or multiple Booking(s). A TicketType has 1 Event.
- A Booking can detail 1 or multiple TicketType(s). A Booking can checkout 1 or multiple Event(s).
- A Ticket can be owned by 1 User. A Ticket contains 1 TicketType.
- A Category has 1 or multiple Event(s).
- A Booking can be booked by 1 User.
- A Business is 1 User.

2.3.2 Ticket Management System Physical Database Diagram

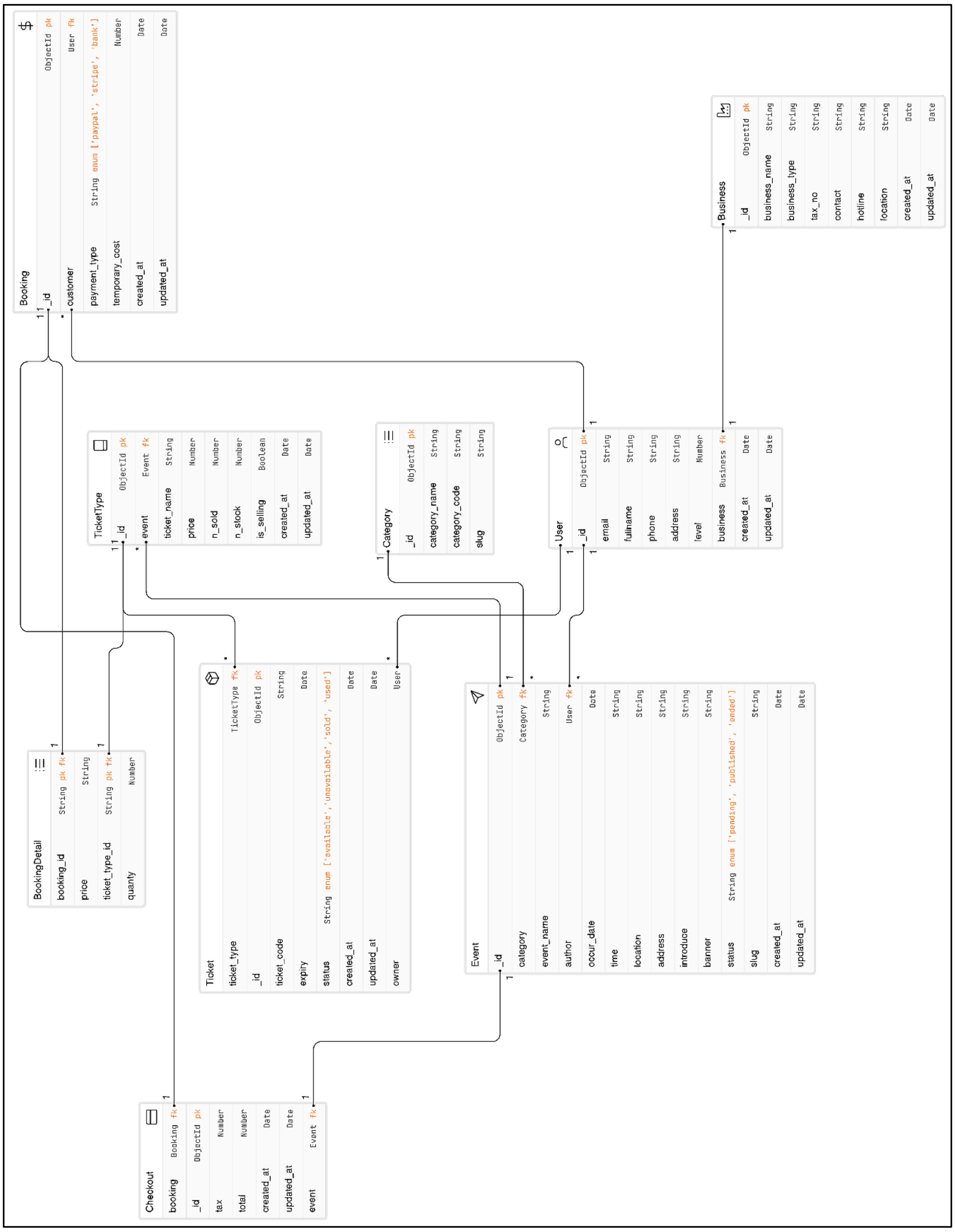


Figure 2.6. Ticket Manage Physical Database Diagram

The physical database diagram is a visual representation of the structure, relationships and entities within the database. We will list the entities and their attributes that define the Ticket Management System physical database diagram.

Checkout properties:

- _id (String): Unique and random generated id.
- booking (Booking): Reference to Booking’s id.
- tax (Number): Number to tax.
- total (Number): Total price after tax.
- Event (Event): Reference to Event’s id.
- created_at (Date): Timestamp when the Checkout was created.
- updated_at (Date): Timestamp when the Checkout information was updated.

BookingDetail properties:

- ticket_type_id (TicketType): Reference to TicketType’s id.
- booking_id (Booking): Reference to Booking’s id.
- price (String): Total price of ticket after booking.
- quantity (Number): Total number of tickets after booking.

Ticket properties:

- _id (String): Unique and random generated id.
- ticket_type (TickeType): Reference to TicketType’s id.
- ticket_code (String): Ticket code.
- owner (User): Reference to User’s id.
- expiry (Date): Ticket expiration date.
- status (String): Ticket status, accepted status available, unavailable, sold and used.
- created_at (Date): Timestamp when the Ticket was created.
- updated_at (Date): Timestamp when the Ticket information was updated.

Event properties:

- _id (String): Unique and random generated id.
- category (Category): Reference to Category’s id.
- event_name (String): Event name.
- author (User): Reference to User.
- occur_date (Date): Occur date of the event.
- time (String): Time that the event occurs.
- location (String): Seat or section within the Event.
- address (String): Event address.
- introduce (String): Event introduction.
- banner (String): Event banner image link.
- status (String): Event status, accept 3 states being pending, published and ended.
- slug (String): Event “slug” link.
- created_at (Date): Timestamp when the Event was created.
- updated_at (Date): Timestamp when the Event information was updated.

TicketType properties:

- _id (String): Unique and random generated id.
- event (Event): Reference to Event’s id.
- ticket_name (String): User fullname.
- phone (String): User phone number.
- address (String): User address.
- level (Number): User account level used for when upgrading.
- business (Business): Reference to Business’s id, this property can be null.
- created_at (Date): Timestamp when the TicketType was created.
- updated_at (Date): Timestamp when the TicketType information was updated.

Category properties:

- _id (String): Unique and random generated id.
- category_name (String): Name of the category.
- category_code (String): Unique code of the category.
- slug (String): Category's slug.

User properties:

- _id (String): Unique and random generated id.
- email (String): User email.
- fullname (String): User fullname.
- phone (String): User phone number.
- address (String): User address.
- level (Number): User account level used for when upgrading.
- business (Business): Reference to Business's id, this property can be null.
- created_at (Date): Timestamp when the User was created.
- updated_at (Date): Timestamp when the User information was updated.

Booking properties:

- _id (String): Unique and random generated id.
- customer (User): Reference to Customer's id.
- payment_type (String): Type of a payment, accept 3 types of payment paypal, stripe and bank.
- temporary_cost (String): Temporary cost of the total booking.
- created_at (Date): Timestamp when the Booking was created.
- updated_at (Date): Timestamp when the Booking information was updated.

Business properties:

- _id (String): Unique and random generated id.
- business_name (String): Business name.
- business_type (String): Type of a business.
- tax_no (String): Business tax number.

- contact (String): Contact information of the business.
- hotline (String): Business hotline.
- location (String): Business location.
- created_at (Date): The date business was created.
- updated_at (Date): Timestamp when the business information was updated.

2.4 Ticket Management System Sequence Diagram

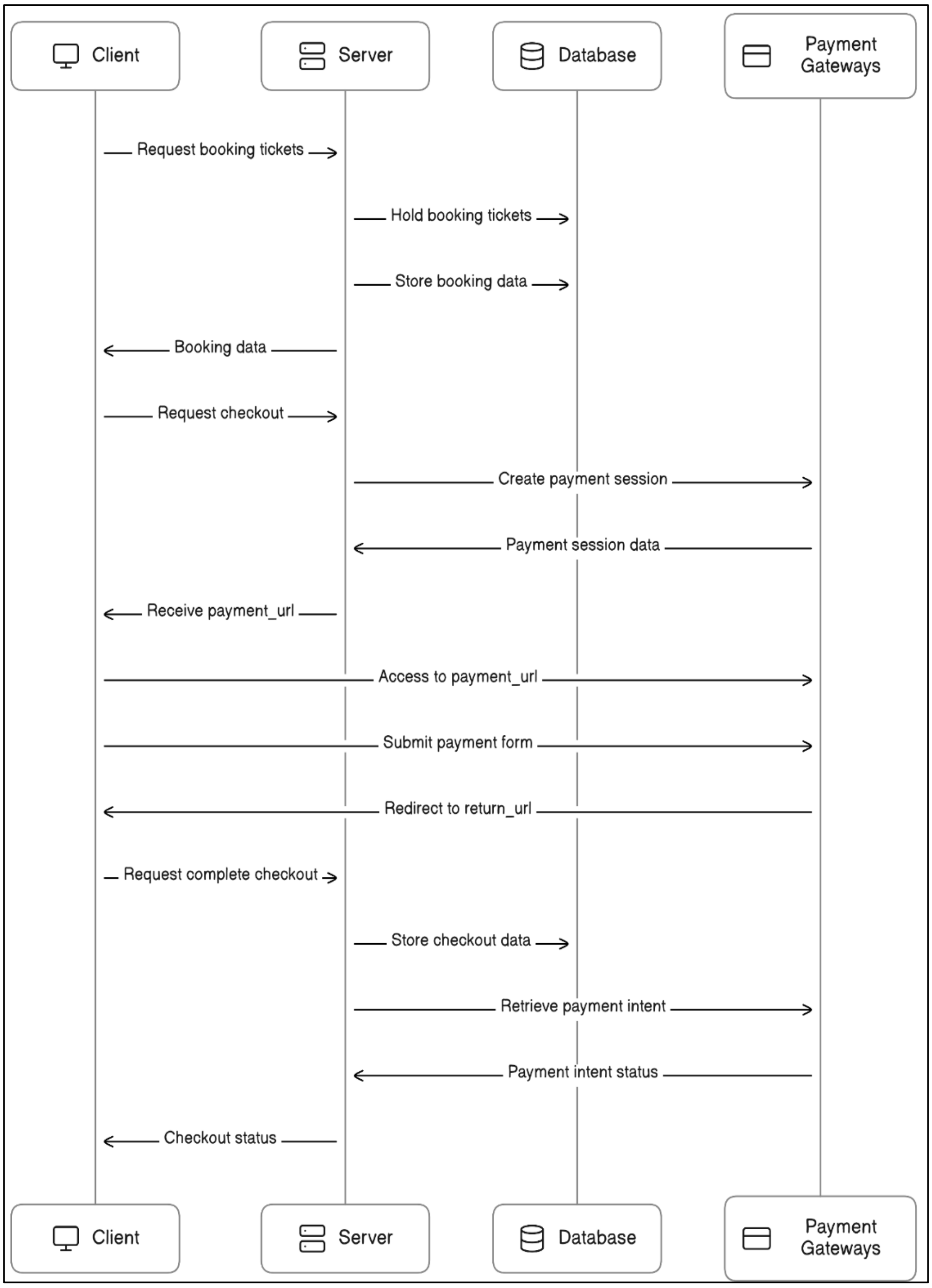


Figure 2.7. Ticket Management System “Checkout Process” Sequence Diagram.

Sequence diagram of the “Checkout Process” includes 4 participants:

- Client: The interface for the users.
- Server: The project server, handling requests and logic.
- Database: Storing data in JSON format.
- Payment Gateway: A gateway service handling payment processes for Paypal and Stripe.

Sequence diagram of the “Checkout Process” description:

1. Client requests booking ticket to the server.
2. Server “hold” booking tickets before saving to database
3. Server sends a request to store booking data to database.
4. Server sends booking data to client.
5. Client requests checkout to the server.
6. Server request to create a payment session to payment gateways.
7. Payment gateways returns a payment session data to server.
8. Server returns payment url to client.
9. Client accesses payment url which was created to payment gateway and provided by the server.
10. Client submits payment form using the supplied the payment url.
11. Payment gateway redirects to return payment url to client.
12. Client requests complete checkout to server.
13. Server requests to store checkout data in database.
14. Server request to retrieve payment data from payment gateway.
15. Payment gateway returns payment status to server.
16. Server returns checkout status to client.

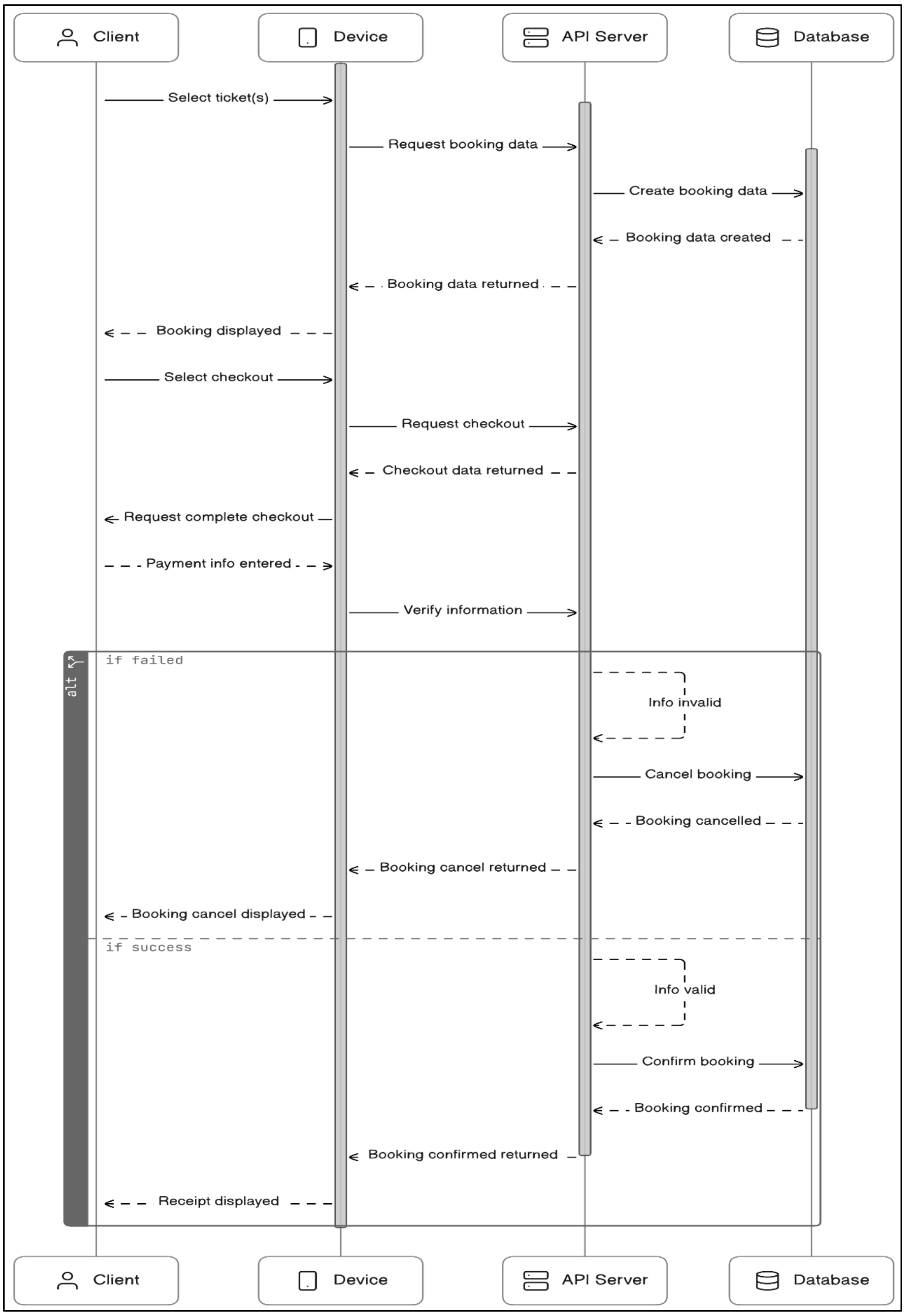


Figure 2.8. Ticket Management System “Booking Ticket” Sequence Diagram.

Sequence diagram of the “Booking Ticket” includes 4 participants:

- Client: Users of the system.
- Device: Interface for the users to interact with the system.
- API Server: The project server, handling requests and logic.
- Database: Storing data in JSON format.

Sequence diagram for “Booking Ticket” includes 1 alternative process section.

Sequence diagram of the “Booking Ticket” description:

1. Client selects ticket(s) and send request to device.
2. Device sends booking request data to API server.
3. API server sends request to save booking data to database.
4. Database replies created booking data to API server.
5. API replies booking data to Device.
6. Device displays booking data to client.
7. Client requests checkout to device.
8. Device requests checkout to API server.
9. API server replies with checkout data to device.
10. Device requests client input payment info.
11. Client enters payment info to device.
12. Device requests to verify payment info to API server.
13. If the “Verify information” process failed:
    a. API server replies info is invalid.
    b. API requests cancel booking to database.
    c. Database replies booking cancelled to API server.
    d. API server replies booking cancelled to device.
    e. Device displays the booking cancelled message.
14. If the “Verify information” process succeeded:
    a. API server replies info is valid.
    b. API requests confirm booking data to database.
    c. Database replies booking is confirmed to API server.

d. API server replies booking confirmation to device.
e. Device displays the booking receipt message.

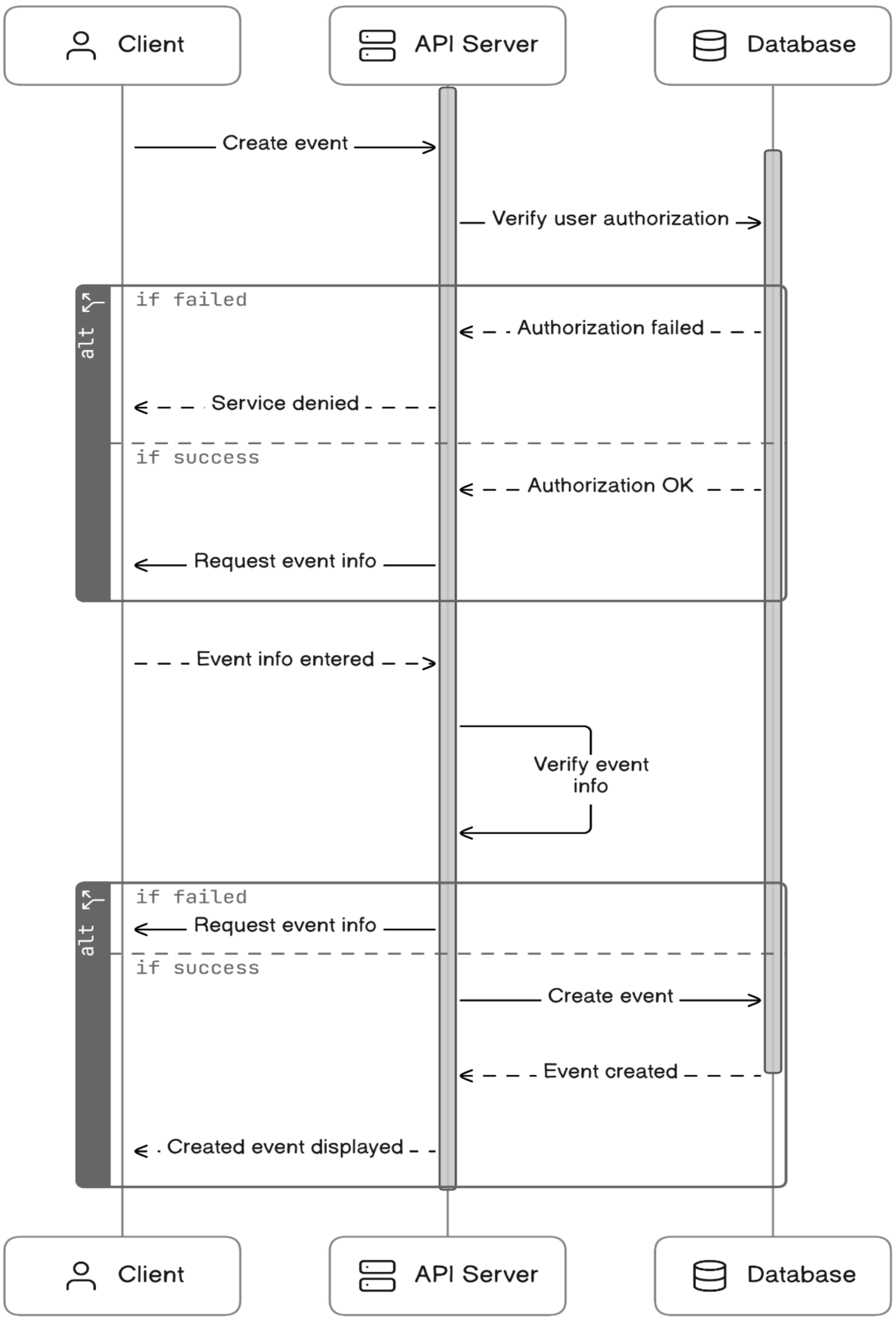


Figure 2.9. Ticket Management System "Create Event" Sequence Diagram.

Sequence diagram of the “Create Event” includes 3 participants:

- Client: Interface for the users to interact with the system.
- API Server: The project server, handling requests and logic.
- Database: Storing data in JSON format.

Sequence diagram for “Create Event” includes 2 alternative process sections.

Sequence diagram of the “Create Event” description:

1. Client request create event to API server.
2. API server request verify user authorization from database.
3. If “User Authorization” process failed:
    a. Database replies authorization failed to API server.
    b. API server replies service failed to client.
4. If “User Authorization” process succeeded:
    a. Database replies authorization OK to API server.
    b. API server request event info from the client.
5. Client replies event info to API server.
6. API server verify event information supplied by the client.
7. If “Event Info Validation” process failed:
    a. API server request event info from client (again).
8. If “Event Info Validation” process succeeded:
    a. API server requests create event to database.
    b. Database replies event created to API server.
    c. API server replies event created message to client.

CHAPTER 3. SYSTEM IMPLEMENTATION

3.1 Technologies

In this section, we will outline technologies our team used in the Ticket Management System:

- NoSQL and MongoDB:
    - Reasons: Our data's properties are complex because there are a lot of properties that have different datatype and constraint; and they keep changing during development, this is why NoSQL is chosen for the flexibility in database and schemas design and its ability to handle volumes of data and fields at high speed. MongoDB is chosen because it is the most used database for NoSQL, MongoDB is easy to setup, connect and use, it has a GUI to visualize the database.
    - Pros: Use human-readable a JSON format to store data. Supports complex queries and can handle large datasets. Has multiple libraries to support such as Sequelize, MongooseJS and MongoDB node packages.
    - Cons: Processing data after queries to render or handle logic can be slow.
- ReactJS:
    - Reasons: We want to use a unify programming language which is Javascript, a front-end that looks modern and can be maintainable, supports custom made and reusable components for a quicker development process, supports for server-side rendering and single-page application for a quick and smooth experience, ReactJS meet our need.
    - Pros: Supports different and modern libraries and UI libraries. Can reuse components for time saving development process.

    - Cons: Build time can take longer to complete. More complex directory structure and learning curves.
- NodeJS and ExpressJS:
    - Reasons: We want to use a single programming language which is Javascript to develop the front-end and back-end. NodeJS has multiple nope packages (add-ons/middlewares) to supports development and large community to for support.
    - Pros: NodeJS and Express are lightweight and suitable for developing API services. Has multiple node packages to support for development.
    - Cons: Not suitable for tasks requiring multi-threaded processing. Build time is range from normal to not fast.
- Postman:
    - Reasons: We want to test our API services and create examples with those tests. Furthermore, postman support the creation deployment of API documentation.
    - Pros: Postman can be used using the application or on the website. It is simple to use and have an option for creating an API documentation.
    - Cons: Postman has limited access when it is offline. API documentation has an old format, simple and plain.
- SUD-Libs[7]:
    - Reasons: The email and payment process that npm provided have unnecessary functions which made it large, we want to have our own process and functions at the lighter speed, so we developed and deployed our own node package to help with development.
    - Pros: It is lightweight, can be easily imported to different projects.

[7] https://www.npmjs.com/package/sud-libs

  - Cons: Support only a few specific functionalities. Works only with our process and development. Need regular updates for bug fixes.

- Tailwind CSS:
  - Reasons: We use Tailwind CSS because it supplies structure template like Bootstrap but it is more bare-bone which help to have more control with CSS components and faster development; and it offers modern looks and UI, which is needed in our project development.
  - Pros: Provide template structure like Bootstrap, which help with faster development. Support responsive designs and large community for support.
  - Cons: Tailwind CSS is new and there is a learning curve. Tailwind CSS file size is large compared to normal hand-made CSS. Need to have a good understanding of CSS components.
- JSON Web Token (JWT):
  - Reasons: JWT encryption user verification token and provide a method to verify these tokens, meaning verify the user; and we are using this method to verify user accounts without storing information on the server.
  - Pros: JWT is compact and lightweight (there is a node package that support JWT) while providing a "stateless" process and security.
  - Cons: JWT size might be large to be contained. Need to configure JWT Expiry date because JWT is public and with enough time it might be able to be decrypted.

3.2 System architecture

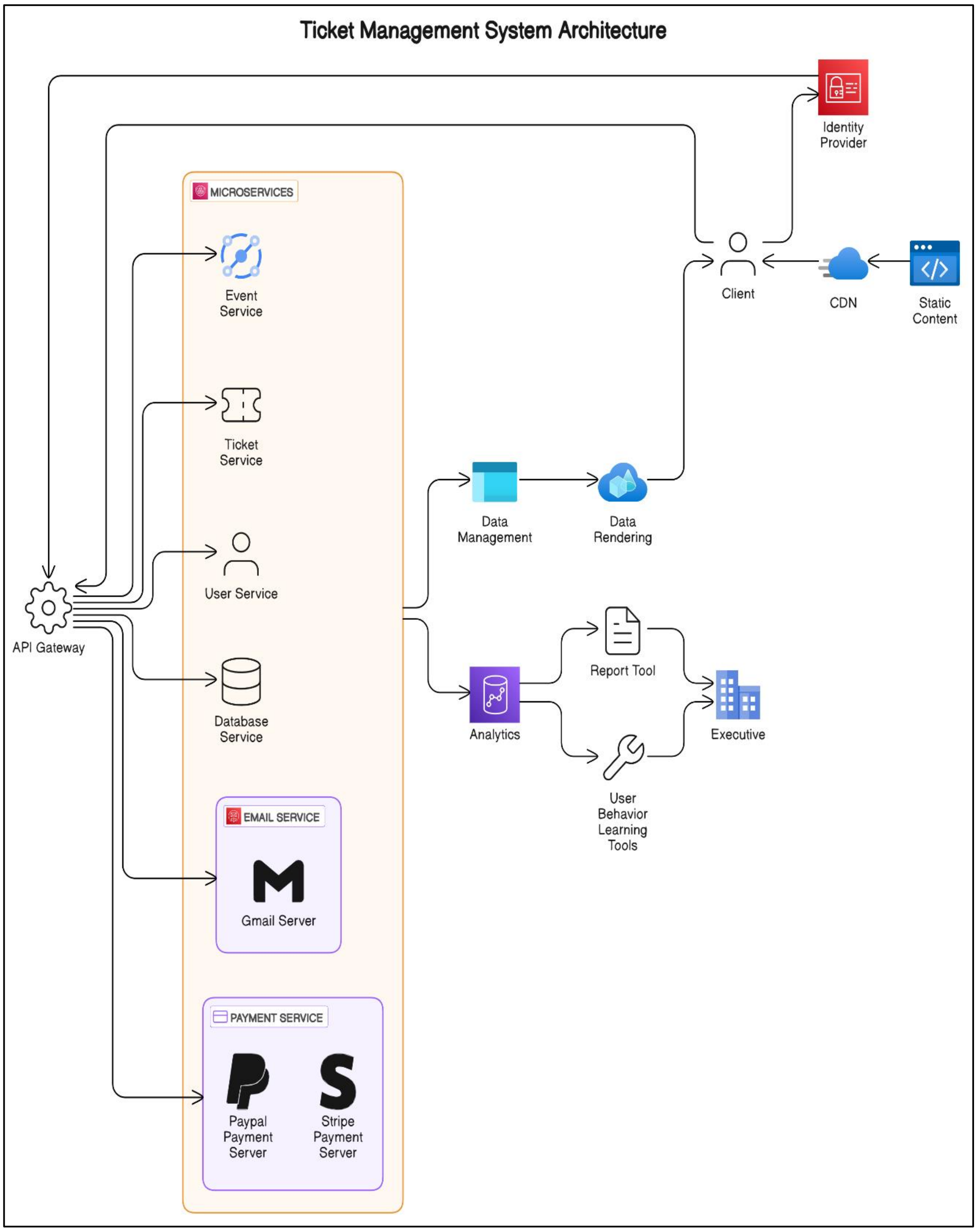


Figure 3.1. Ticket Management System Architecture.

Components presented in Ticket Management System Architecture:

- Client: They are the clients that use and create data while using the platform.
- Executive: They are executives that view the data created by the clients to make necessary change toward the platform’s business.
- Identity Provider: It is the identity provider that the clients need to have in order to use the platform.
- CDN and Static Content: They are the website CDN and contents that are static; and they are served to the clients.
- API Gateway: It is a gateway/hub that take in requests and replies with the correspond data to the clients and executives.
- Microservices: They are a collection of smaller services that is used to handle requests, system logic and processes which includes:
  - Event Service: Handling event related requests.
  - Ticket Service: Handling ticket related requests.
  - User Service: Handling user related requests.
  - Database Service: Handling database related process.
  - Email Service: Handling email related requests.
  - Payment Service: Handling payment related requests for Paypal and Stripe.
- Data Management: It is used to manage data returned by the microservices
- Data Rendering: It is used to render data from Data Management and returns to the clients.
- Analytics: It is the raw process and transactions data made by the clients.
- Report Tool: Create a report based on the raw data from the Analytics.

- User Behavior Learning Tools: Run the raw data from Analytics to through some model to have comparison and predict clients purchase pattern.
- Executives: Read the report and comparison of user purchase patterns to make business decision changes.

3.3 System functions implementation

3.3.1 Create and connect to MongoDB using MongooseJS

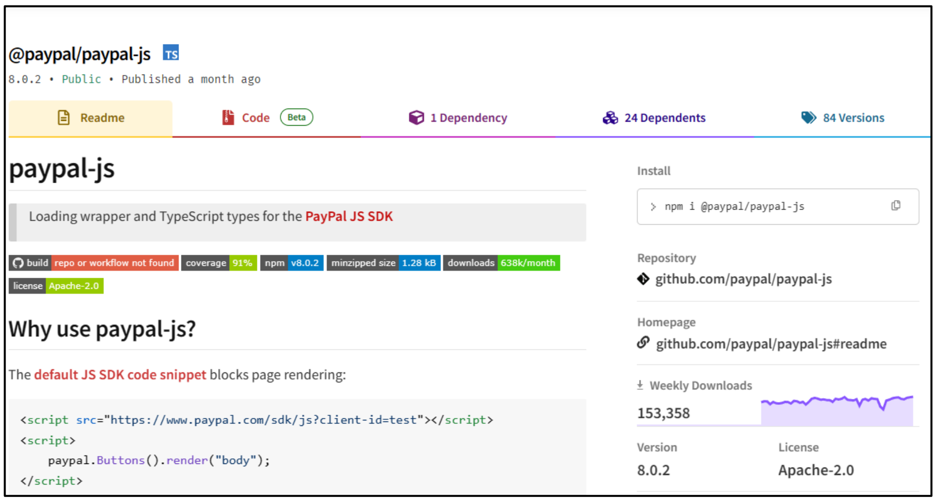
We use MongooseJS which is a node package from npm to helps with the connection from NodeJS and ExpressJS to MongoDB. MongooseJS provides a simple way to connect by using `.connect()` function to create a connection pool to MongoDB. We also use `.lean()` which is Plain Old JavaScript Object (POJO) techniques to simplify the data returned by Mongoose because the data returned by MongooseJS is in Mongoose document format and has a lot of unneeded data. After we have created a connection to MongoDB using MongooseJS, we create the schemas for each or our objects:

- The Booking Model is used to store Bookings created by Users and Event Organizer.
- The Event Model is used by Event Organizer to create events and used by Registered Customer and Unregistered Customer to view events.
- The Ticket Model is used to create tickets for events. Each ticket has these statuses: available, unavailable, pending, sold and used, if the status does not match with the pre-defined statuses, it will be saved with the default status.
- The TicketType Model is used to store the ticket type that is bound to each event.
- The OTP Model is used to create validation for accounts and ticket transactions.

3.3.2 Payment gateway

For the payment gateway we use 2 payment methods being Paypal and Stripe. Because we develop our project using NodeJS and ExpressJS as our back-end server we have node packages that help us with configuration and handling requests.

You need to create a Paypal and Stripe developer account and get the API from both. Configure Stripe and Paypal for transactions. The API keys are saved in `.env` file which can be configured in separate environment. After that, you can download the node packages supporting Paypal and Stripe.


@paypal/paypal-js TS
8.0.2 • Public • Published a month ago

paypal-js

Loading wrapper and TypeScript types for the PayPal JS SDK

Why use paypal-js?

The default JS SDK code snippet blocks page rendering:

```
<script src="https://www.paypal.com/sdk/js?client-id=test"></script>
<script>
    paypal.Buttons().render("body");
</script>
```

Install
```
> npm i @paypal/paypal-js
```

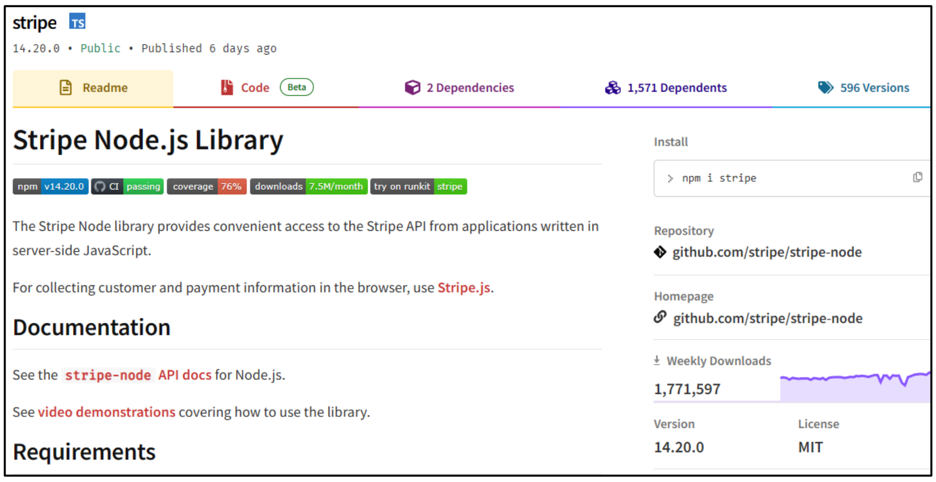
Repository
github.com/paypal/paypal-js

Homepage
github.com/paypal/paypal-js#readme

Weekly Downloads
153,358

Version	License
8.0.2	Apache-2.0

Figure 3.2. Paypal node package.


stripe TS
14.20.0 • Public • Published 6 days ago

Stripe Node.js Library

The Stripe Node library provides convenient access to the Stripe API from applications written in server-side JavaScript.

For collecting customer and payment information in the browser, use Stripe.js.

Documentation

See the `stripe-node` API docs for Node.js.

See video demonstrations covering how to use the library.

Requirements

Install
```
> npm i stripe
```

Repository
github.com/stripe/stripe-node

Homepage
github.com/stripe/stripe-node

Weekly Downloads
1,771,597

Version	License
14.20.0	MIT

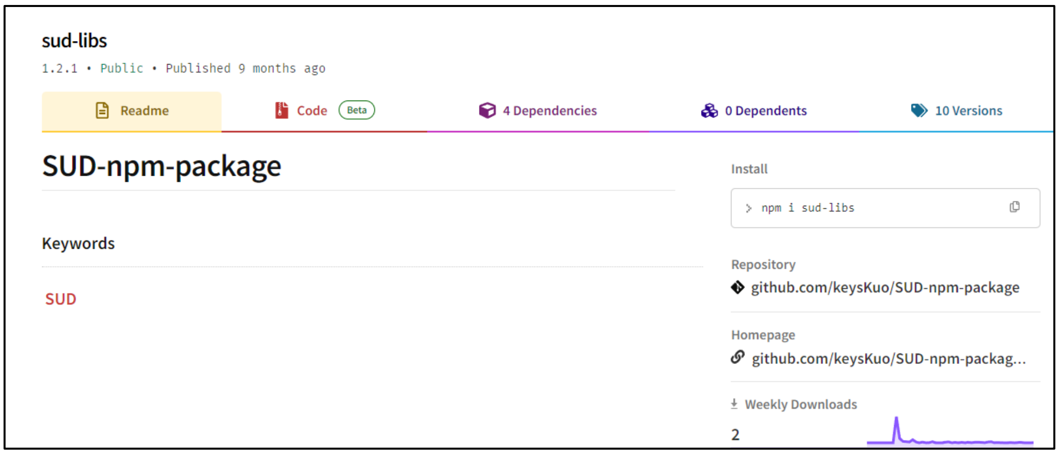
Figure 3.3. Stripe node package.

After integration, both Paypal and Stripe use a sandbox to demonstrate the purchase process and redirect the user to the sand box website.

3.3.3 Email for transactions and encryption

We implement encrypted for our user data process by using CryptoJS and One-Time Password (OTP) which is sent to user via their validated email address. We have developed and deployed a node package to handle encryption and email process because other node packages like nodemailer[8] can handle emailing process but it has many unnecessary functions and not lightweight. Our deployed node package helps to contain our process while providing the same methodology.

Because “sud-libs” is a node package, we can use `npm i sud-libs` to download and import it in the project.



sud-libs
1.2.1 • Public • Published 9 months ago

Readme | Code Beta | 4 Dependencies | 0 Dependents | 10 Versions

SUD-npm-package

Keywords

SUD

Install
```
> npm i sud-libs
```
Repository
github.com/keysKuo/SUD-npm-package

Homepage
github.com/keysKuo/SUD-npm-packag...

Weekly Downloads
2

Figure 3.4. Our deployed “sud-libs” npm package.

3.3.4 Event, Ticket and Booking Management

We divided Event, Ticket and Booking into separate routes to handle which are `/api/event`, `/api/ticket` and `/api/booking`. They contain processes to handle requests sent from the front-end and return the processed data through an API gateway back to the front-end to be rendered for the customers.

Each Event, Ticket and Booking have a separate CRUD operation to handle POST, GET, PUT AND DELETE requests. The Booking route has middlewares that

[8] https://www.npmjs.com/package/nodemailer

intercept the requests to validate information within the process before making to the Booking route.

3.3.5 Using Python to analyze user purchasing behavior

We use Python, Google Collab and sklearn library to handle customers' purchase data and compare user purchasing pattern between the Random Forrest Model, XGBoost Classification Model and Decision Tree Model (Sundari, 2020).

We need to first import our data, the library and the library functions that we use to read the data:

```
#importing basic packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#Loading the data
data0 = pd.read_csv('./drive/MyDrive/data3.csv')
data0.head()
#checking the data for null or missing values
data.isnull().sum()
# shuffling the rows in the dataset
data = data.sample(frac=1).reset_index(drop=True)
data.head()
# Sepratating & assigning features and target columns to X & y
y = data['Label']
X = data.drop('Label',axis=1)
X.shape, y.shape
# Splitting the dataset into train and test sets: 80-20 split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = 12)
X_train.shape, X_test.shape
#importing packages
from sklearn.metrics import accuracy_score
# Creating holders to store the model performance results
ML_Model = []
acc_train = []
acc_test = []
#function to call for storing the results
def storeResults(model, a,b):
  ML_Model.append(model)
  acc_train.append(round(a, 3))
  acc_test.append(round(b, 3))
```

After the data has been read, we will train the data with the Random Forrest Model:

```
# Random Forest model
from sklearn.ensemble import RandomForestClassifier
# instantiate the model
forest = RandomForestClassifier(max_depth=5)
# fit the model
forest.fit(X_train, y_train)
#predicting the target value from the model for the samples
y_test_forest = forest.predict(X_test)
y_train_forest = forest.predict(X_train)
#computing the accuracy of the model performance
acc_train_forest = accuracy_score(y_train,y_train_forest)
acc_test_forest = accuracy_score(y_test,y_test_forest)

print("Random forest: Accuracy on training Data:
{:.3f}".format(acc_train_forest))
print("Random forest: Accuracy on test Data:
{:.3f}".format(acc_test_forest))
#storing the results. The below mentioned order of parameter passing is
important.
#Caution: Execute only once to avoid duplications.
storeResults('Random Forest', acc_train_forest, acc_test_forest)
```

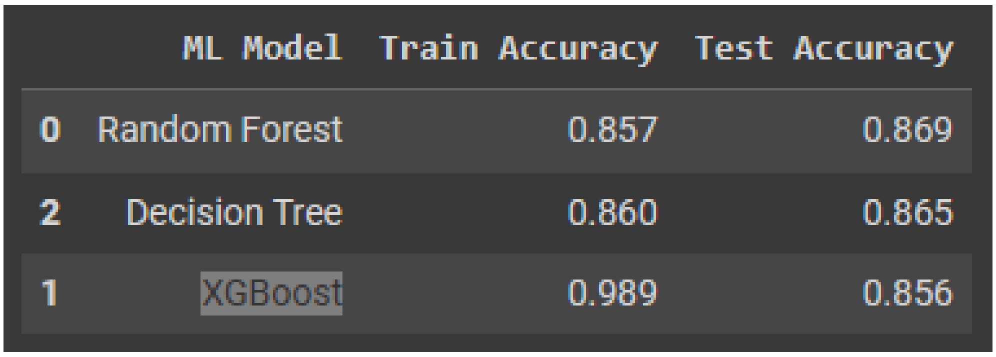
We do similar process with the XGBoost Classification Model:

```
#XGBoost Classification model
from xgboost import XGBClassifier
# instantiate the model
xgb = XGBClassifier(learning_rate=0.4,max_depth=7)
#fit the model
xgb.fit(X_train, y_train)
#predicting the target value from the model for the samples
y_test_xgb = xgb.predict(X_test)
y_train_xgb = xgb.predict(X_train)
#computing the accuracy of the model performance
acc_train_xgb = accuracy_score(y_train,y_train_xgb)
acc_test_xgb = accuracy_score(y_test,y_test_xgb)
print("XGBoost: Accuracy on training Data: {:.3f}".format(acc_train_xgb))
print("XGBoost : Accuracy on test Data: {:.3f}".format(acc_test_xgb))
#storing the results. The below mentioned order of parameter passing is
important.
#Caution: Execute only once to avoid duplications.
storeResults('XGBoost', acc_train_xgb, acc_test_xgb)
```

We do similar process with the final model, Decision Tree Model:

```
# Decision Tree model
from sklearn.tree import DecisionTreeClassifier
# instantiate the model
tree = DecisionTreeClassifier(max_depth = 5)
# fit the model
tree.fit(X_train, y_train)
#computing the accuracy of the model performance
acc_train_tree = accuracy_score(y_train,y_train_tree)
acc_test_tree = accuracy_score(y_test,y_test_tree)
print("Decision Tree: Accuracy on training Data:
{:.3f}".format(acc_train_tree))
print("Decision Tree: Accuracy on test Data:
{:.3f}".format(acc_test_tree))
storeResults('Decision Tree', acc_train_tree, acc_test_tree)
```

Comparing the models' accuracy:



	ML Model	Train Accuracy	Test Accuracy
0	Random Forest	0.857	0.869
2	Decision Tree	0.860	0.865
1	XGBoost	0.989	0.856

Figure 3.5. Random Forest, Decision Tree and XGBoost model accuracy.

Our goal of performing the training of Random Forrest Model, XGBoost Classification Model and Decision Tree Model with user purchase data is to help assisting in detection of users performing illegal purchases of tickets (anomaly) by creating a comparison between the models combining with the decision making of the executives based on the machine learning results.

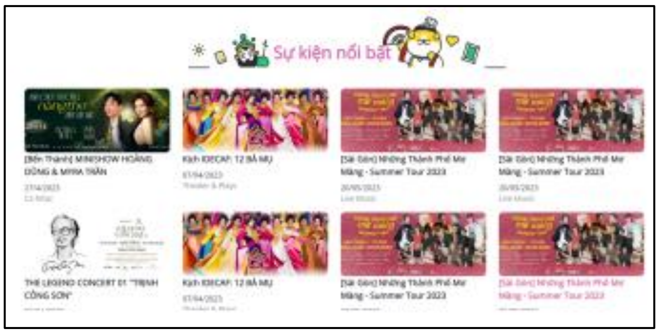
CHAPTER 4. SYSTEM DEMONSTRATION

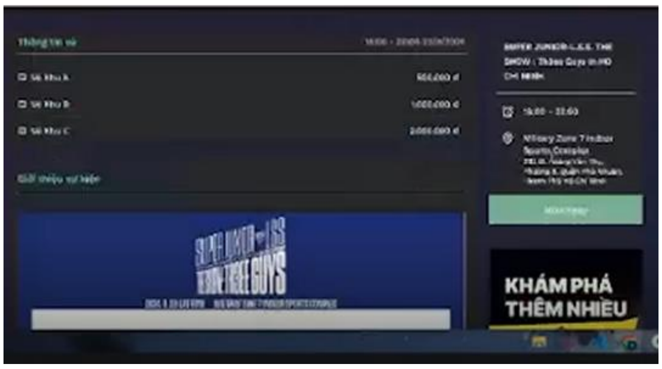
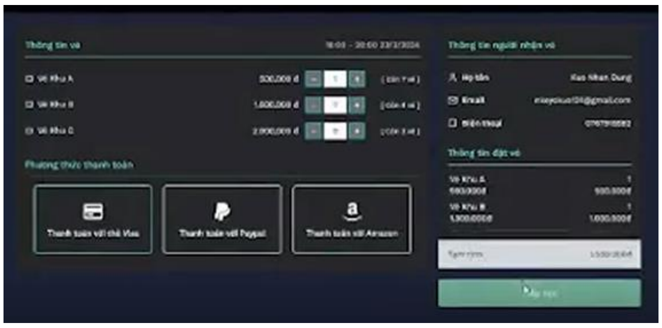
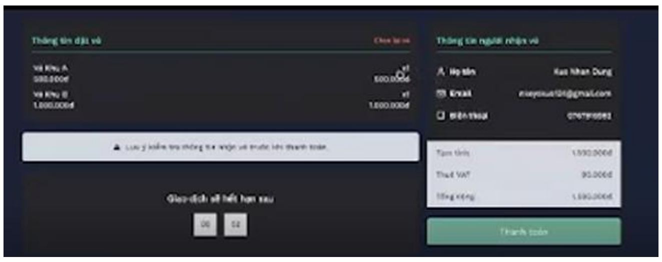
4.1 System main flows

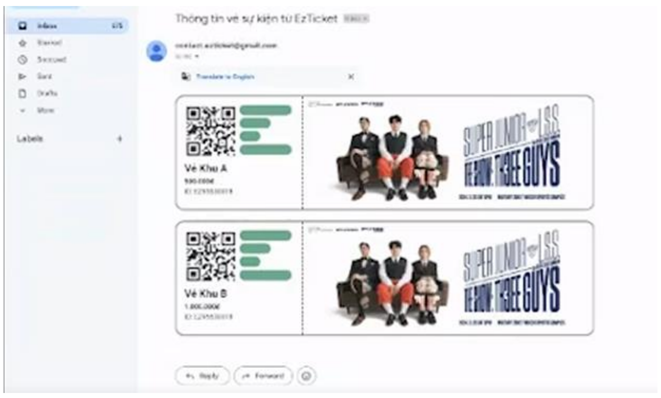
- Event creation and event flow management: Including CRUD functions of the event from the creation by the event organizers, creating tickets for the events and managing events visibility.
- Booking ticket: Include selection of tickets, booking process and transaction completion.
- Payment processing flow: Include interaction with payment gateways, transaction validation, and the confirmation of successful payments.
- Admin dashboard and reporting flow: Include interactions with the admin dashboard, analytics based on user purchase data the help of using machine learning.

4.2 System demonstration

Table 4.1. Booking ticket process.

Images	Description/Steps
 Sự kiện nổi bật [Bến Thành] MINISHOW HOÀNG DŨNG & MYRA TRẦN 27/04/2023 [illegible] Kịch IDECAF: 12 BÀ MỤ 07/04/2023 Theater & Plays [Sài Gòn] Những Thành Phố Mơ Màng - Summer Tour 2023 20/05/2023 Live Music [Sài Gòn] Những Thành Phố Mơ Màng - Summer Tour 2023 20/05/2023 Live Music THE LEGEND CONCERT 01: "TRỊNH CÔNG SƠN" Kịch IDECAF: 12 BÀ MỤ 07/04/2023 [illegible] [Sài Gòn] Những Thành Phố Mơ Màng - Summer Tour 2023 [Sài Gòn] Những Thành Phố Mơ Màng - Summer Tour 2023	**<u>Description:</u>** Event listed on EzTicket homepage. **<u>Steps:</u>** 1. User access EzTicket URL. 2. A list of events is displayed on the homepage of the website once a user accessed the website.

	**Description:** Event page with detailed information about the event (location, date/time, ticket types). **Steps:** 1. Event details page contain a list of details about the event. 2. Event banner, event location/time and ticket information are displayed for the customers.
 Thanh toán với thẻ Visa Thanh toán với Paypal Thanh toán với Amazon	**Description:** Choosing type of ticket, quantity of ticket and payment method. **Steps:** 1. Customer selects the type of ticket and the quantity of ticket. 2. Customer selects the payment method.
	**Description:** Waiting time for the customer to complete their transaction. **Steps:** 1. After customer has selected the ticket type, quantity and payment

	method; they will be redirect to a waiting page. Customer will have 10 minutes to complete the booking transaction. 2. If booking transaction is not completed after 10 minutes, process will be cancelled.
 Thông tin vé sự kiện từ EzTicket Inbox Starred Snoozed Sent Drafts More Labels Translate to English Vé Khu A Vé Khu B SUPER JUNIOR-L.S.S. THE SHOW: TH3EE GUYS Reply Forward	**<u>Description:</u>** Once the booking transaction is completed, the ticket will be sent to the customer verified email. **<u>Steps:</u>** 1. Customer completed the booking transactions. Event ticket(s) will be sent to the user verified email. 2. Customer can view the ticket(s) with event information in their verified email.

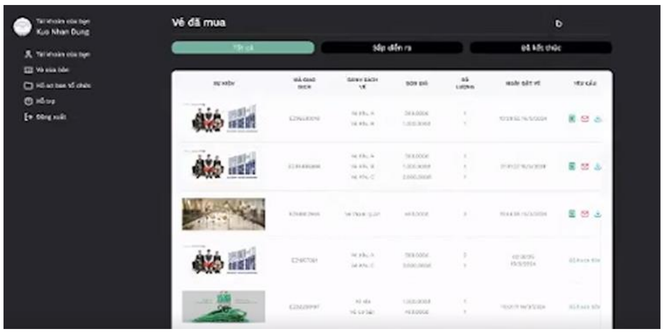
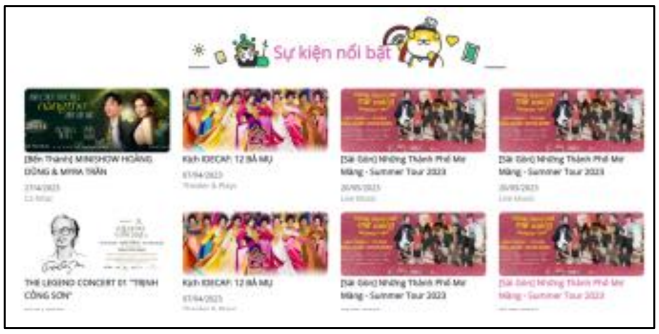
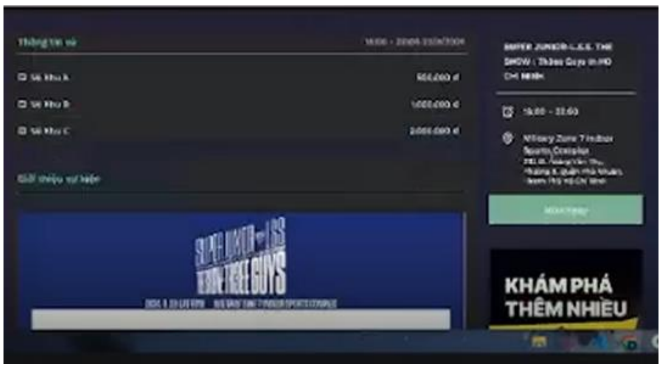
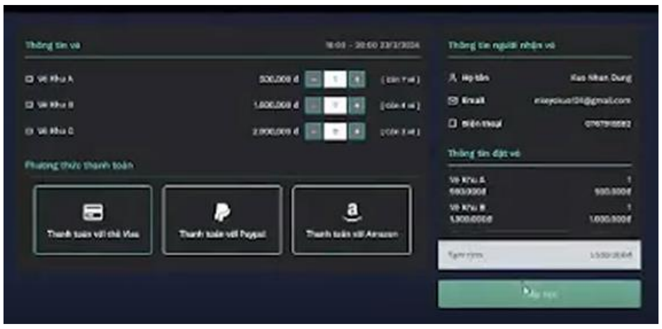
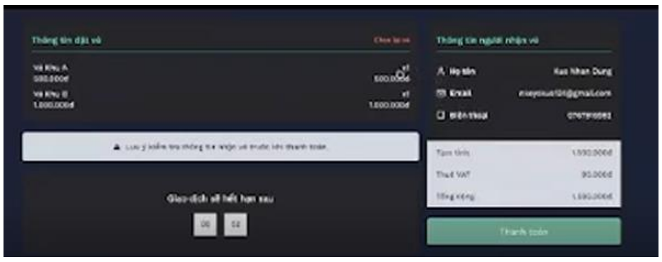
 Tài khoản của bạn Kuo Nhan Dung Tài khoản của bạn Vé của bạn Hồ sơ ban tổ chức Hỗ trợ Đăng xuất Vé đã mua Tất cả Sắp diễn ra Đã kết thúc	**<u>Description:</u>** Customer's booking history will be saved and listed in the customer's account. **<u>Steps:</u>** 1. Customer accesses their account. 2. Customer can view the booking history.

Table 4.2. Payment process flow.

Images	Description
	**<u>Description:</u>** Event listed on EzTicket homepage. **<u>Steps:</u>** 1. User access EzTicket URL. 2. A list of events is displayed on the homepage of the website once a user accessed the website.
	**<u>Description:</u>** Event page with detailed information about the event (location, date/time, ticket types). **<u>Steps:</u>** 1. Event details page contain a list of details about the event. 2. Event banner, event location/time and ticket information are displayed for the customers.

 Thông tin vé Vé Khu A Vé Khu B Vé Khu C Phương thức thanh toán Thanh toán với thẻ Visa Thanh toán với Paypal Thanh toán với Amazon Thông tin người nhận vé Họ tên Email Điện thoại Thông tin đặt vé	**Description:** Choosing type of ticket, quantity of ticket and payment method. **Steps:** 1. Customer selects the type of ticket and the quantity of ticket. 2. Customer selects the payment method.
 Thông tin đặt vé Vé Khu A Vé Khu B Thông tin người nhận vé Họ tên Email Điện thoại Giao dịch sẽ hết hạn sau Thuế VAT Thanh toán	**Description:** Waiting time for the customer to complete their transaction. **Steps:** 1. After customer has selected the ticket type, quantity and payment method; they will be redirect to a waiting page. Customer will have 10 minutes to complete the booking transaction. 2. If booking transaction is not completed after 10 minutes, process will be cancelled.

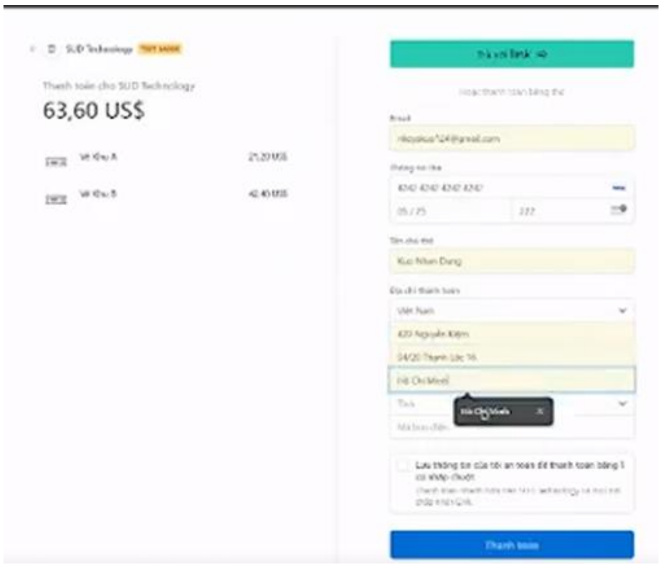
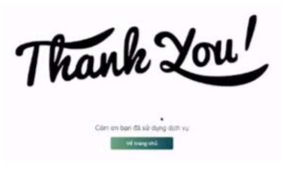
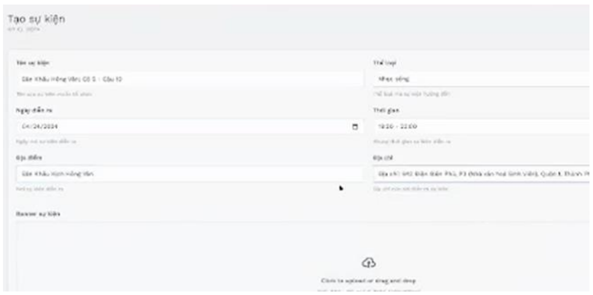
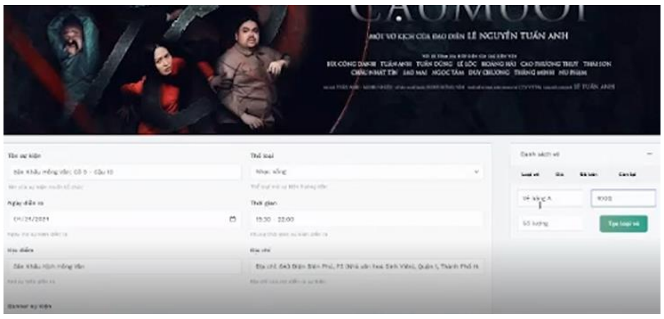
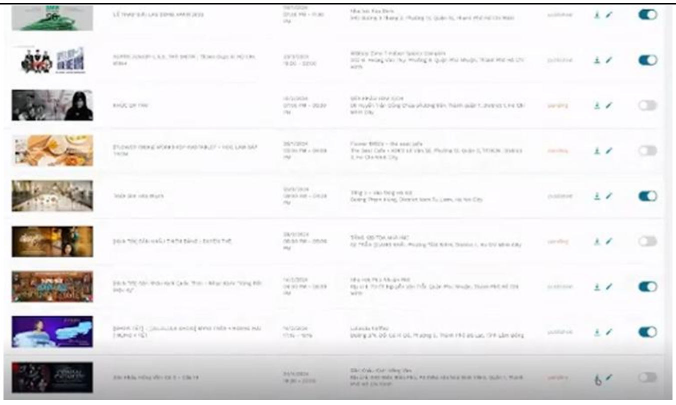
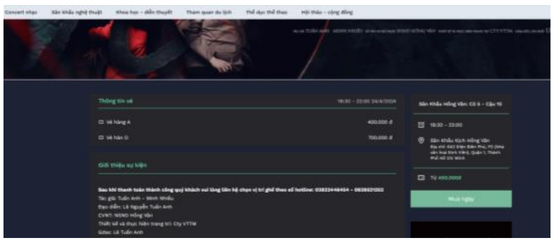
 SUD Technology Thanh toán cho SUD Technology 63,60 US$ 21,20 US$ 42,40 US$ 4242 4242 4242 4242 05 / 25 Việt Nam Thanh toán	**<u>Description:</u>** Paypal payment form for customer. **<u>Steps:</u>** 1. Customer will be redirected to a Paypal page with the amount of transaction they need to complete. 2. Customer fill out their payment information. 3. Customer click “complete” to complete the transaction.
	**<u>Description:</u>** Customer completed the Paypal payment process. **<u>Steps:</u>** 1. After payment is completed, customer will receive confirmation of payment. 2. Customer will be redirected back to EzTicket.

Table 4.3. Event creation and event flow management.

Images	Description
	**Description:** Event organizer “create event” page. **Steps:** 1. Event organizer login into their account and access the create event page. 2. Event organizer fill in event information and click “create event”.
	**Description:** Event organizer create tickets for the event. **Steps:** 1. Event organizer will be redirected to a preview of their event. 2. Event organizer create the ticket types for the event in this page.
	**Description:** Event Admin event management page. **Steps:** 1. Once the event is created, it will be displayed on the Event Admin page.

	2. Event Admin will view the page and decide whether to permit or denied the event.
 Concert nhạc Sân khấu nghệ thuật Khoa học - diễn thuyết Tham quan du lịch Thể dục thể thao Hội thảo - cộng đồng Thông tin vé 18:30 - 22:00 24/4/2024 Vé hạng A 400.000 đ Vé hạn D 700.000 đ Giới thiệu sự kiện Sân Khấu Hồng Vân: Cô 6 - Cậu 10 19:30 - 22:00 Sân Khấu Kịch Hồng Vân Từ 400.000đ Mua ngay Sau khi thanh toán thành công quý khách vui lòng liên hệ chọn vị trí ghế theo số hotline: [illegible] Tác giả: Tuấn Anh - Minh Nhiều Đạo diễn: Lê Nguyễn Tuấn Anh CVNT: NSND Hồng Vân Thiết kế và thực hiện trang trí: Cty VTTM Gdsx: Lê Tuấn Anh	**<u>Description:</u>** The published event homepage. **<u>Steps:</u>** 1. Once the event is verified by the Event Admin, it will be published to the website. 2. Customer and Event Organizer can view the published event on the website.

CHAPTER 5. SUMMARY AND FUTURE WORK

5.1 Summary

By finishing the development of the “Ticket Management Platform” project, we have completed some results:

- Researched ticket management business model and process.
- Developed a modern ticket management platform using MERN stack technology.
- Implementation of Paypal and Stripe payment method in the purchasing process.
- Create a modular architecture with separate front-end and back-end services, including API documentation for the back-end.
- Using user purchasing data to train and build a machine learning process that help to detect anomaly users.

Nonetheless, certain sections of the project are still in the state of developing:

- A complete understanding of the business model of ticket management system is still incomplete.
- Certain payment methods like Amazon are incomplete.
- Problems with connections when deploying to a public domain.
- More user data is needed to train the machine learning models.

With our project, we hope to contribute to theses aspects:

- The creation of a ticket management platform using MERN stack with the implementation of different payment methods such as Paypal and Stripe.
- Detecting bad actors who want to buy tickets to sell at a lower price by creating a tool to assist with staff decision making process based on user purchase data.

5.2 Future work

We proposed some additions to our “Ticket Management Project” that we would like to implement in the future:

1. Implement “low code” or “no code” customizable event page for event organizers: Because we want event organizers to have more control with their customization with or without programming knowledge.
2. Integration blockchain for ticket transaction: Because blockchain provides security, transparency and traceability; and we want our ticket purchases to be protected from fraudulent activity.
3. Multilingual support: Because we want to expanse our platform to more countries and different culture, reaching a global audience.

REFERENCES

English

Deshpande, C. (2023, 10 05). *The Best Guide to Know What Is React*. Retrieved 02 25, 2024, from Simplelearn: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs

Dev, R. (2024, 02 25). *React Reference Overview*. Retrieved 02 25, 2024, from React: https://react.dev/reference/react

Gollmann, D. (2011). Computer Security 3rd Edition. In D. Gollmann, *Computer Security 3rd Edition* (pp. 251-273). Wiley.

NodeJS. (2024, 02 25). *NodeJS Documentation*. Retrieved 02 25, 2024, from NodeJS Org: https://nodejs.org/docs/latest/api/

Postman. (2024, 02 25). *Postman Explore*. Retrieved 02 25, 2024, from Postman: https://www.postman.com/explore

Sundari, S. G. (2020, May 11). *GitHub*. Retrieved March 11, 2024, from Phishing-Website-Detection-by-Machine-Learning-Techniques: https://tinyurl.com/DetectionTechniques

Vernon, V. (2016). *Domain-Driven Design Distilled* (1st Edition ed.). O'Riley Media.

APPENDICES

Appendix A: Multiple Choice Questionnaires

1. How many events do you participate (on average) in a year?
    a. I don’t participate
    b. Less than 20 events
    c. More than 20 events
2. What type of event booking system do you often use?
    a. I don’t use one
    b. Book ticket offline/At a ticket stall
    c. Through phone calls/SMS
    d. Online platforms
3. What is the most common problem you face when purchasing tickets?
    a. Customer service
    b. Long queue
    c. Others
4. What are your expectations for an online movie booking and touch screen kiosk system?
    a. Easy navigation, easy-to-use system, user friendly layouts
    b. Useful functions
    c. Others
5. What age category do you fall in?
    a. Below 18 years old
    b. Over 18 years old
    c. Over 50 years old
6. What type of payment method do you often carry with or prefer using?
    a. Cash
    b. Bank transfer
    c. Credit card/Debi card

7. Do you currently in a job?
    a. Yes
    b. No

Appendix B: Interview Type Questionnaires

1. In your opinion, what makes a user-friendly and convenient ticketing system?
   - Easy navigation, easy to use.
2. Do you know any event ticketing system that operate like a marketplace?
   - Natively is Ticketbox, foreign places are EventZilla, TicketSpice,…
3. What are the advantages of booking tickets on ticketing system to offline ticket purchasing?
   - No queue or line-up
   - Easy to buy with online payments
   - Business profit: Users are approached by new events
4. What make you want to stay and use a ticketing system in the next times?
   - Continuously updating events
   - Responsive and cross-platform enabled
   - Convenient payment methods supported
   - Competitive price (with many discounts provided)
   - Good customer services
5. How important would you say customer service is?
   - It is one of the main factors keep customer stay with the ticketing system. For example, a new user do-not know how to check-out for a ticket, a pop-up Messenger icon display at the bottom-right corner of the website is come to the rescue.
6. Would you rather pay for the ticket via online payment or offline payment?
   - Online payment, since I just want to check-in then enjoy the show.
7. Do you want the system to build up an advanced searching tool?
   - Yes, because I do not want to spend so much time finding the exact event I want to anticipate.

8. What main color do you want to display on the website?
    - Maybe with bright color. Enable the user to choose between dark mode and light mode.