DACNTT_DigiKey_520H0011.pdf

by Bảo Chu Trần Gia

Submission date: 19-Aug-2024 06:34AM (UTC+0700)
Submission ID: 2434021957
File name: DACNTT_DigiKey_520H0011.pdf (2.79M)
Word count: 9372
Character count: 54312

Vietnam General Confederation of Labor
TÔN ĐỨC THẮNG UNIVERSITY
INFORMATION TECHNOLOGY FACULTY

CHU TRẦN GIA BẢO - 520H0011

DIGIKEY: EFFICIENT WEBSITE DEVELOPMENT WITH MICROSERVICE

INFORMATION TECHNOLOGY PROJECT

SOFTWARE ENGINEERING

HO CHI MINH CITY, 2024

Vietnam General Confederation of Labor
TÔN ĐỨC THẮNG UNIVERSITY
INFORMATION TECHNOLOGY FACULTY

CHU TRẦN GIA BẢO - 520H0011

DIGIKEY: EFFICIENT WEBSITE DEVELOPMENT WITH MICROSERVICE

INFORMATION TECHNOLOGY PROJECT

SOFTWARE ENGINEERING

Instructor
MSc. Nguyễn Ngọc Phiên

HO CHI MINH, 2024

ACKNOWLEDGEMENTS

We are sincerely want to give a thanks

to …… ……
……
……
……
……
……
……
……

Ho Chi Minh city, ... 20..

Author

(Sign and write full name)

THE PROJECT IS SUCCESS FULLY CONDUCT AT TON DUC THANG UNIVERSITY

I hereby declare that this is my own research project and is under the scientific guidance of MSc. Nguyen Ngoc Phien. The research content and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments, and evaluation were collected by the author from different sources and clearly stated in the reference section.

In addition, the Project also uses a number of comments, assessments as well as data from other authors and other organizations, all with citations and source notes.

If any fraud is detected, I will take full responsibility for the content of my Project. Ton Duc Thang University is not involved in copyright violations caused by me during the implementation process (if any).

Ho Chi Minh city, ... 20..

Author

(Sign and write full name)

DIGIKEY

SUMMARY

From the analysis above, DigiKey is a modern and perfectly designed electronic market place intended to offer strands of digital software. My project, DigiKey, uses a complex micro services for building a rather powerful, flexible, and work-oriented platform. Such architecture guarantee that every part of the platform works independently but simultaneously reduces the level of interdependence – which is always a sign of high-level flexibility and performance. The platform has the ability to not only scale but also grow in strength especially as DigiKey is likely to engage in a multitude of other transactions involving digital software.

While designing the structure of DigiKey, every aspect of the digital software business has been taken into account carefully. The aforementioned needs are met in the platform due to the development of simplistic and intuitive interface, secure means of transaction, and versatility in terms of available digital software formats that require licensing. The use of microservices self-services architecture bolsters the incorporation of progressive technologies making it feasible to facilitate processing and analyzing of big data in real time boosting decision support to business.

Also, the company is committed to providing the best service since it states that DigiKey values improvement. It currently caters for the dynamic market of digital software by adapting to provide the most efficient platform for customers. Focusing on the quality, we go through numerous testing to guarantee that users of digital software businesses would feel an effective interaction with our application.

DIGIKEY:

ABSTRACT

Our project goal is to conduct research and develop a e-commerce system with the aim to manage products, sells, revenues and the warehouse purchased by the users and managed by system administrator. Our approach for this e-commerce system is to use tech stack including: ReactJS, NodeJS, Strapi, ChartBrew and integrate with PayPal and Stripe. MySQL is used as a database to store data that will be applied for the data visualization purposes and report purposes, ReactJS is used as a front-end service to handle and render data, NodeJS and Strapi are used as a back-end service to process request from user and data between different actors within the use case.

The development sprints of our project has result in a modern, user-friendly e-commerce system and platform with security in managing and purchasing products and codes. Our project seeks to give a general understanding in developing a e-commerce system using Strapi and implement user data analysis.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATION

ERD	Entity Relation Diagram

CHAPTER 1. INTRODUCTION & PROJECT OVERVIEW

1.1 Project introduction

The first reason for choosing this topic is rooted in understanding when it comes to selecting different e-commerce platforms what one fails to note is that the present day platforms are highly flexible and allow for the purchase of electronics products as well as digital transaction codes without having to undergo any form of identification verification. This capability is attained while having sound security measures in place so as to protect the users of such products as well as the storage systems where such products are kept.

Different from many other e-business websites that tend to have consistent processes, layouts, and products, my project will integrate the following unique elements. I am fully conscious that most of the current platforms lack usefulness when it comes to real-life implementation but I hope that my project is going to contribute improvements. These distinctive aspects will be discussed in this broad report, with a focus on explaining the customer purchase process as well as the application of the latest technologies in the implementation of a range of functionalities. Besides, it will analyze the website's usability based on several user-oriented outlooks and discuss the potential vision connected it the advancement of the low-code and no-code approaches within the modern technological setting.

It is hoped that through analyzing the specific aspects of my project within this report, it will be possible to illustrate the opportunity for a more flexible and user-focused e-commerce environment and how, as such, it may majorly redefine the overall retail experience for digital goods. The detailed analysis will give a good understanding of the possibilities and real advantages of using technologies in my project to avoid main problems and limitations of regular e-commerce solutions.

1.2 Goal of the project

DigiKey is one of the well-developed online marketplaces strategically designed and constructed to enable the smooth buying and selling of digital software. Our project DigiKey incorporates and springs from a complex and refined microservices system that makes the whole solution very reliable and scalable on the one hand, and very much efficient on the other. This architecture means that every element of the platform works in parallel while at the same time being self-contained thus offering the best freedom and functionality. The microservices approach is also beneficial not only in terms of scalability but also in terms of reliability as DigiKey should be able to easily accommodate various digital software transactions thanks to the use of microservices.

However, while designing the DigiKey, intention has been paid to every thing that digital software businesses may require. These needs are well met through a friendly user interface, secure procedures of carrying out transactions, and support of numerous formats and licenses for digital software. Actualization of complex technologies within the microservices architecture makes it possible to process and analyze data in real-time, thus providing businesses with better insights and improving decision-making.

Also, there is an element of consistency in the excellence of DigiKey as demonstrated by its propensity to evolve. To this end, the platform has been made more responsive to an ever dynamic software market hence providing optimal solutions for efficiency. Through this testing process and development cycles, our main goal is to make sure that the experience that users will have throughout the different digital software businesses surpass all expectations.

Therefore, it can be concluded that DigiKey is the epitome of online sites that will facilitate transactions in digital software. The positive angle that brings out

the strengths of this tool is its microservices architecture coupled with the understanding of the digital software market, thus acting as a crucial tool for businesses that desire to survive in the digital software market. This stable and clever provide not only answers to the current digital software transactions' needs but also has a potential to be upgraded in the future and will remain to be useful in years to come.

These principles make sure that every transaction made through DigiKey is secure and effective for all the users. The professionally developed security measures and simplified layouts of the website make it suitable for firms that intend to ease the procurement of their digital software. Due to its smooth combination of modern technologies and sensitive focus to customers, DigiKey introduces a new level of excellence to the world of digital software.

1.3 Key features

While the DigiKey team will be focusing on an e-commerce platform, the vision is to create a project that can be used in as many contexts as possible, not being narrowed down to e-commerce alone. This versatility is kept in check by a complete feature list and a reliable microservices structure which makes for scalability, data efficiency and flexibility across the many applications that DigiKey is used.

Service:

Frontend: ReactJS Web App

The frontend of DigiKey is developed using ReactJS a robust and versatile JavaScript framework for constructing UI. This provides the responsiveness, which makes the user experience dynamic varying with the user's interactions to navigate across the platform. The usage of the ReactJS also allows for effective management and scalability due to its usage of the components concept, which is suitable when developing such a large scale project.

Inventory Management: Strapi

Stock management for the application is provided by Strapi, an open-source headless CMS. Thus, Strapi offers a convenient tools for content management and it is closely connected with the frontend. Consequently, due to the mentioned characteristics of content and relations' nature, it can easily support a lot of types and relations of various digital content and services. Architecture of Strapi makes it possible for other services in the microservices architecture to both access as well as leverage data.

Visualization: ChartBrew (Strapi plug-in)

Concerning data visualization, DigiKey uses ChartBrew which is a plug-in for Strapi. ChartBrew helps to create real time and dynamic charts or dashboards which offer useful information about sales, stock, and users. This is an effective means of decision making involving all the stakeholders since it incorporates all the appropriate and updated information.

Payments: ExpressJS (connected with Stripe, PayPal, and so on.)

Order processing is achieved with the help of ExpressJS service for payment processing and work with the most popular payment systems such as Stripe and PayPal. This is achieved to guarantee safe experimentation with transactions, which includes the ability to take different payments to address users from across the world. By integrating with reliable payment providers it increases the platform's credibility and overall user satisfaction.

1.4 Structure of the report

Chapter 1: Introduction

Chapter 2: Methodology

Chapter 3: Frontend - ReactJS

Chapter 4: Warehouse - Strapi

Chapter 5: Visualization - ChartBrew

Chapter 6: Payments - ExpressJS

Chapter 7: Results

Chapter 8: Conclusion

CHAPTER 2. METHODOLOGY

2.1 Research Design

After doing some research on the notion of the project, I personally think should implement a realistic project, but I want to make it possible for everyone can use and implement, so that it can be re-used for different frameworks, projects or websites, specifically. That is the reason why I come up with the idea of doing a e-commerce website, but have to meet following criteria:

- Low-code
- Re-usable
- Following microservice architecture
- Smooth user experience
- User-friendy UI

For front-end:

- As I want the User Interface to be friendly, I did some research for the digital product website, these are the e-commerce website that allow users to buy products right on the website without any complex actions or any third-party services action needed. Users still can buy the product without actually logging in. That is why I looked at Divine shop, they have a very good sense of User Experience, when the layout of the user interface is very bright, including the product presentation is very visible and easy to find. That is the first reason to meet the requirement - “User-friendly interface”.
- After that, I did re-design the user interface on the website, especially for the product detail page. I edit some components for the layout.
- For ReactJS, I choose this framework since the functionality since following features and agility it provides for the developers. I also learning while implementing the UI, so that I can gain more realistic knowledge to apply for the project.

For back-end:

- System Administrator: since I am doing the project on my own, and I also want to find the way to implement services without putting too much attempts on developing services, so that I can save more time doing other tasks like Front-end, Writing document and build up payment services. So I'm thinking of a way to apply a low-code open-source for the project, which allow me to apply all CRUD based service (Create, Read, Update, Delete) so that I can be a System Administrator for my product. And also, have a clear UI for the Admin site to mange the product, transaction, report and future trends.
- Data storage: while doing this, I found Strapi, and found out that it will use MySQL to manage datasets within the local host (the personal computer), which means I have to study about MySQL, how to use it, how to write a query and manage the system via MySQL.
- Payment gateway: In a process of purchasing item, there is a vital step of the payment processing, because at the current, my website allow the user to purchase product codes without logging in, so that the user will have to some steps of verification. To solve that, my website will require the user to enter some validation information, so that we can prevent scams and brute-forces attacks

For system design:

- For system design, I did some research on the payment gateway to support payment for the customer, so that they can receive product immediately after purchasing.
- Microservice: Since my product has multiple services, I want to keep the independent, so that when a service crashes, it won't affect other services like, while the other services still can run smoothly
- Database management: As mentioned above, I use Strapi, so everything can be managed on the very user-friendly user interface and easy to use since it follows the low-code convention.

2.2 Entity Relation Diagram (ERD)

Figure 2.1 ERD

Currently this are the table within the system that is defined to apply for the current system.

Account table: Used to store users' accounts including login credentials, purchase history, payment credentials, and the expiration of the purchased products.

Customer table: Used to store the session of each time the user's sessions starts when they start to checkout all items in cart.

Transaction table: Used to store all transaction of a user, update the status, amount of the products after each transaction, also used for revenue report purposes. Also, when applying new promotion programs, this table will record all price changes in order to keep track of the promotion effect in a specific number of transaction, also keep track of payment information, email information, so that if there is any scams our tricks happen, the administrator can trace back using this table to make sure every changes are transparent.

Product table: Used to store product information, since the website is selling digital products including game code,… so this table will only store the core, which is the hashed code, will be highly secure by algorithms to keep the unique code stands for a single product code.

Product Type table: Used to store the product type, this is what the user will see, but what user receives, will be stored in the Product table, which is the code or the allowance for that product type.

Category: Used to define the category of the product type, which brings up good experience for the user to find products faster and easier. As the system has different products in different area including: gaming, working, family-used service.

2.3 Use case

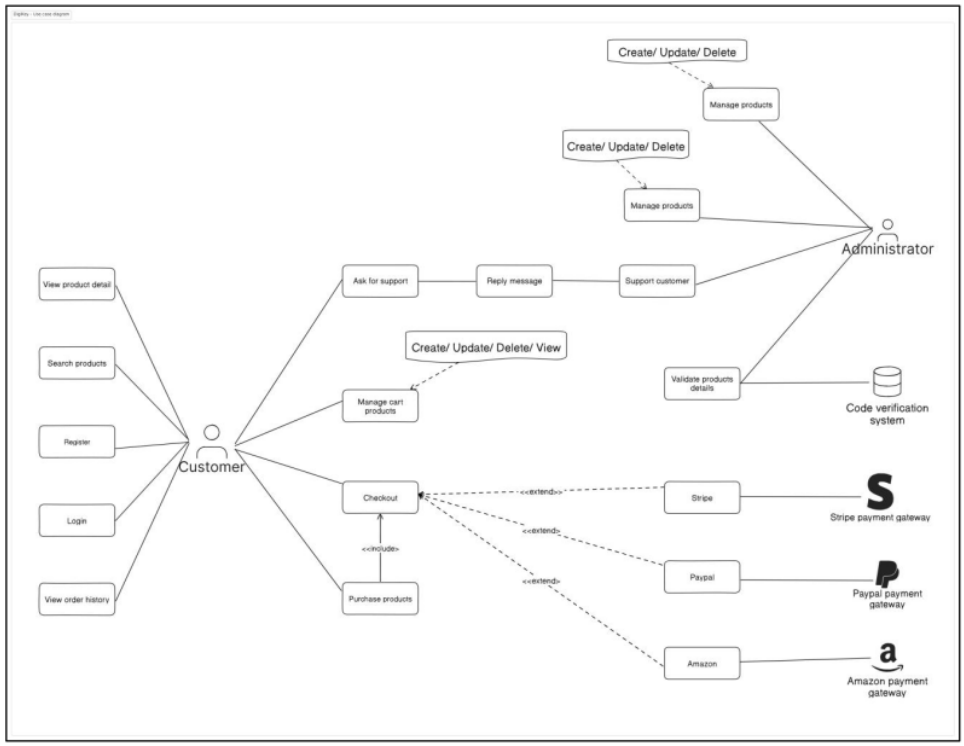


Figure 2.2 Use case diagram

According to the Use Case diagram

Factors:

- Customer
- System Administrator (Me)
- Inventory Staff (Me

Use case:

- As mentioned in the Functional Requirements section.

External actors:

- Third-party payment gateway

2.4 Functional Requirements

In this project, here is the text-based solution for the use case of the system by actors: User, System administrator, Inventory Staffs

- Digital profile management
  - The system must allow the user to register
  - The system must allow the user to login
  - The system must allow the user to change password
- Account management
  - The system must allow the user to update account credentials
  - The system must allow the user to view payment history
- Order management
  - The system must allow the user to place an order
  - The system must allow the user to update an order details
  - The system must allow the user to view order history
- Cart management
  - The system must allow the user to add products to cart
  - The system must allow
- Payments management
  - The system must allow the user to pay a bill without logging in via Stripe, PayPal.
  - The system must allow the user to pay a bill with login via payment gateway system

2.5 Non-functional requirements

1. Security requirements

- The system must ensure the security and privacy of the data and transactions of the customers and products by using encryption authentication and authorization techniques.

2. UI requirements

- The system must ensure the performance and responsiveness of the system by using optimization, compression, and caching techniques.

- The system must ensure the usability and accessibility of the system by using user-friendly, intuitive, and responsive design principles.

2.6 Tools and Technologies Used

Tools:

- VSCode: is one the most popular code editor in the community, as it supports multiple programming languages and has a wide range of extension ecosystem. It is such a good IDE to writing, debugging, and testing code efficiently, which makes it so user-friendly to the developers community.

- Eraser.io (Document): This tool is developed by the small team, and also used by a wide range of small teams and small collaborative environments. It allow the user to real-time editing and sharing documents to different collaborators. The plus point is that it use Markdown as the main language, which is the one of the favourite language among the developers community.

- GG docs (Document): Google Docs is a cloud-based document editing and collaboration. It has already to well-known by the office workers, and integrates well with other cloud-based Google Workspace tools, making it idealic for project documentation and planning.

- Jira (Document): It allow user to manage different aspects of the project, which is commonly used for agile development team. To be honest, I should have

use Trello for my project, since this is just a small project, the Kanban board would be much more helpful to the user rather than the Scrum board itself.

- Postman: This tool is widely known for it use, which is for API development and testing tool. It simplifies the process of developing and testing APIs.

Tech stack:

- Front-end:
  - HTML: This is a standard markup language for create a web pages. So I will not spend so much time talking about this language.
  - CSS: CSS is used to style a HTML elements, it brings the structure from HTML to become more alive, but it still just a look, can not move.
  - JS: To make it more beauty yet more lively, it will requires the language to implement the behavior to the code itself. This is when JavaScript comes in to play.
  - React JS: I will write more about why I choose this framework as the main Front-end framework. But to sum up, it is a powerful JavaScript library allowing developers to create reusable components and ensures fast and precise rendering and performance, easy for scaling up the project and maintain for Front-end development.
  - TailwindCSS: This framework is a utility-first CSS framework. With the use in building responsive and custom designs quickly, it provides a set of predefined classes without writing extensive custom CSS.
- Back-end:
  - NodeJS (ExpressJS): This is a runtime allowing the JavaScript to write back-end code on Chrome's V8 engine, ExpressJS is a small and quicker NodeJS web applicatoin framework that provides a agile set of components and features for web and mobile applications.

- Strapi: This is the core technology used in my project, as it is the open-source headless CMS and a user-firendly interface that provides a customizable API for content management. It allows for easy integration with various front-end frameworks and other services.
- MySQL: This tool is used mainly with ChartBrew, a plug-in in Strapi used for report purposes, is a relational database management system. In terms of handling structured data and complex queries, it offer a wide range of data management capabilities and is well-suited for a fast scaling project.

- Integration:
  - Stripe: Is one of user-friendly payment gateway, it also allow an open-source API to integrate into my system, which is used as one of the two main payment gateway in my system. With a wide range of supporting different payment methods and currencies, it making sure that secure and efficient transactions of the user will be processed fast and seamlessly through different systems.
  - PayPal: Besides with Stripe, PayPal also provide developers with an API to integrate with and because this a globally recognized payment service that offers secure online payment solutions. A lot of users will feel save and convenient when the system can integrate PayPal, give them a good enough option to make payment.
  - ChartBrew: This is a tool from Strapi, act as an plug-in of Strapi system. It allow the data to be visualized that integrate from Strapi. This tool demonstrates data through real-time and dynamic charts and dashboards, provide users with insights into sales, inventory and user data.

CHAPTER 3. FRONTEND - REACTJS

3.1 Introduction

For front-end, I chose ReactJS as a front-end framework and Tailwind-CSS as a library for the website.

ReactJS is a open-source JavaScript library, it allows the developer to implement different style with a flexible and easy-to-use structure. This library is trusted and used by big Tech companies like Facebook, Instagram, WhatsApp, Netflix, e.t.c.

I also use Tailwind-CSS to implement the UI, this library also support easy-to-use methods to customize the UI directly on the script page, instead of make separate folders UI elements.

3.2 Reason to select ReactJS

1. Easy to learn: With simple syntax, ReactJS allow the developer to learn to use quickly, which will save a lot of time to get used to it. The only knowledge required is to know how to code using JavaScript or TypeScript first.

2. Reusable components: ReactJS makes building web application easier as it saves time and money, which allows developers to create reusable components that they can use to create new applications or used across different projects.

3. Strong community support: One of the main reasons to choose React is because of how popular React is with developers. This frameworks is backed by individual contributors, they are ready to provide immediate help for newcomers and experienced.

4. Fast rendering: React is optimized for fast rendering, it is essential to define the structure ò the app at the beginning á it can greatly enhance the user experience, and also spend less time for the developers to build up an application.

5. Agility: ReactJS allow the developers to build web application and also mobile application with React Native, which also fast and easy to expand the domain knowledge.

3.3 Advantages of using ReactJS

1. Virtual DOM: This is the main factor for React high performance and speed. The virtual DOM creates components of the memory data structures. So the developers can update the browser, or make change to the system and compute the notifications, but it will not affect the real DOM on the website if only they apply those changes to the real DOM it is.

2. Code stability: This points out the code stability of ReactJS provides with developers. The data-binding mechanism stabilize the code and keep the consistent app performance are guaranteed.

3. Familiar with new-comers: I just spend about 3 weeks to learn deeply about the syntax and the basis of this framework. This framework has save a quite some time for me to focus more on the Back-end development. Also for the record, I find that React has a good combination with other UI framework like MUI, TailwindCSS, e.t.c.

4. User-friendly: With a strong foundation, it provides users with wide range of resources, technical documentation, tutorials and training materials, making JavaScript developers quickly catch up the speed of the trend technology.

3.4 ReactJS project File Structure

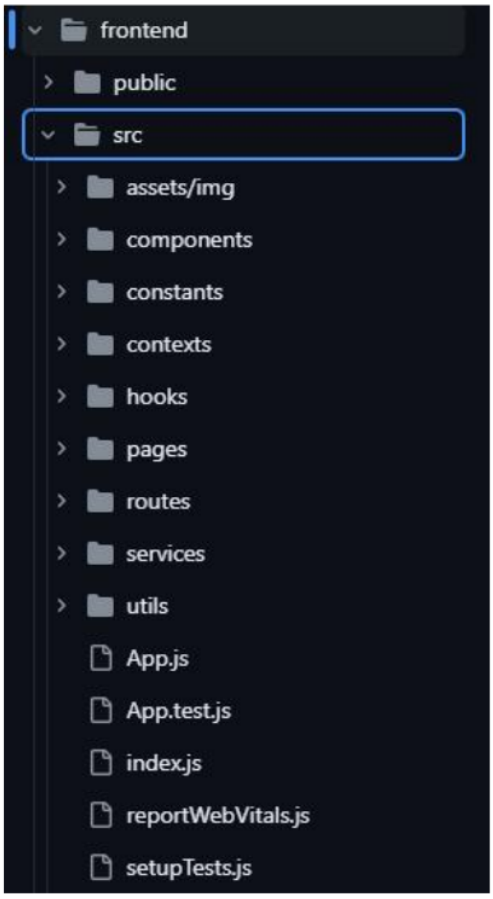


Figure 2.3. File Structure of ReactJS

Since my project is following the microservice architect, so there has to be multiple services from the back-end. And each of the service will be integrated differently from the “services” document. The rest of the document structure is the other component and pages.

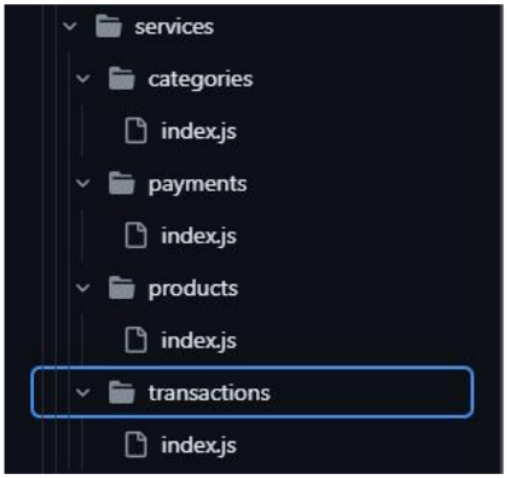


Figure 2.4. Service files structure

3.5 Integration with TailwindCSS

Using React

Installing dependencies

Tailwind UI for React depends on Headless UI to power all of the interactive behavior and Heroicons for icons, so you'll need to add these two libraries to your project:

```
npm install @headlessui/react @heroicons/react
```

These libraries and Tailwind UI itself all require React >= 16.

Creating components

All React examples are provided as a simple single component and make no assumptions about how you want to break things down, what prop APIs you want to expose, or where you get any data from.

Some data has been extracted into basic local variables just to clean up duplication and make the code easier to read and understand, but we've tried to do as little as possible to avoid enforcing any unnecessarily rigid opinions.

When you're adapting code from Tailwind UI for your own projects, you should break the examples down into smaller components as necessary to achieve whatever level of reuse you need for your project.

For example, you might start with this stacked list component:

```
const people = [
  {
    name: 'Calvin Hawkins',
    email: 'calvin.hawkins@example.com',
    image:
      'https://images.unsplash.com/photo-1491528323818-fdd1faba62cc?ixlib=rb-1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&a
  },
  {
    name: 'Kristen Ramos',
    email: 'kristen.ramos@example.com',
    image:
      'https://images.unsplash.com/photo-1550525811-e5869dd03032?ixlib=rb-1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&autc
  },
  {
    name: 'Ted Fox',
    email: 'ted.fox@example.com',
    image:
      'https://images.unsplash.com/photo-1500648767791-00dcc994a43e?ixlib=rb-1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&a
```

Figure 2.5. Integrate ReactJS with tailwindCSS

Resources & assets

Icons

All the icons we use across Tailwind UI are drawn from Heroicons, which is an open-source MIT-licensed icon set that we created and coded internally when we kick started determining Tailwind UI.

Images

The source of icons and pictures in Tailwind UI is limited and the major one you will find is Unsplash. If you are looking for resources for photographic and graphic material that can be used freely in your projects then it is a perfect match.

Illustrations

In using Tailwind UI, there are some instance that employs illustrations from the Lucid Illustrations set that is free from Pixsellz. They provide all of the illustrations shared in this post and other design materials on their website.

Figma assets

To address the third point, it’s important to note that we have discontinued Figma assets so that we could devote more time to creating more great examples using Tailwind CSS.

For a while, though, we offered downloadable Figma assets for Tailwind UI, although it was a tremendous amount of effort to sustain, and only a small number of users engaged with it. We’ve made the really hard decision to stop doing them so that we can focus on the code as that is where we believe we can add the most value.

3.6 Benefits of integrate with TailwindCSS

User-oriented approach: It allows developers to style element uses a utility-first approach directly in the HTML code. This leads to a faster and more efficient workflow.

Consistency: By using a predefined set of utility classes, it enforces design consistency, which will make sure that the same styles are applied across the app.

Customization: It enabling developers to use extensive configuration options to customize the design system to meet specific project requirements.

Responsive design: TailwindCSS includes built-in support for responsive design, allowing developers to create design that work seamlessly across different devices and screen sizes.

Performance: TailwindCSS minimizes the amount of CSS script by generating only the styles that are used in the project that needs to be loaded, improving the overall performance of the web application.

3.7 Project Set up and Configuration

1. Create React App

```
npx create-react-app my-app
cd my-app
```

2. Install TailwindCSS

```
npm install tailwindcss postcss autoprefixer
```

3. Configure TailwindCSS

```
npx tailwindcss init -p
```

4. TailwindCSS configuration

```
module.exports = {
  content: [
```

```
    "./src/**/*.{js,jsx,ts,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

5. PostCSS configuration

```
module.exports = {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
}
```

6. Import TailwindCSS

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

7. Start the development server

```
Npm start
```

8. Implement TailwindCSS

```
import React from 'react';

function App() {
  return (
    <div className="App">
```

```
        <header className="bg-blue-500 text-white p-4">
          <h1 className="text-3xl">Welcome to My App</h1>
        </header>
      </div>
    );
}
export default App;
```

3.8 Conclusion

This is an excellent combination when it comes to front-end development since integrating ReactJS with Tailwind CSS. ReactJS is great for creating an interactive web application, and on the other hand, Tailwind CSS is an efficient way of rapidly developing material that looks polished, and follows design guidelines. This combination enables the developers to design effective and high performing web applications that are also response driven.

By making the most of two technology solutions, developers do not need to dedicate a lot of time to crafting great styling processes because the tools can do that for them. The active community of ReactJS and the continuously growing community of TailWindCSS means that help, tutorials, and best practice are easily accessible for developers when starting with those technologies.

Its selection for this project will demonstrates the commitment of using up-to-date, clean, fast and scalable front-end structures such as ReactJS and Tailwind CSS. It does not only cut down the development time but also makes the end product maintainable and built for the future.

CHAPTER 4. WAREHOUSE - STRAPI

4.1 Back-end service

Since my project is an e-commerce website, but the problem I trying to solve in this project is not the website itself. The website itself is just an instance of the real core technology that I have applied in this project. Strapi, an open-source headless CMS. Which integrate with NodeJS. With the slogan:

"Manage Any Content. Anywhere."

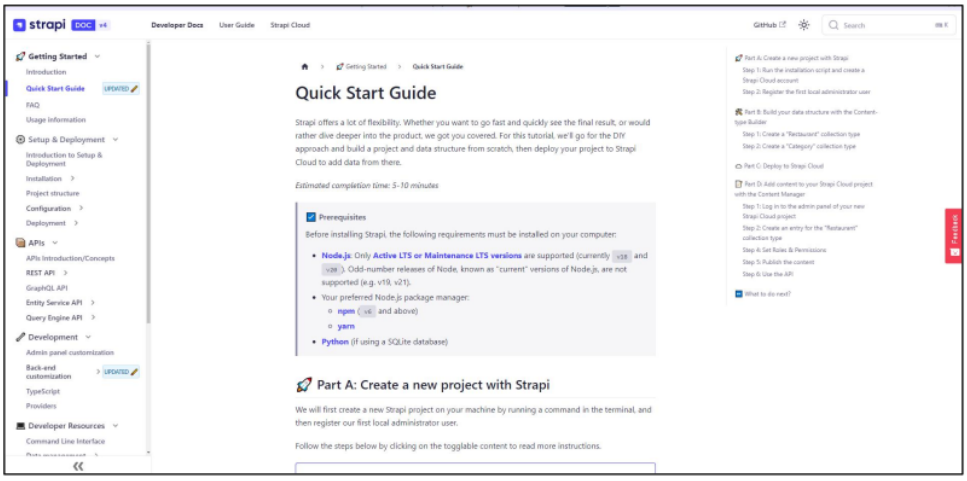


Figure 4.1. Introduction to Strapi

Since it is a "headless" open-source CMS, it allow everything to be setup quickly but still maintain the flexibility. Whether the user want to go fast with the trendy style of Low-Code or No-Code to quickly see the final result, or would rather dive deeper into the product, and everything will be visualized by a very user-friendly set of UIs only for the system administrator or any users that has enough permissions to join in this greatest of all time open-source CMS. Since it allow the user to go for the DIY approach and build a project and data structure from scratch. Every data should be stored on cloud storage, so no worries about data loss.

This open-source CMS provide developers with:

- The user-friendly and intuitive user interface for the main administrator to create and manage records of different content. This shall be used for the content management module.
- The authorization structure is following:
  - Super Admin
  - Admin
  - Staff
- The integration with the NodeJS framework will enable to developer to customize UI without any interfere or any kinds of the back-end services, since the service itself has already handle all the CRUD services (Create, Read, Update, Delete) *restful API* for the application. Besides that, super administrator can also make change to the layout of the structure for the admin structure

4.1.1 Authentication & Authorization

The authentication & authorization module of Strapi is the following image, the user can get to this page once they logged in successfully and manage to run the npm command, which I will show a demonstration in Chapter 7. Result.

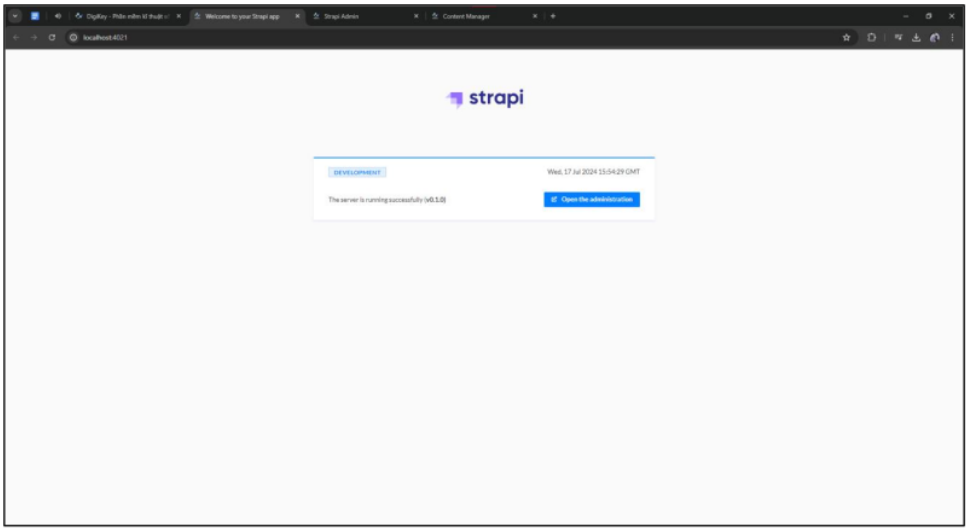


Figure 4.2. Welcome page - Strapi

For the authentication, the system will prompt the user to enter their email and password, in case the user don't have an account yet, the system allows them to create a new account, which will enable the user to enter any email address, it doesn't have to be the real one, since this is stored on the local host, so every update will only be on the host's computer. The rest of those changes will not be synchronized to others' computers.

For Authentication, before you can log in, make sure you have your account connected to the email address, but that email do not have to be the real one, since this is just for the local host, every login credentials have been store in .env file in the main directory. So that the user can have free access to different aspect of the system with different email addresses and operate like a real company managing the flow of their warehouse and the content manager in general.

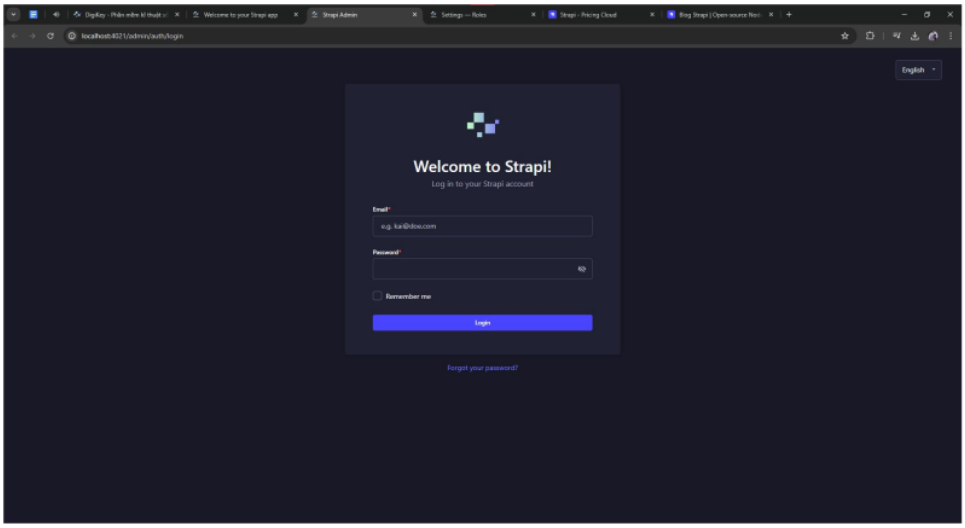


Figure 4.3. Login page - Strapi

For Authorization, currently the main user who is developing and integrating Strapi will by default become a Super Admin of the Backoff system. As I have mentioned before, currently the Strapi system has 3 default roles, which are:

- Super Admin: This role allow the user to access and manage all features and settings. They can also create new roles and users to access to the system and log into it, so that they can help the super admin to create and manage the content. Specifically, here is what the Super Admin (Me) can do in the system:
  - Full access to all manage collections types, and of course, the super admin himself can remove some rights, but in the end, the super admin will only the one to restore it.
  - Full access to manage all single types, these single types mean they are plug-ins or standalone of Strapi, but the thing is they are independent, since most of them are 3rd party plug-ins. So only the Super Admin can have full access to these kind of modules.
  - Full access to manage all plug-ins including: Content-Manager, Content-type-builder, Upload and User-permissions.

- Full access to Settings module, where the system allow the super admin to configure: Email, Media Library, Internationalization, Plugins and marketplace, Webhooks, User and Roles, API tokens, Project, Transfer tokens.

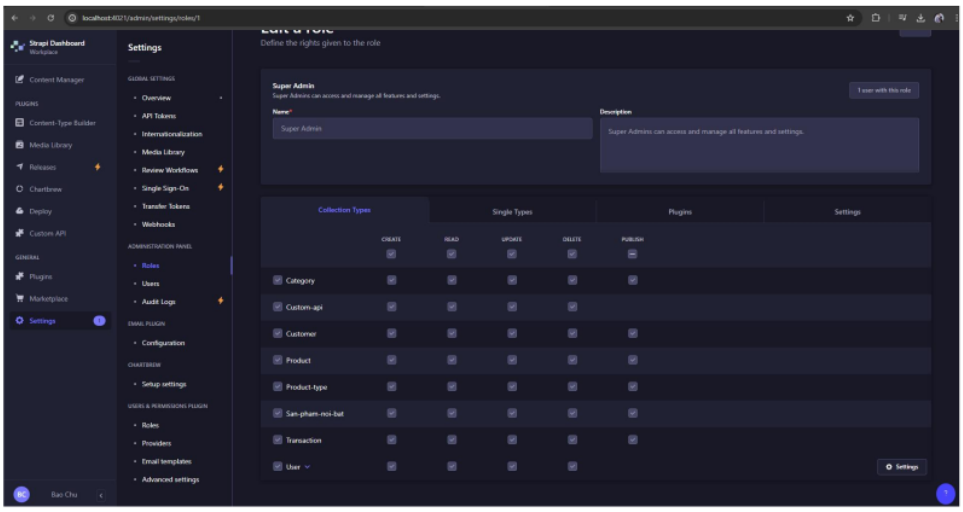


Figure 4.4. Super Admin - Strapi

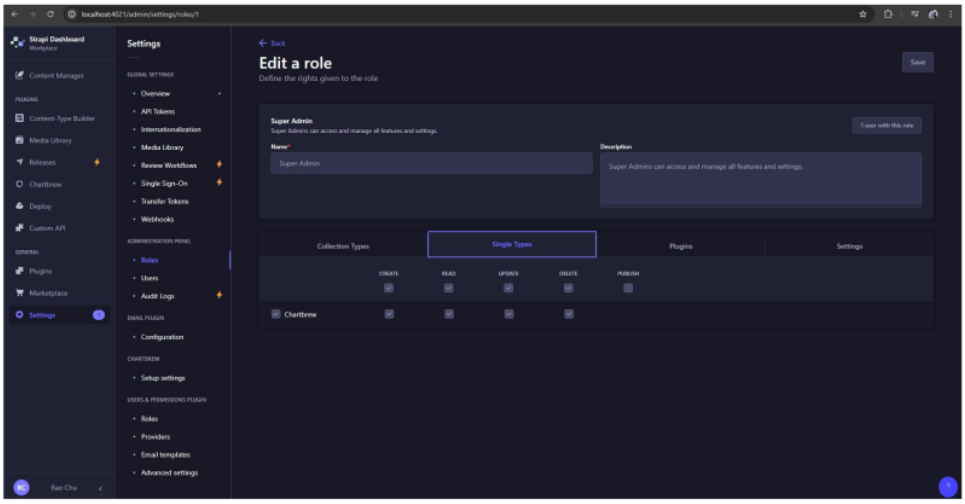


Figure 4.5. Super Admin - Strapi

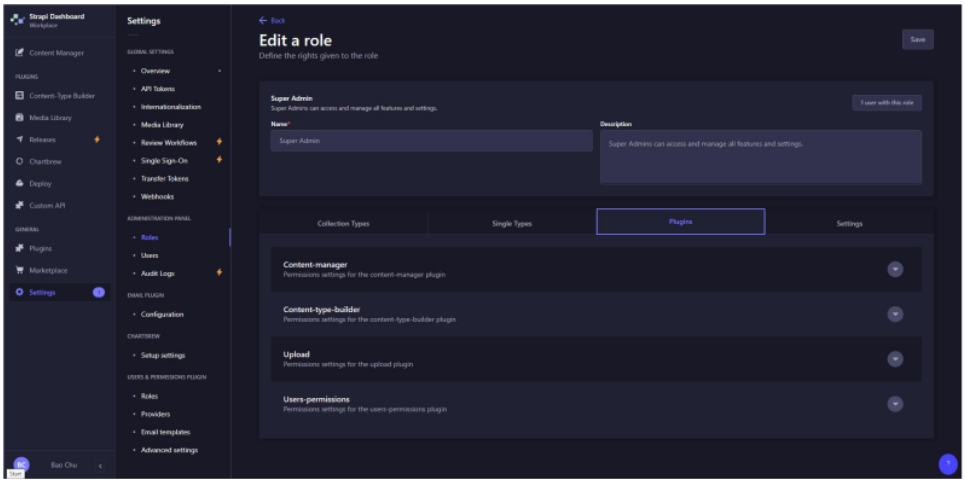


Figure 4.6. Super Admin - Strapi

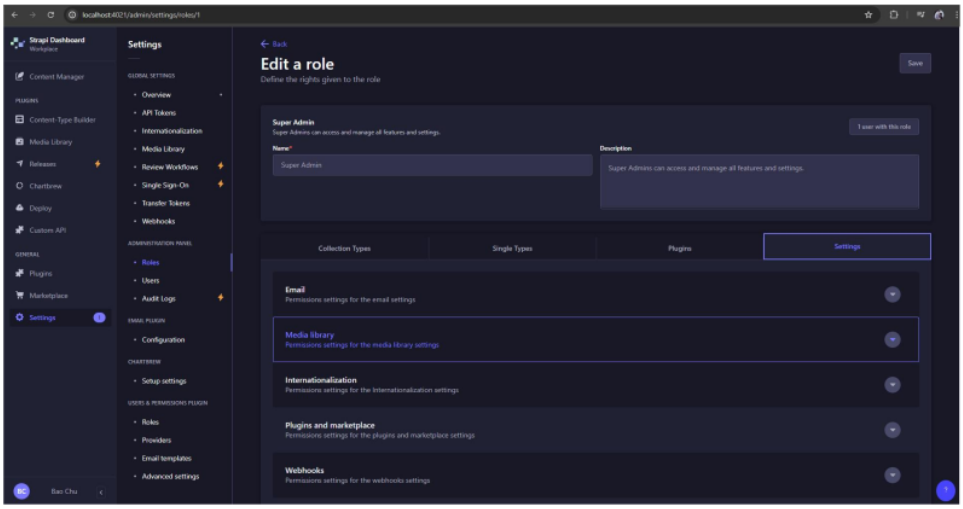


Figure 4.7. Super Admin - Strapi

- Admin: The user with the role of Admin can only manage content they created.
  - Only full access to collections they created, other content created by the Super Admin is read-only, or in case that the super admin give the admin permissions to do these actions.

- Only full access to single types that they created and only read-only other contents created by the super admin, or in case that the super admin give the admin permissions to do these actions.
- Has no access to plug-ins.
- Partial access to Settings module in a specific modules.

- Editor: The user with the role of Editor can manage and publish contents including those of other users.

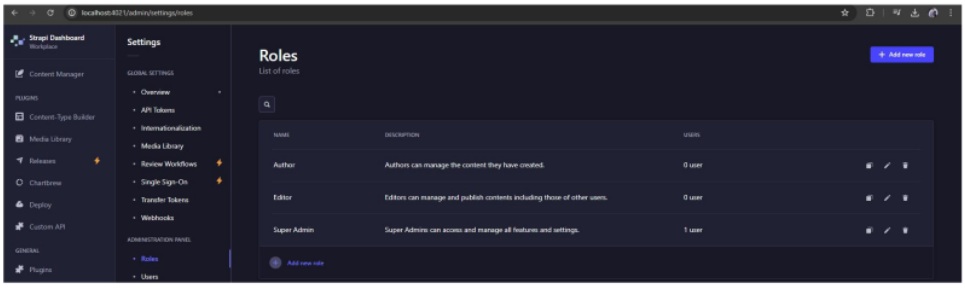


Figure 4.8. Users and Roles- Strapi

4.1.2 Installation guide

Required specification:

- Strapi will need following requirements to be installed on your computer:
  - NodeJS v18. or v20. But prefer v18.
  - NodeJS package manager:
    - npm (v6. and above) (Currently Im using npm)
    - yarn
  - Python (If using a SQLite database)
  - MySQL (v8.0)

1/ Run the installation to create a Strapi Cloud account

Npx create-strapi-app@latest my-strapi-project --quickstart

2/ Register the local administrator user.After create a new account, a new admin panel will be host on: *localhost:1337/admin*

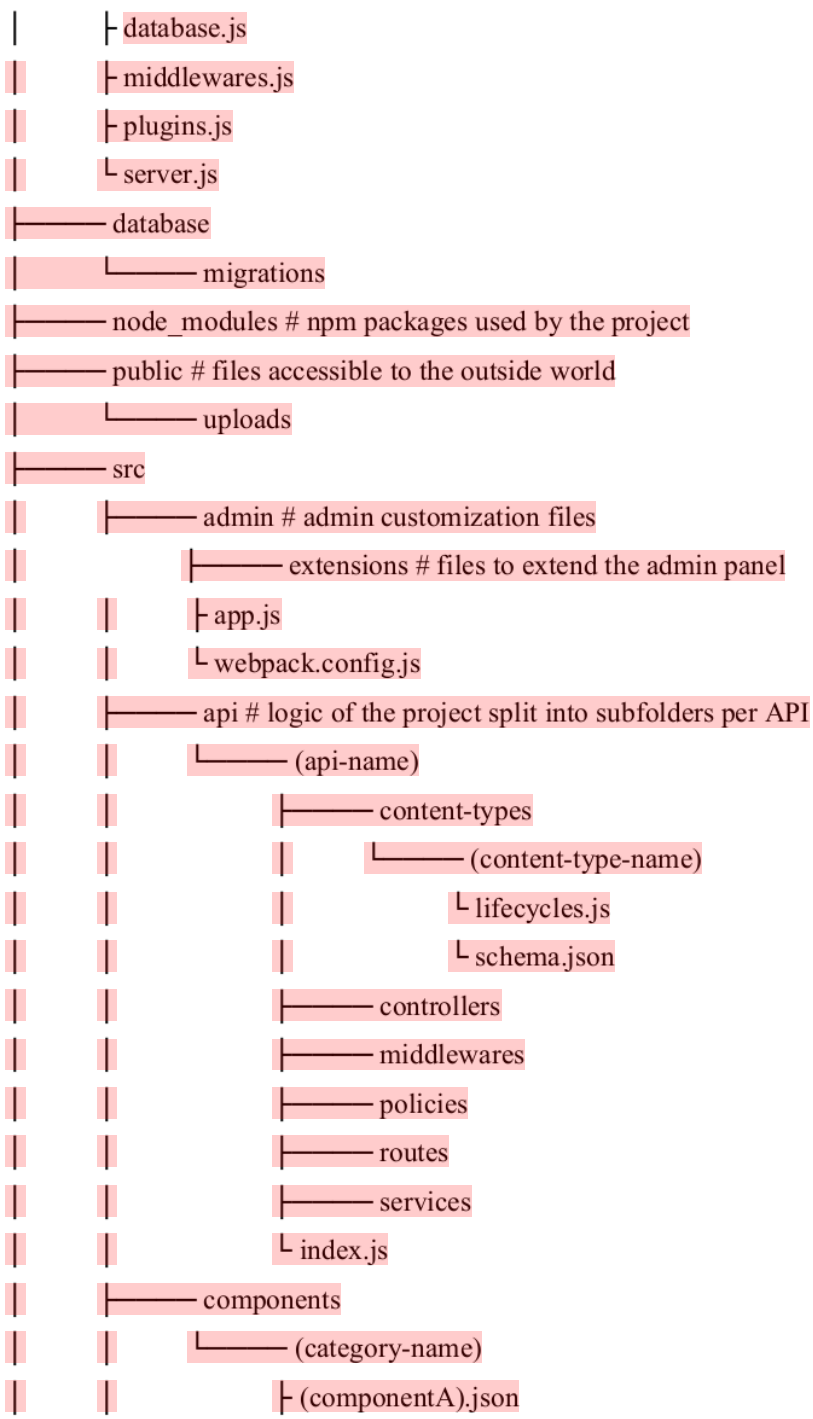
3/ Create the first collection type

- Click on the *Create your first Content type* button
- Click on *Create new collection type* button
- Click on "*First-collection"* for the Display name, and click Continue
- Click the Text field
- Enter name in the *Name* field
- Switch to *Advanced Setting* lab, and check the Required field and the Unique field settings.
- Click on *Add another field*
- Chose the rich text (Blocks) field in the list
- Type Description under the *Name* field, then click *Finish*
- Finally, click *Save* and wait for Strapi to restart.

4.1.3 Project Structure

This is just an example of the directory structure from the website, since I am building the project at the time I writing this, so I will not share the information about the strucutre, but rather show to the readers based on my github.

```
. # root of the application
├──── .strapi # auto-generated folder — do not update manually
│       └──── client # files used by bundlers to render the application
│               ├ index.html
│               └ app.js
├──── .tmp
├──── build # build of the admin panel
├──── config # API configurations
│       ├ api.js
│       ├ admin.js
│       ├ cron-tasks.js
```


├ database.js
├ middlewares.js
├ plugins.js
└ server.js
database
└ migrations
node_modules # npm packages used by the project
public # files accessible to the outside world
└ uploads
src
admin # admin customization files
extensions # files to extend the admin panel
├ app.js
└ webpack.config.js
api # logic of the project split into subfolders per API
└ (api-name)
content-types
└ (content-type-name)
└ lifecycles.js
└ schema.json
controllers
middlewares
policies
routes
services
└ index.js
components
└ (category-name)
├ (componentA).json

```
│   │   └ (componentB).json
│   ├─── extensions # files to extend installed plugins
│   │   └─── (plugin-to-be-extended)
│   │       ├─── content-types
│   │       │   └─── (content-type-name)
│   │       │       └ schema.json
│   │       └ strapi-server.js
│   ├─── middlewares
│   │   └─── (middleware-name).js
│   ├─── plugins # local plugins files
│   │   └─── (plugin-name)
│   │       ├─── admin
│   │       │   └─── src
│   │       │       └ index.js
│   │       ├─── server
│   │       │   ├─── content-types
│   │       │   ├─── controllers
│   │       │   └─── policies
│   │       ├ package.json
│   │       ├ strapi-admin.js
│   │       └ strapi-server.js
│   ├─── policies
│   └ index.js # include register(), bootstrap() and destroy() functions
├ .env
└ package.json
```

4.1.4 Strapi vs. WordPress

WordPress has already well-known in the developers community when it allows the user to build their own website with low-code or no-code configuration. However as the limitation in the capabilities of customization with the limited block styling and a way of approaching for non-developer users will be quite hard for the script editor that requires users to understand their own logic rather than using a popular script among all the script in the environment at that time. In this section, I will compare the feasibility that Strapi provide users with when it comes to low-code or no-code trend by pointing out the advantages and disadvantages of 2 CMS. But I will only compare on the technical side, since the business side is out of my reach and out of the scope of this project, so there will be no mentioning about the revenue, market share or any statistic of the business value to the community.

Key features - Strapi:

- Headless Architecture: Strapi operates as a back-end as a service basis, which means it divides the back-end from the front-end; developers can implement any type of front-end solution including React, Vue, or Angular, etc.
- API First: Besides, Strapi provides automatically RESTful and GraphQL API according to the content types specified allowing the application to interact with other applications and services.
- Customization: In this context, Strapi is highly extensible so the developers can even change how the admin panel looks like or develop new plugins and adjust the functionality according to their needs.
- User-Friendly Interface: The admin interface is simple to use which makes it easy for even non-professionals to come in and post stuff.
- Role-Based Access Control: All that is subject to changes or can just be read-only, depending on the user's role or needs – Strapi supports user permission at a micro level.

- Plugin Ecosystem: Currently, Strapi has various plugins provided which additional features to the platform, for instance, the email plugin, the upload plugin, and the documentation plugin.
- Local and Cloud Deployment: There are two authentic scenarios in which Strapi can be deployed: and as a local development environment and in cloud environments for production.

Key features - WordPress:

- Themes and Plugins: Like any other commonly used website builder, WordPress hosts a multitude of themes and plugins which allows for the easy creation of a website with a unique look and feel and implement new additional functions.
- User-Friendly Interface: The admin panel is friendly where users with little understanding of how websites and applications are built will have the capability of managing the content.
- Community and Support: WordPress is an open source software and currently has a large and engaged user base that creates an abundance of documentation, guides, and help resources.
- SEO and Marketing Tools: WordPress has SEO features integrated into the software and hundreds of various SEO and marketing plugins available.

Compare Strapi vs WordPress

Feature	Strapi	WordPress
Architecture	Headless (API-first)	Monolithic (Traditional CMS)
Customization	High (custom plug-in, extendable admin)	High (themes, plugins, custom code)
Performance	High (NodeJS based)	Moderate (PHP based)
Scalability	Excellent	Good
Ease of use	Moderate (developer-centric)	High (user-centric)
Plug-in system	Growing strong	Extensive but slow
API integration	Automatic RESTful and GraphQL	REST API (require plugin)
Security	High (modern practice, customizable)	Moderate (many plugins can be unsecured)
SEO tools	Requires customization	Built-in and extensive plugins
Community support	Growing	Large but lack of innovation
Cost	Free and open-source	Free and open-source
Deployment flexibility	Local and Cloud	Local and Cloud

Tableg 4.1: Strapi compare to WordPress

Pros of Strapi:

- User-friendly interface for content management: Strapi's admin panel is built with usability in mind; the interface is clearly laid out and gives content creators the ability to create and manage content with no coding knowledge required. It is most advantageous in that aspect when used in low code or no code scenarios, where the key focus needs to be placed on how they solve problems.
- Flexible content modeling: The last perk of Strapi is the ability to design content types according to the user's choice with each created type appearing as a graphical interface. Users can create, edit, and overload content structures with little to no interference with the actual code, which is completely consistent with the concept of low-code and no-code.

- Automatic API integration: Among Strapi's strong suits, it is worth mentioning that the platform automatically creates RESTful and GraphQL API for each created content type. This capability implies that the users can easily share their contents to several front-end frameworks and applications in a more convenient way since it does not involve creation of new API.
- Headless Architecture: With the headless approach, the CMS Strapi allow developers to choose and change front-end technologies without any affect on the back-end services, which making it future-proof and versatile.
- Performance: Since this open-source is build on NodeJS, generally offers better performance and scalability compared to WordPress PHP-based architecture.
- API-first: Strapi automatically generates RESTful and GraphQL APIs for all content types, making it easier to integrate with other systems and applications.
- Customization: Strapi has a high ability that allow the developers free in customize the the structure, which allow the developers to tailor the CMS to their specific needs, extending the administrator panel and integrate with custom plugins.
- Tech stack: Strapi uses a modern tech stack, which has a lot more potential to grow and appealing to the contemporary developers.
- Visual workflow management: Some of the features of Strapi include the ability to visually design or redesign work flows and content approval checklists. This feature is important in most low code platforms to allow stakeholders who do not have technical backgrounds to get involved in the actual content creation and the publishing of these contents.
- Cloud-native capabilities: Currently, Strapi is built to be deployed on different cloud providers so that it can be handled easier in cloud environments. This scalability is desirable especially for low code projects given that they can experience huge variances in traffic without the need to overhaul the infrastructure.

CHAPTER 5. VISUALIZATION - CHARTBREW

5.1 Report Service

As an System Administrator, it is very important that the admin can view, manage and adjust the warehouse import and export status, that is when the report comes in, not only the report show a statistic in the specific time frame, but also since the number doesn't lie, that's why it is so smooth that the admin can FORESEE the trend of the customer behavior based on the report diagram and a little bit of data analysis and data visualization skill.

The reason I start using ChartBrew is because the Strapi CMS only allows users to manage the content of the website, there is no data visualization, data analysis features, then I researched on Strapi to find for the plug-in that can help me with this.

Cons of ChartBrew:

- User-friendly: This plug-in allows users to customize different types of data visualization way such as:

Pros of ChartBrew:

Pie chart: Allows the users to present data with a specific number of percentage with the max range is 100%, every elements inside the chart will summarize into 100%, and each of the element will be presented as different color to distinguish them from others.

Line chart: Allows users to present data with a specific statistic in a specific amount of time or any kind of unit, and the chart will link these values and map them into a line, to show the difference of that data set through a selected period of time.

And more, but I will show you in the following images

Figure 5.1. Chart of Transaction

Figure 5.2. Chart of Product warehouse status

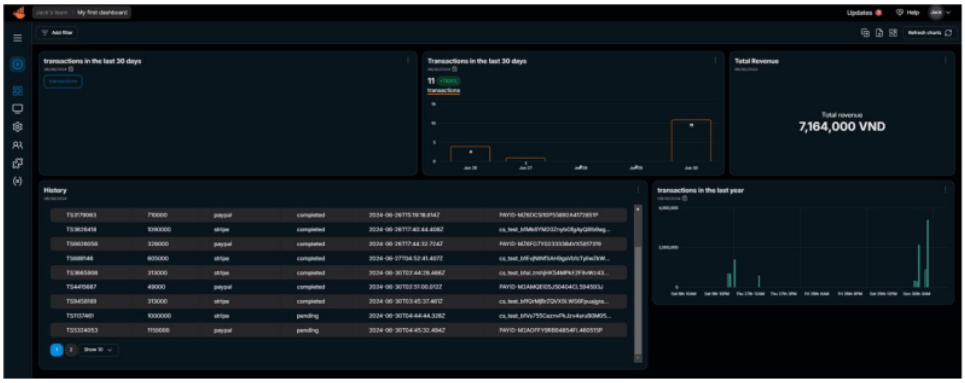


Figure 5.3. Dashboard of ChartBrew

- Real-time data visualization: Not only allowing the user to present data, but it also allow the user to present all data set and update in real-time. With that, from Strapi, the user can edit the structure of the report, dashboard and type of the chart, and that change will be autosave then synchronize with the Strapi at a glance. Which will help the system administrator can see the information on warehouse import and export status. This real-time data and information will be the important element for making timely decisions and adjustment.

Figure 5.4. ChartBrew in Strapi

Integration with multiple Data source: Not only Strapi that being supported by ChartBrew, ChartBrew also is able to integrate with a wide range of data sources, This will enhance the flexibility allows administrator to pull data from different sub-systems, allowing the administrator with a comprehensive view of operations.

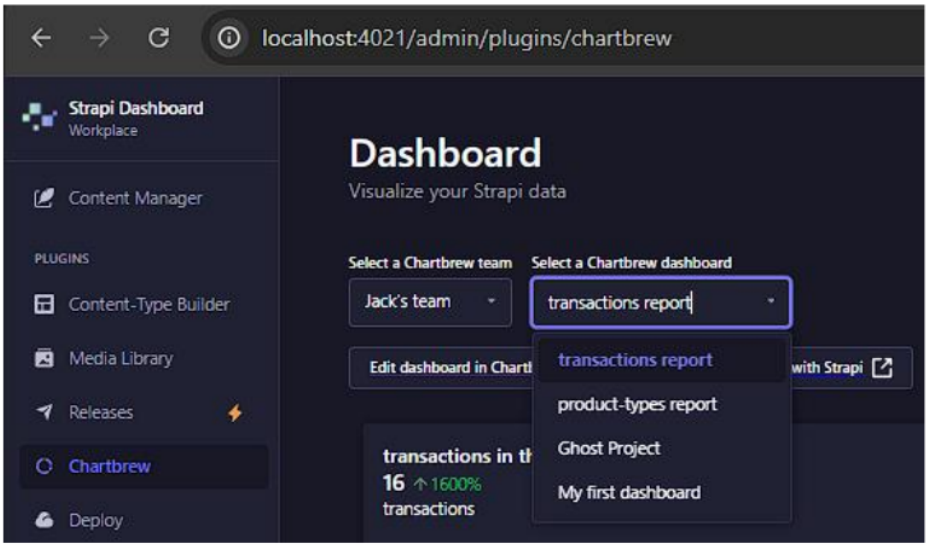


Figure 5.5. Support for multiple dashboard (data source)

Interactive visualization: As mentioned in the last section, ChartBrew allow administrator to interact with charts and diagrams, which will help them to drill down into specific data points so that they can analyze more detailed on the tailored data set. This ability enhances the ability to understand complex data trends and customer behavior.

Open-source and extensive: Just like the idea of interaction, ChartBrew is highly extensible when allowing the developers to customize and extend its functionality to meet specific organization needs. This is a good solution to handle tailored and effective reporting tasks.

Cons of ChartBrew:

- Initial Setup and Configuration: Setting up and integrating ChartBrew with Strapi has consume a lot of time and technical expertise. This setup process might be challenging for non-technical users.

- Limited Advanced Analytic: Since the ChartBrew plugin only focus on data visualization, this is also the drawback of it, when it is incapable of implementing more complexes and eye-catching data analysis tool. If this module is applied on the complex analytical needs, then they will need additional tools.

- Performance with large datasets: ChartBrew sometime have difficulty in handling with very large datasets, which will lead to performance issue such as bottleneck, e.t.c. Which will lead to negative user experience when processing and presenting extensive data.

- Community and Support: Since the clout of ChartBrew is not so big, which means that the development team is not large enough to provide users with official supports and documentation comparing to commercial products. Hence, the user will have to rely on community support and contribute to ongoing development.

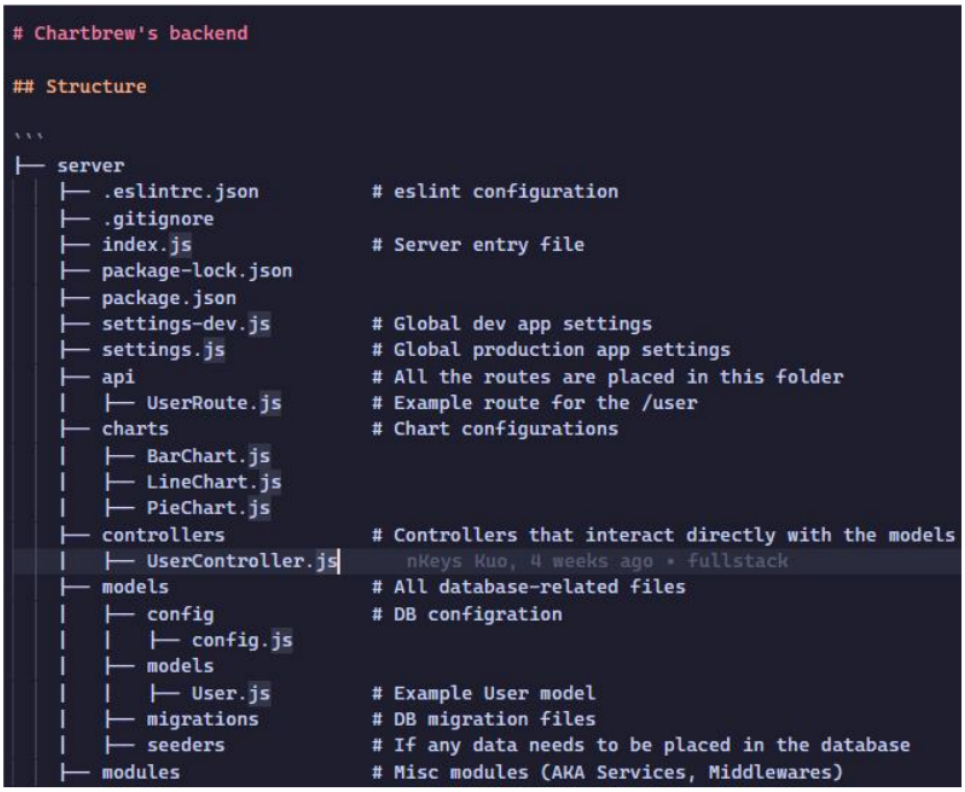
- Dependency on external library: ChartBrew relies on several external libraries and dependencies, compatibility conflicts might happen since the dependencies will have to update the software regularly to ensure security and compatibility.

- Learning Curve: Even though ChartBrew has a clear and user-oriented view, we can't deny the fact that there can still be a learning curve for new or non-technical users to deal with data visualization concepts or who are new to using such plugins within Strapi.

Overall, ChartBrew provides significant benefits and features to meet the bare minimum requirement from the user to present data via charts and other forms. Despite some limitations, its advantages in real-time data visualization,

customization make it a good tool for managing warehouse import and export status and gaining insights into customer behavior trends.

5.2 Applying ChartBrew



```
# Chartbrew's backend

## Structure

```
├── server
    ├── .eslintrc.json          # eslint configuration
    ├── .gitignore
    ├── index.js                # Server entry file
    ├── package-lock.json
    ├── package.json
    ├── settings-dev.js         # Global dev app settings
    ├── settings.js             # Global production app settings
    ├── api                     # All the routes are placed in this folder
    |   ├── UserRoute.js        # Example route for the /user
    ├── charts                  # Chart configurations
    |   ├── BarChart.js
    |   ├── LineChart.js
    |   ├── PieChart.js
    ├── controllers             # Controllers that interact directly with the models
    |   ├── UserController.js      nKeys Kuo, 4 weeks ago • fullstack
    ├── models                  # All database-related files
    |   ├── config              # DB configration
    |   |   ├── config.js
    |   ├── models
    |   |   ├── User.js         # Example User model
    |   ├── migrations          # DB migration files
    |   ├── seeders             # If any data needs to be placed in the database
    ├── modules                 # Misc modules (AKA Services, Middlewares)
```

Figure 5.6. Applying ChartBrew into the project
```

CHAPTER 6. PAYMENT - EXPRESSJS

6.1 Payment Service

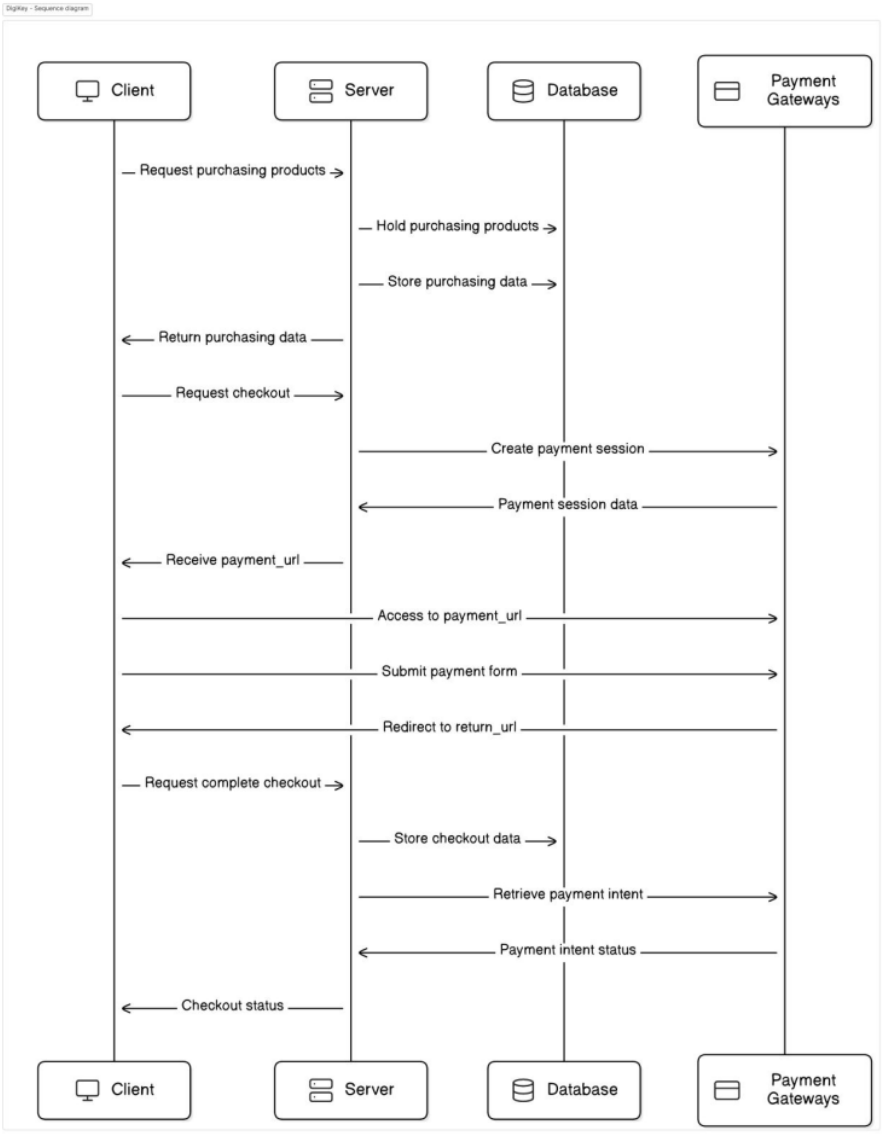


Figure 6.1. Payment services

Describing the sequence diagram:

1/ Request purchasing products

Client: The user start the session by purchase a set of products and sending a request to the server

Server: The server receive the request and and store it in the database

2/ Return purchasing data

Server: The server checks and returns the purchasing data to the client

3/ Request checkout:

Client: Reviewing the data, the user requests to proceed to checkout.

Server: Run a payment session, call the payment gateway (Stripe & PayPal)

4/ Create payment session:

Server: The server also create a session and stores it in the database

5/ Receive Payment URL:

Server: Receiving the payment URL from the payment session data

Client: Send payment URL back to the client

6/ Access Payment URL:

Client: Go to the payment URL and fulfill the payment form

7/ Submit payment form:

Payment Gateway: Process the payment

8/ Redirect to return URL:

Payment Gateway: If payment is success, redirect user to the return UR

9/ Request complete checkout

Client: Client application request the server to complete the checkout process

10/ Store checkout data:

Server: Stores the checkout data in the database

11/ Server receives the payment intent status from the payment gateway

12/ Checkout status:

Server: Update checkout status

Client: Send successful message to client

For payment services, the project is integrate with Stripe and PayPal to connect with 3rd party that allow the system to process the payment. Since we want to use payment gateways that is well-known among the community, so that it will be save for the user from the local environment, UAT and even production environment.

In this project, we are using:

- PayPal:
  - **Wide Adoption:** With millions of clients all over the world, PayPal is one of the most popular payment gateways. Which allows it to give the nature of the website, therefore, informs the fact that many clients have subscribed to PayPal and as such have shunned the delay confirmations.
  - **Security:** PayPal has it clout from the very strong security mesaures as it includse the encryption aspect and fraud fighting tools. Transactions of the users from the different cases is being guarded of fraudulent acts and even cyber attacks.
  - **Ease of use:** With a very intuitive user interface, the payment can be processed very smooth and easily. With the multiple currency support, it is very helpful if the target niche is global.
  - **Seller protection:** Not just the customer, PayPal aslo bring the security to the seller, which help the seller in averting fraudenlent transactions and chargebacks and therefore increases security to the seller.

- Stripe:
  - **Developer-friendly:** Talking about Stripe, even an average IT student at least heard about it once in a life time. With a rich, powerful and versatile resources of APIs, Stripe is famous for developer-friendly. This is why Stripe is often the first choice of the developers for the payment gateways, considering it ideadlice because of the versatility that brings in the construction of these payment interfaces.

- **Flexibility:** With the coverage of acceptable means of payment, Stripe's scope is on a wide range of area in terms of credit cards, debit cards, and other payment methods accepted within a particular country, which allow users to make a payment like whatever they want, Stripe can support everything that is on their hand.
- **Advanced feaetures:** Wirth supoerior features available that Stripe is working well on that, such as pay as you go, invoicing and integraetd financial reporting services.
- **Security and Compliance:** Stripe has a very high security working with elements like elements crediting, encryption, and PCI DSS on transaction protection, and fraud related issues just like PayPal.

Feature	PayPal	Stripe
Integration	Moderate	High
Transaction Fees	2.9%+$0.30 per transaction	2.9%+$0.30 per transaction
Currencies	25+	135+
Customer support	Extensive	Extensive
Security features	Strong	Strong
Advanced	Limited	Extensive(Subscription,invoicing)

Table 6.1: PayPal compare to Stripe

Pros and Cons:

	PayPal	Stripe
Pros	- Large community - Well-secured for users - Visible workflow for user	- Developer-friendly - Extensive support for various payment - Competitive transaction fees
Cons	- Higher feess for curency conversion - Limited advanced features	- Requires more technical knowledge for integration. - Less well-known to end-users compared to PayPal.

Table 6.2: PayPal & Stripe pros and cons

6.2 Integration with Stripe and PayPal

```
const createStripeSession = async (options) => {
        try {
                const { items, success_url, cancel_url } = options;

                const lineItems = items.map((item) => {
                        const unitAmount = Math.round(item.price * 1
                        return {
                                price_data: {
                                        currency: "usd",
                                        product_data: {
                                                name: item.name,
                                                images: [
                                                        item.image |
                                                                "htt
                                                ],
                                        },
                                        unit_amount: unitAmount,
                                },
                                quantity: item.quantity,
                        };
                });

                // console.log(lineItems);

                const session = await stripeGateway.checkout.sessior
                        payment_method_types: ["card"],
                        mode: "payment",
                        success_url: success_url,
                        cancel_url: cancel_url,
                        line_items: lineItems,
                        //  Asking address in Stripe
                        billing_address_collection: "required",
                });

                // console.log(session);

                return {
                        paymentId: session.id,
                        url: session.url,
                };
        } catch (error) {
                console.log(error);
                return undefined;
        }
};
```

Figure 6.2. Stripe integration code

```
const createPaypalSession = async (options) => {
	try {
		const { success_url, cancel_url, items, total } = options;
		const create_payment_json = {
			intent: "sale",
			payer: {
				payment_method: "paypal",
			},
			redirect_urls: {
				return_url: success_url,
				cancel_url: cancel_url,
			},
			transactions: [
				{
					item_list: {
						items: items.map((item) => {
							return {
								...item,
								price: item.price.toFixed(2),
								currency: "USD",
							};
						}),
					},
					amount: {
						currency: "USD",
						total: total.toFixed(2),
					},
					description: "SUD payment gateway test",
				},
			],
		};

		const payment = await new Promise((resolve, reject) => {
			paypal.payment.create(
				create_payment_json,
				function (error, payment) {
					if (error) {
						console.log(error.response.details);
						reject(error);
					} else {
						// console.log(payment)
						resolve(payment);
					}
				}
			);
		});

		for (let i = 0; i < payment.links.length; i++) {
			if (payment.links[i].rel === "approval_url") {
				return {
					paymentId: payment.id,
					url: payment.links[i].href,
				};
			}
		}
	} catch (error) {
		console.log(error);
		return undefined;
	}
};
```

Figure 6.3. PayPal integration

CHAPTER 7. RESULTS

7.1 Implementation results

7.1.1 Microservice

The DigiKey team wishes to develop the project that can be applied in different scenarios and use case in stead of only for e-commerce scenario.

Service:

- Front-end: ReactJS Web App
- Warehouse Management: Strapi
- Visualization: ChartBrew (Strapi plug-in)
- Payments: ExpressJS (Integrate with Stripe, PayPal, e.t.c)

7.2 Front-end

- Tech stack:
  - ReacJS
  - TailwindCSS
- Design:

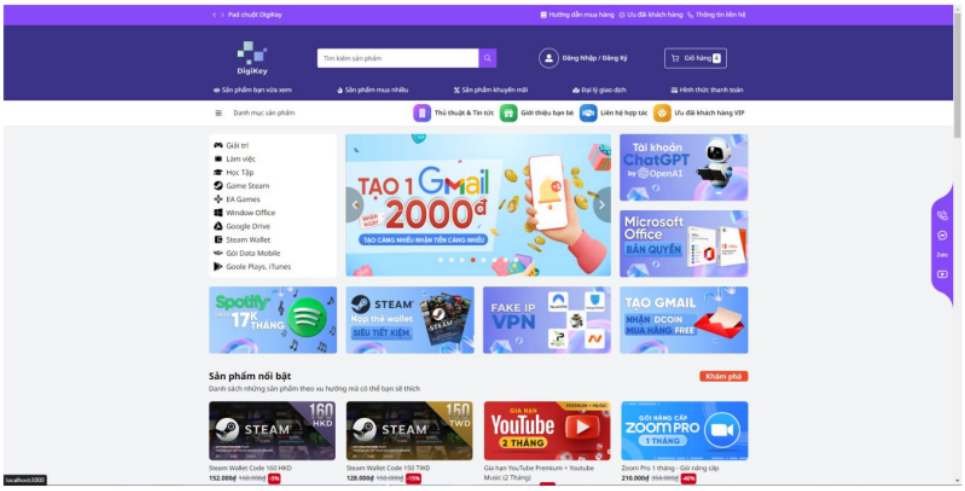


Figure 7.1. Home page - DigiKey

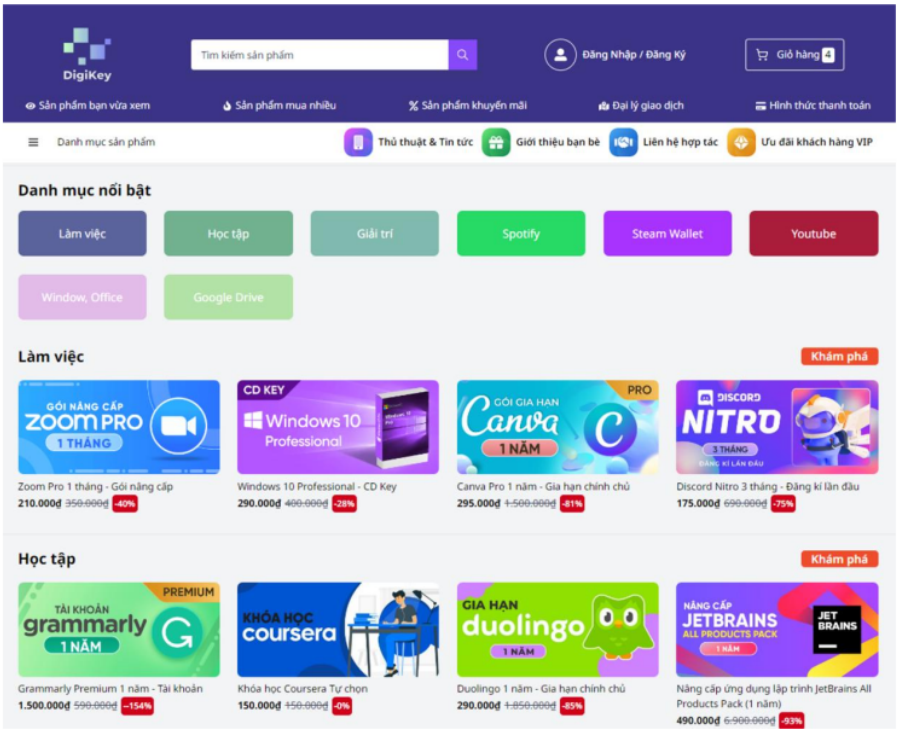


Figure 7.2. Best seller - DigiKey

8

Figure 7.3. Product detail 1 - DigiKey

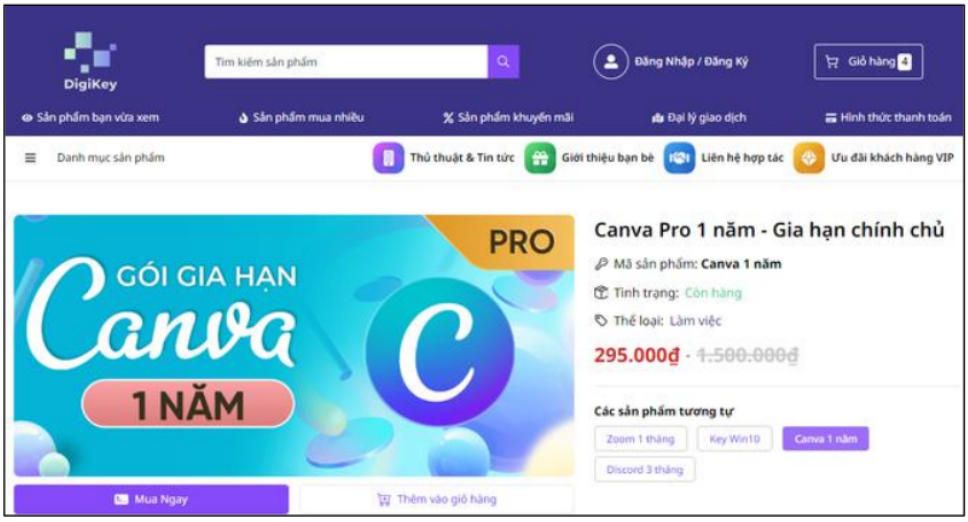


Figure 7.4. Product detail 2 - DigiKey

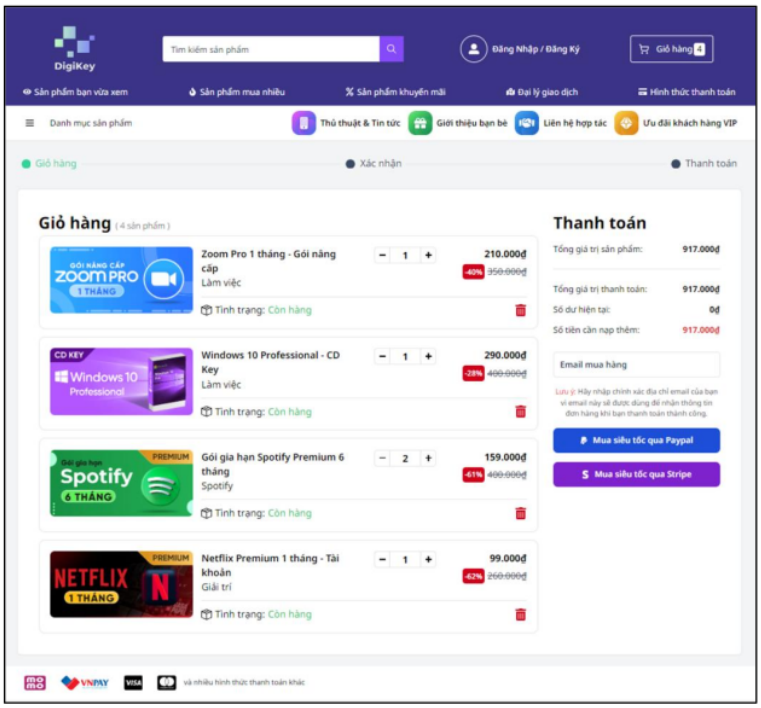


Figure 7.5. Cart detail - DigiKey

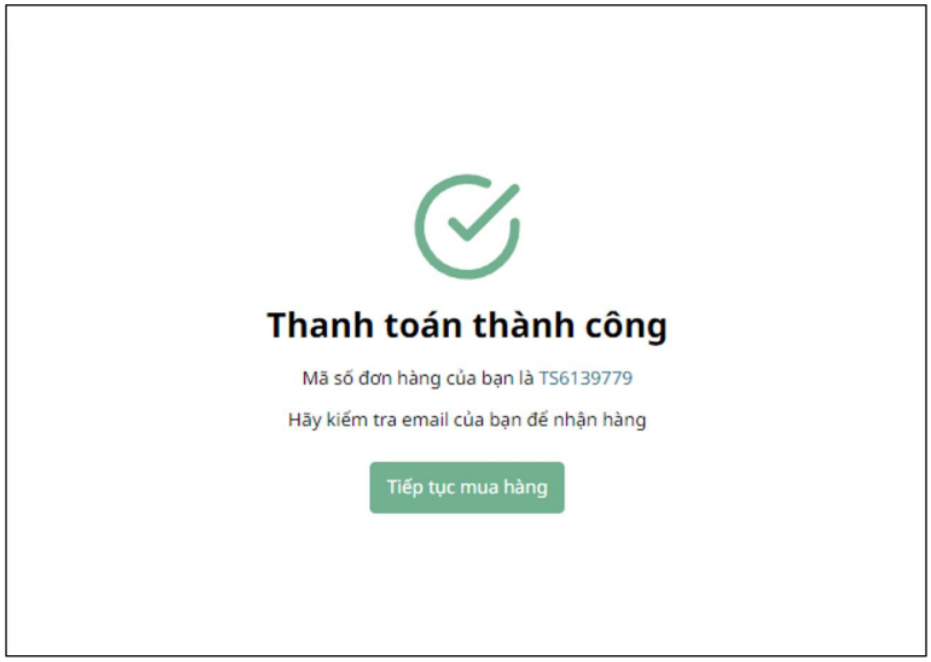


Figure 7.6. Payment success - DigiKey

7.3 Warehouse management service

- Tech stack:
  - NodeJS
  - ExpressJS
  - MySQL (Payment, ChartBrew)
  - Strapi
  - ChartBrew
  - Stripe
  - PayPal

Our back end service utilizes Strapi to provide a flexible and powerful content management system, ensuring efficient and secure handling of your digital products.

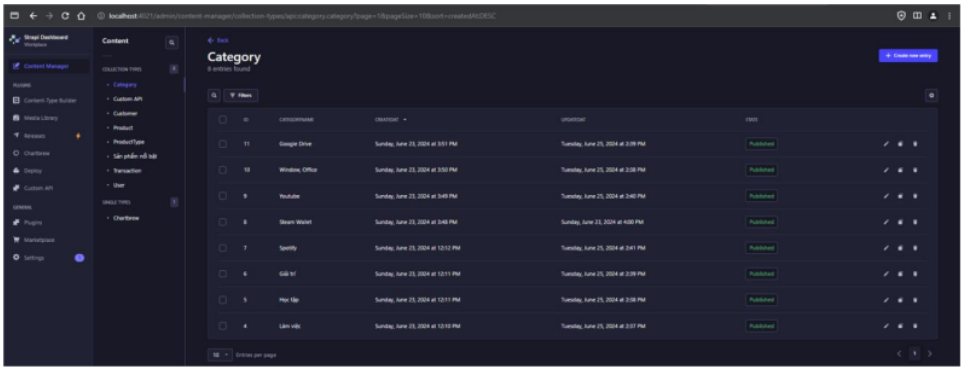


Figure 7.7. Administrator site - Strapi

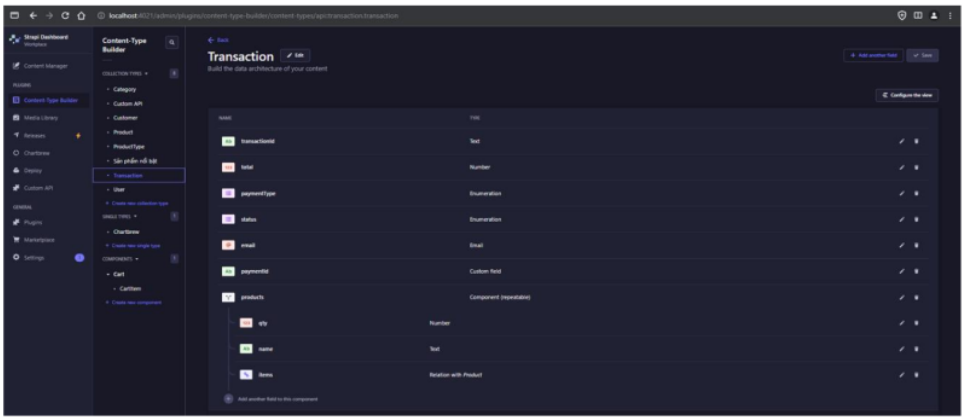


Figure 7.8. Data collection manger - Strapi

7.4 Payment service

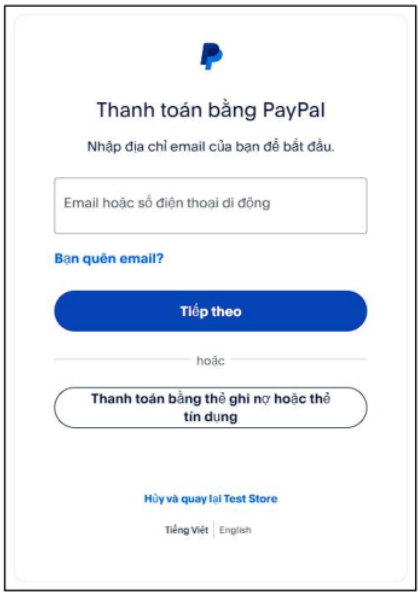


Figure 7.9. PayPal integrate - DigiKey

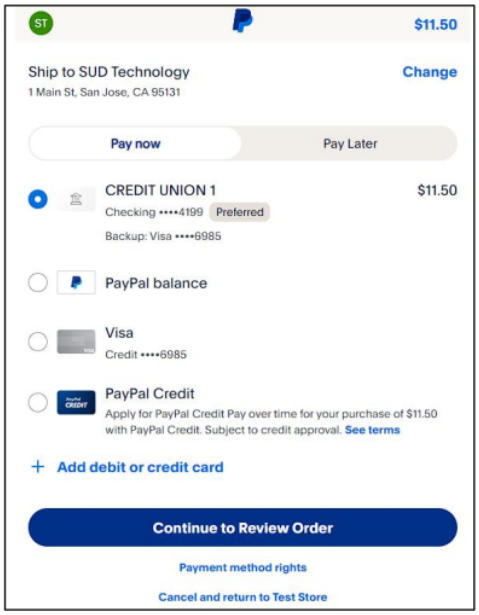


Figure 7.10. PayPal - DigiKey

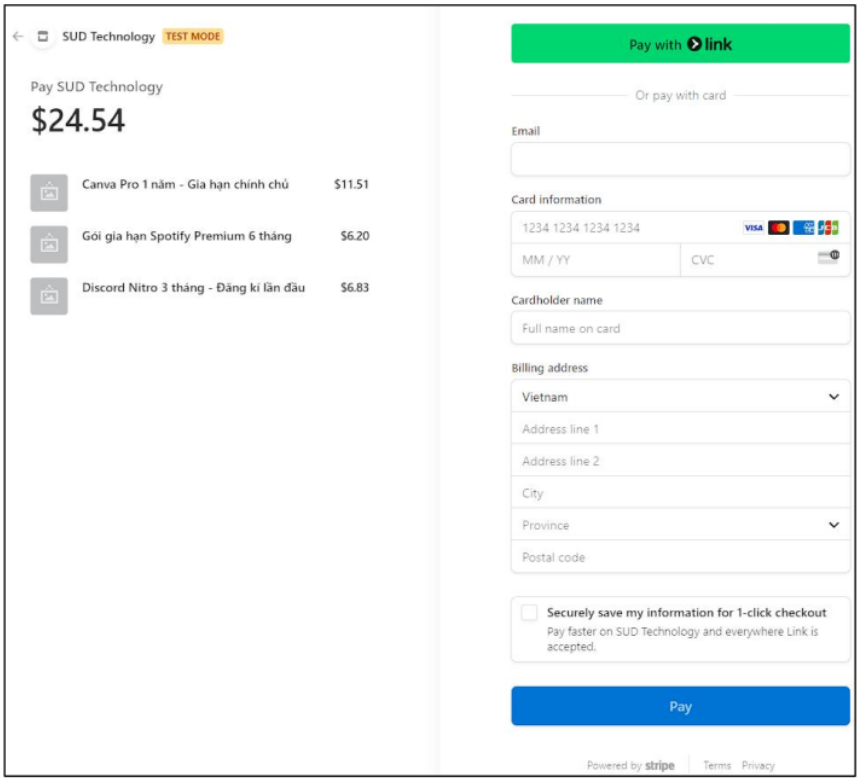


Figure 7.11. Stripe - DigiKey

7.5 Data Visualization service

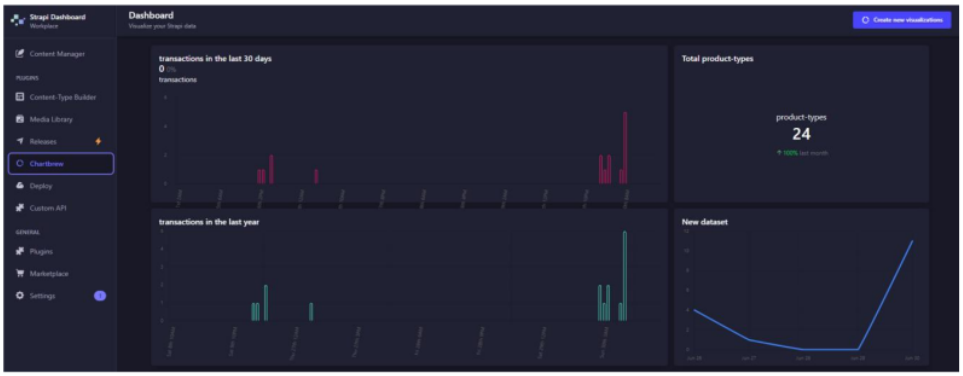


Figure 7.12 Dashboard - ChartBrew

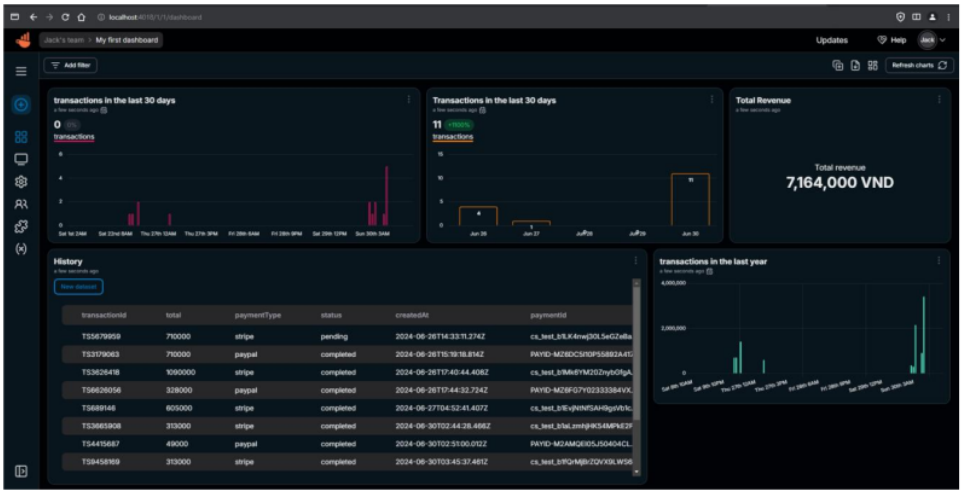


Figure 7.13 Revenue report - ChartBrew

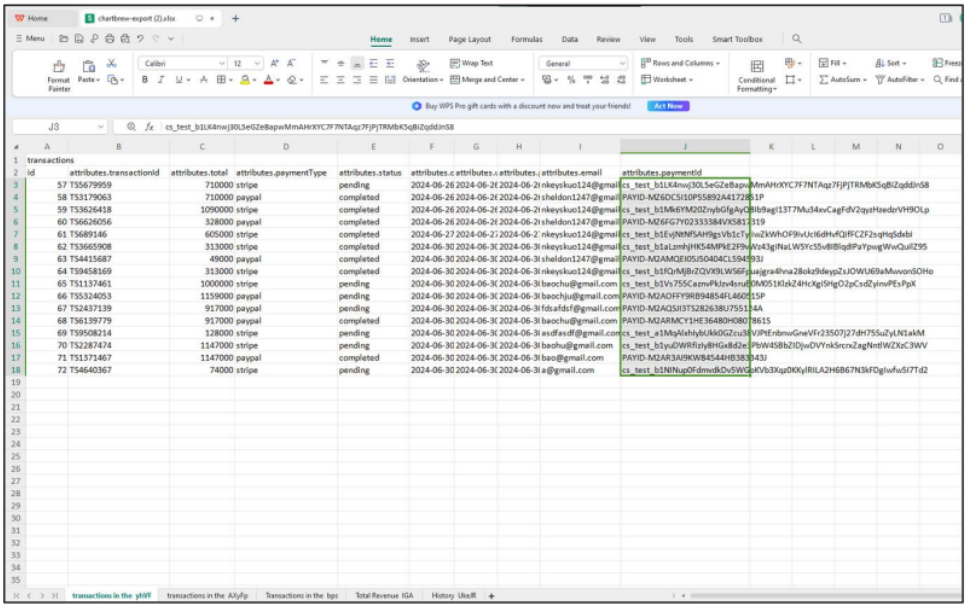


Figure 7.14 Export report - ChartBrew

CHAPTER 8. CONCLUSION

8.1 Conclusion

The DigiKey team has implement a very minimal, comprehensive customer journey that mainly focus on the importance of integrating up-to-date technologies to build a **MICROSERVICE** and efficient e-commerce platform. The project's architecture, which is microservices:

- ReactJS for Front-end
- Strapi for Warehouse management
- ChartBrew for data visualization
- Expresses and MySQL for payments

Has shown a a blend of flexibility, security and user-friendly interfaces. This tech stack ensures a smooth and clean customer experience for both types of tech-savvy or non-tech-savvy users. When the app still manage to keep the integrity and performance. The well-thought selection of technologies and the focus on low-code solutions have allowed for fast and clear development and easy management, making DigiKey a scalable and adaptable platform for future needs.

8.2 Future work

Moving forward, this is the to-do list for my project:

- Enhance security measures: Implement 2FA and end-to-end encryption can provide additional layers of protection to project users' data and transactions.
- AI integration:
    + Integrate with Chat Bot
    + Product recommendation based on user's preferences.
- Improve analytic and reporting: Innovating the features and the output of data visualization capabilities with more complex analytic and reporting tools, which allowing the user to access to deeper insights into user behavior, sales trends and enhancing the operation of their organization, also support ideas to improve better decision-making.

REFERENCES

English

Unlocking 10 Advantages of ReactJS, *CMC Global,* Link

React, *React Homepage,* Link

Strapi, The leading headless open-source CMS, Link

Strapi, the Developer Docs, Link

DACNTT_DigiKey_520H0011.pdf

ORIGINALITY REPORT

8%	6%	2%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	docs.strapi.io Internet Source	3%
2	Submitted to Ton Duc Thang University Student Paper	2%
3	repository.up.ac.za Internet Source	<1%
4	Submitted to De Montfort University Student Paper	<1%
5	Submitted to International Academy of New Zealand Student Paper	<1%
6	dev.to Internet Source	<1%
7	5dok.net Internet Source	<1%
8	Submitted to Informatics Education Limited Student Paper	<1%
9	lup.lub.lu.se Internet Source	<1%

10	www.vdocipher.com Internet Source	<1 %
11	thewebtier.com Internet Source	<1 %
12	Submitted to Wright State University Student Paper	<1 %
13	doi.org Internet Source	<1 %
14	www.upgrad.com Internet Source	<1 %
15	Submitted to Babes-Bolyai University Student Paper	<1 %
16	oa.upm.es Internet Source	<1 %
17	digitalarchive.boun.edu.tr Internet Source	<1 %
18	Submitted to University of Limerick Student Paper	<1 %
19	strapi.io Internet Source	<1 %
20	www.coursehero.com Internet Source	<1 %
21	bora.uib.no Internet Source	<1 %

22	hdl.handle.net Internet Source	<1%
23	umpir.ump.edu.my Internet Source	<1%
24	Sufyan bin Uzayr. "CSS Frameworks - The Ultimate Guide", CRC Press, 2023 Publication	<1%
25	Submitted to Wawasan Open University Student Paper	<1%
26	link.springer.com Internet Source	<1%
27	www.kindacode.com Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		