

Linh Nguyen

DEVELOPING AN ANIMAL RESCUE AND ADOPTION WEBSITE

Bachelor's thesis

Bachelor of Engineering

Information Technology

2024



South-Eastern Finland
University of Applied Sciences

Degree title	Bachelor of Engineering
Author(s)	Linh Nguyen
Thesis title	Developing an animal rescue and adoption website
Commissioned by	-
Year	2024
Pages	48 pages, 3 pages of appendices
Supervisor(s)	Tuomas Reijonen

ABSTRACT

The goal of this thesis was to develop a website for animal shelter organizations. This platform allows individuals to apply for pet adoption through an application form. Additionally, the website enables the organization to receive donations, either as one-time contributions or monthly.

The website was developed with the use of MERN stack which consists of MongoDB, ExpressJS, ReactJS, and NodeJS. The front-end is powered by ReactJS, while NodeJS and ExpressJS keep the back-end connected to the database in MongoDB. Communication between the front-end and backend occurs via REST API, allowing for autonomous development of both the front-end and back-end elements of the application. The theoretical background provided deep knowledge for the implementation part.

As a result, the website will serve as a place to connect animal rescue organizations with those who wish to adopt and support these relief activities. From here, we contribute to helping abandoned and injured animals that deserve attention and care from the community.

Keywords: Web development, user, MERN Stack, animal adoption.

CONTENTS

1	INTRODUCTION	4
2	THEORY BACKGROUND	5
2.1	JavaScript.....	5
2.2	React.js	7
2.2.1	Virtual DOM.....	7
2.2.2	JSX.....	8
2.3	Node.js	9
2.4	Express.js.....	10
2.5	MongoDB	11
2.6	REST API.....	13
3	IMPLEMENTATION	15
3.1	Website Overview	15
3.2	Back-end Implementation.....	16
3.2.1	API planning	16
3.2.2	Server implementation.....	18
3.3	Home page.....	27
3.4	Register page	28
3.5	Log in page.....	31
3.6	Adopt page.....	33
3.7	Animal detail page.....	35
3.8	Adoption application form	36
3.9	Donate page.....	38
3.10	Contact page	40
4	FUTURE DEVELOPMENT.....	41
5	CONCLUSION	42
	REFERENCES	44
	LIST OF FIGURES	47

1 INTRODUCTION

Since the inception of the Internet in the early 1980s, the world has undergone significant transformations. Presently, the Internet has evolved into an essential component of people's daily lives, appearing in all fields and wielding an undeniable influence. Concurrently, propelled by rapid technological advancements, the desire to establish websites pervades most organizations, enterprises, and individuals. Websites serve multifaceted purposes, not only enabling businesses to remain abreast in the high-tech era but also bolstering brand visibility. Serving as a visual representation of organizational information, websites facilitate seamless user access to projects, thereby fostering user interaction and facilitating a positive user experience.

The development of the Internet has brought about significant changes in many different aspects of society, reshaping the way we communicate, access information, and interact with the world, thereby creating impacts on social norms and behavior. Before there was no Internet, social issues about people, the environment, and animals seemed to be little known by people who did not read newspapers or listen to the radio. Now increased access to information and greater awareness of these issues have caused a shift in perception. What we want to talk about regarding this topic is humanity's concern for animals, especially homeless dogs and cats. More and more non-profit organizations are being established to support and rescue unfortunate animals, and people are willing to lend a helping hand to these creatures. By harnessing the power of technology to facilitate the adoption process and provide support for vulnerable animals. Further bridging the gap between digital innovation and altruistic endeavors, this project demonstrates the intersection of technological advancement and humanitarian values in our connected world.

Websites provide an efficient tool for updating information about animals at care stations quickly and accurately. This offers users a convenient and enjoyable

experience. The website has two roles: admin and users. The admin manages the website by adding, updating, and deleting animals' information. Users can view, search for animals that they want, fill out the adoption application, and donate through payment method (eg. PayPal).

This thesis is divided into four chapters, described as follows:

- Chapter 1 (Introduction): Give an overview of the thesis topic.
- Chapter 2 (Theory background): Overall description of the types of programming language and technology tools used in the project.
- Chapter 3 (Implementation): Show the process of the project.
- Chapter 4 (Conclusion): Summary of the results and future development of the project.
- Chapter 5 (Reference): Lists all the sources and references cited throughout the thesis.

2 THEORY BACKGROUND

The theoretical framework forms the foundation of any thesis, providing a structured foundation on which to build the research. This chapter elucidates the key concepts, principles, and tools used and relevant to the project.

2.1 JavaScript

JavaScript serves as a pivotal scripting language among the trio essential for website development. While HTML and CSS lay the foundation and design elements of a webpage, JavaScript empowers developers to infuse functionality and interactivity into their creations. This dynamic language enables the implementation of intricate web features, facilitating engaging user experiences beyond the capabilities of HTML or CSS alone. (DeGroat, 2019)

JavaScript is a versatile and widely used programming language that is built on the ECMAScript standard. It includes characteristics from procedural, imperative, weakly typed, and dynamic languages. Initially named Mocha and later LiveScript, it was ultimately branded JavaScript. It was developed by Brendan Eich at Netscape Communications Corporation in 1995. Despite its name, JavaScript is distinct from the Java programming language, offering different syntax, semantics, and use cases. (Evenson, 2023)

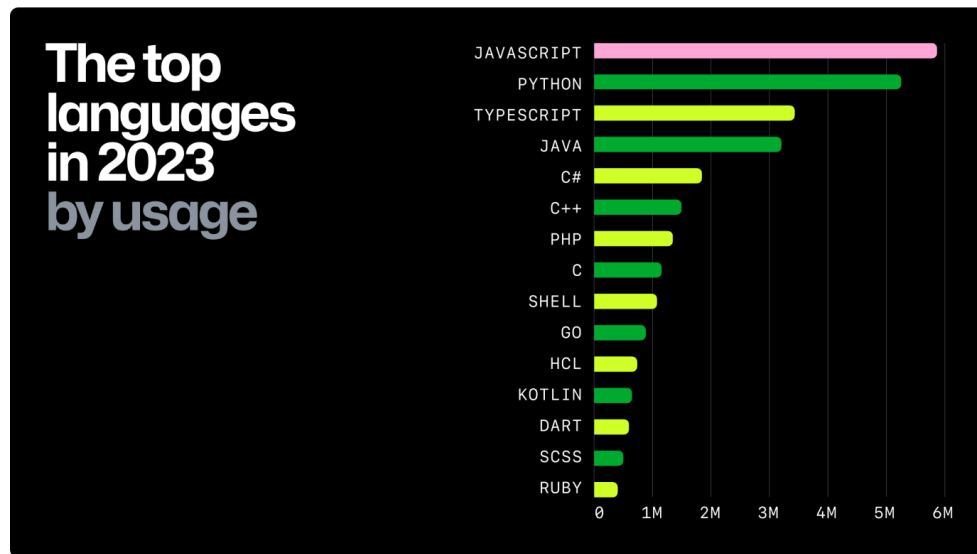


Figure 1. Rated of JavaScript language (Daigle, 2023)

In web development, JavaScript occupies a central position in contemporary web development, serving as the fundamental framework for creating dynamic and interactive web applications. JavaScript is used on two sides of a website: Client-side and server-side. Nevertheless, its prevalence predominantly stems from its role in client-side implementations. Additionally, JavaScript extends its reach beyond web applications, for example serving as a means to introduce interactivity to PDF documents and desktop widgets. Due to its versatility and wide adoption on the internet, JavaScript has emerged as the most used programming language by 2023 as shown in Figure 1. (Evenson, 2023)

2.2 React.js

React, often referred to as React.js or ReactJS, has gained widespread adoption in web development due to its declarative and component-based approach to building UIs. It is responsible for handling the components of the application's view layer. The story of React.js begins in 2010 when a Facebook engineer named Jordan Walke created an early version of React.js called "FaxJS,". Walke was inspired by XHP, an HTML component library for PHP. React.js first debuted on Facebook's News Feed in 2011, then expanded to Instagram in 2012, and became an open-sourced JavaScript library in 2013. (Anderson, 2023)

2.2.1 Virtual DOM

The DOM stands for Document Object Model and it is an Application Programming Interface (API). DOM manipulation is the core of modern interactive web pages and can dynamically change the content of a web page. The Virtual DOM (VDOM) is a programming concept where an ideal, or virtual, representation of a user interface (UI) is stored in memory and synchronized with the real DOM by a library like ReactDOM. This synchronization process, known as "reconciliation", aims to expedite UI updates within React applications. (Anderson, The History of React.js: A Story of Innovation and Community, 2023)

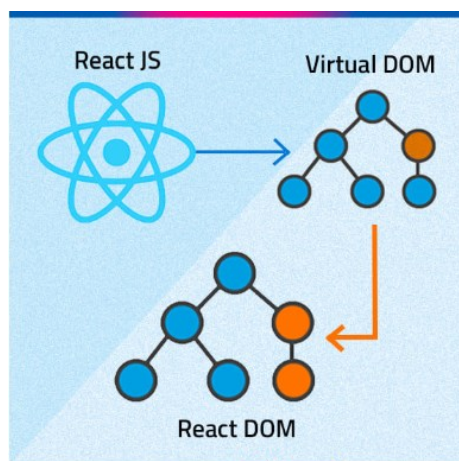


Figure 2. DOM process (CHAUHAN, 2023)

Figure 2 shows the process of how React updates the user interface efficiently. Initially, React creates a Virtual DOM, a lightweight, in-memory representation of a component's UI structure. When a component's state or props change, React constructs a new Virtual DOM and reconciles it with the previous one using a diffing algorithm to identify differences. To optimize performance, React batches multiple updates into a single process, minimizing changes to the real DOM. Once changes are determined, the real DOM is updated with minimal manipulations. React also reuses components and uses unique key props for list elements to streamline the process. Additionally, React's state management and Hooks system play a crucial role in identifying which components need re-rendering, ensuring efficient and responsive UI updates. This systematic process ensures that only the necessary changes are made, keeping the application fast and responsive. (CHAUHAN, 2023)

2.2.2 JSX

JSX, which stands for JavaScript XML, is a syntax extension for JavaScript that allows developers to write HTML-like code directly within JavaScript. It simplifies the process of creating and manipulating UI elements in React applications by combining HTML structure and JavaScript logic into a single file. JSX is used by enclosing HTML-like syntax within curly braces `{}` within JavaScript code.

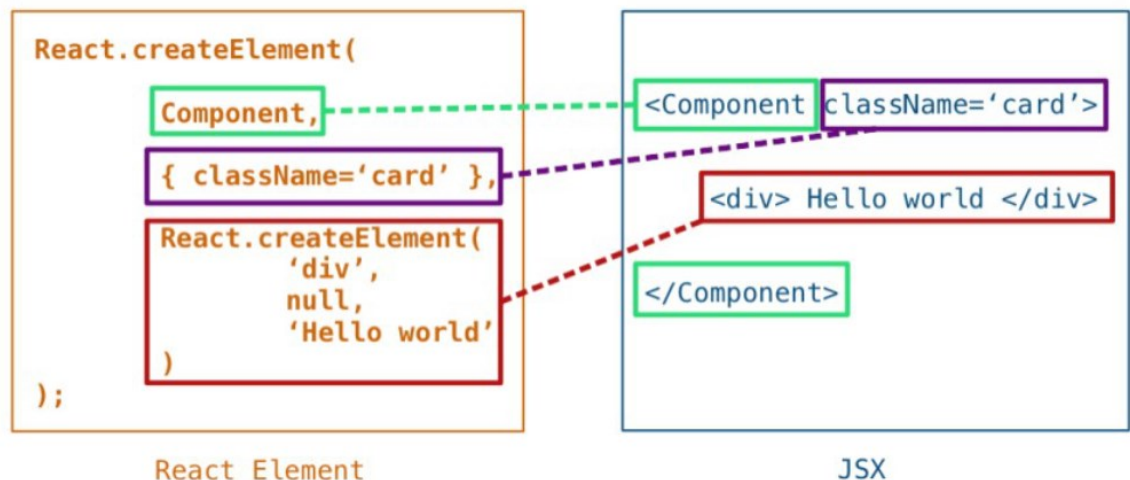


Figure 3. Using JSX in React (Decpk, 2021)

Figure 3 illustrates the difference between writing React code with and without JSX. The benefits of JSX include improved readability, easier integration of UI components with JavaScript logic, and enhanced developer productivity. Additionally, JSX is transpiled into regular JavaScript code by tools like Babel, making it compatible with all browsers. Overall, JSX streamlines the development process and enhances the maintainability of React applications. (Corbeel, 2024)

2.3 Node.js

Node.js is an open-source and cross-platform JavaScript runtime environment built on Chrome's V8 JavaScript engine. Node.js was developed in 2009 by Ryan Dahl, it has revolutionized server-side development by allowing developers to write JavaScript code for both client-side and server-side applications. Node.js is preferred for server-side development due to several compelling reasons. Firstly, it harnesses the power of Google Chrome's V8 engine, ensuring swift execution and high performance. Additionally, its extensive library of over 50,000 packages in the Node Package Manager (NPM) enables developers to integrate functionality into their applications, saving valuable time effortlessly.

Node.js excels in real-time and data-intensive applications thanks to its asynchronous, non-blocking nature, eliminating the need to wait for API responses. Furthermore, it facilitates reduced loading times for audio and video content by enabling seamless synchronization between client and server code bases. Lastly, its open-source nature and reliance on JavaScript make it easily accessible for developers already familiar with the language, simplifying the process of starting new projects with Node.js. (Sufiyan, 2023)

Some basic features of Node.js are the following:

- **Asynchronous in Nature and Event-driven:** Node.js servers never wait for data from APIs, instead swiftly moving on to the next API without blocking, making all Node.js APIs non-blocking. It employs an event-driven mechanism to receive and track responses, enhancing efficiency.

- **Single Threaded Architecture:** Node.js follows a single-threaded architecture with event looping, which improves scalability by efficiently handling concurrent requests without creating multiple threads. This architecture allows Node.js to process a large number of requests compared to traditional servers.
 - **Scalable:** Node.js addresses scalability concerns by efficiently managing concurrent requests using a cluster module for load balancing across active CPU cores. It can partition applications horizontally, catering to distinct app versions and client preferences through child processes.
 - **Quick Execution Time for Code:** Leveraging the V8 JavaScript runtime engine, Node.js ensures fast execution time for code processing. This engine, also used by Google Chrome, accelerates the runtime process, enhancing overall performance.
 - **Compatibility on Cross Platforms:** Node.js is compatible with various systems such as Windows, UNIX, LINUX, MacOS, and mobile devices. It can be seamlessly paired with appropriate packages to generate self-sufficient executions.
 - **Uses JavaScript:** Node.js utilizes JavaScript, a familiar language for most developers, simplifying development tasks and accelerating the learning curve for newcomers.
 - **Fast Data Streaming:** Node.js processes data transmitted to different streams at a rapid pace, improving overall speed and efficiency in data and video streaming applications.
 - **No Buffering:** Node.js applications do not buffer data, resulting in efficient data processing and improved performance.
- (Sufiyan, 2023)

2.4 Express.js

Express.js is a lightweight framework built on top of Node.js web server functionality, aimed at simplifying APIs and enhancing features. It streamlines

application organization through middleware and routing. It offers utilities for rendering dynamic HTTP objects.

Express.js has several advantages, including simplicity and minimalism, making it easy to learn and implement server setups, route definitions, and HTTP request handling efficiently. Its flexibility allows for customization based on individual preferences, offering a strict yet adaptable application architecture. Express.js benefits from a rich middleware ecosystem, facilitating the integration of various functionalities like authentication and error handling. With a focus on scalability, Express.js caters to projects of all sizes, thanks to its lightweight, asynchronous, and event-driven architecture, capable of handling large volumes of requests. Additionally, its active community support ensures regular updates and comprehensive documentation, contributing to its continuous growth and improvement. (Express.js Tutorial, 2024)

2.5 MongoDB

MongoDB was founded in 2007 by Dwight Merriman, Eliot Horowitz, and Kevin Ryan – the team behind DoubleClick. MongoDB is a popular open-source NoSQL database management system designed for scalability, flexibility, and performance. It uses a document-oriented data model, storing data in flexible, JSON-like documents, making it ideal for handling unstructured or semi-structured data.

Features of MongoDB are the following:

- **Document-oriented:** MongoDB stores data in documents rather than tables, enhancing flexibility with key-value pairs and unique document IDs.
- **Schema-less Database:** MongoDB allows collections to contain varying document types, sizes, and fields, providing unparalleled flexibility compared to relational databases.

- Scalability: MongoDB achieves horizontal scalability through sharding, distributing data across multiple servers, and enabling the addition of new machines to run databases.
- Indexing: MongoDB indexes every document field with primary and secondary indices, facilitating efficient data retrieval and search operations.
- Aggregation: MongoDB supports operations on grouped data, offering aggregation pipelines, map-reduce functions, and single-purpose aggregation methods for obtaining computed results.
- High Performance: With features like scalability, indexing, and replication, MongoDB delivers exceptional performance and data persistence, surpassing other databases in terms of efficiency.

(Top Features of MongoDB, n.d)

MongoDB's advantages include schema-less flexibility, facilitating dynamic data structures without predefined schemas. Its document-oriented approach allows for versatile query operations, tailored to document characteristics. Performance optimization is simplified due to MongoDB's architecture, enhancing data management efficiency. Efficient memory utilization ensures fast data access and overall system performance. Horizontal scaling via sharding enables distributed data storage across multiple servers, optimizing data retrieval. MongoDB's ease of maintenance is attributed to its flexible schema and streamlined optimization processes. Replication and workload distribution features ensure data availability and system reliability by spreading tasks across multiple instances. (MongoDB Advantages & Disadvantages, 2023)

Despite MongoDB's many strengths, it also carries a few drawbacks that warrant consideration. Its limited transaction scope and hindered complex operations across multiple data pieces. While offering ACID compliance at the document level, MongoDB lacks full ACID compliance across collections, posing challenges for intricate transactional requirements. Limited join capabilities compared to relational databases necessitate manual operations, potentially affecting

performance. Data redundancy and increased memory usage arise from storing key names with each value pair, impacting efficiency. A document size limit of 16 MB imposes constraints on handling larger data sets, requiring alternative approaches. Additionally, MongoDB restricts document nesting to 100 levels, influencing data organization and structure within documents. (MongoDB Advantages & Disadvantages, 2023)

2.6 REST API

REST stands for Representational State Transfer, it is an architectural style for designing networked applications, particularly web services, that adhere to certain constraints to ensure scalability, performance, and simplicity. API stands for Application Programming Interface. It is a set of rules, protocols, and tools that allow different software applications to communicate with each other. APIs are used to enable interaction between different software systems, allowing them to share data and functionality seamlessly. (REST API: Key Concepts, Best Practices, and Benefits, 2022)

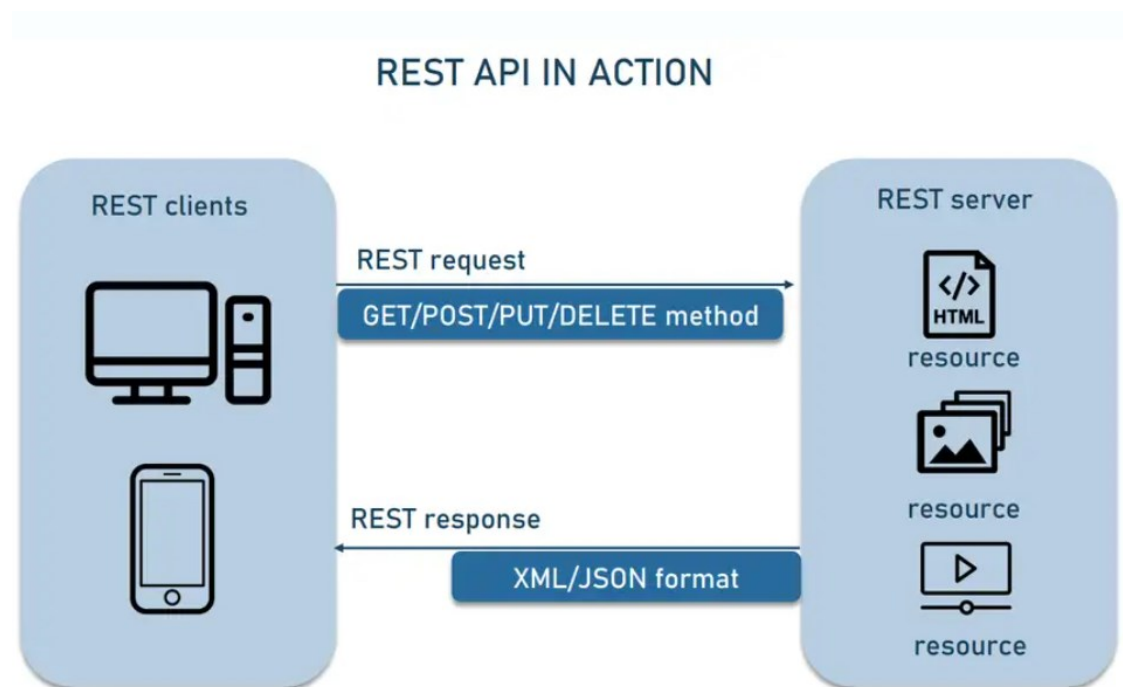


Figure 4. How REST API processes between clients and servers (REST API: Key Concepts, Best Practices, and Benefits, 2022)

Figure 4 illustrates the core functionality of the REST API, which mediates communication between clients and servers while adhering to the principles of REST. This enables clients to interact with server resources via standard HTTP methods like GET, POST, PUT, and DELETE, facilitating scalable and stateless exchange of data and functionality across diverse software systems over the internet.

GET: The GET method requests data from a specified resource. It retrieves data without altering the server's state or causing any side effects. It is typically used for fetching information such as retrieving a resource's details or a collection of resources.

POST: The POST method is used to send data to a server to process a specified resource. It can be utilized for various purposes, such as creating a new resource, updating an existing resource, or triggering server-side processing.

PUT: The PUT method is used to update or replace an existing resource or create a new resource if it doesn't exist at the specified URI. It sends data to the server to update or replace the state of the target resource with the provided representation.

DELETE: The DELETE method is used to remove a specified resource from the server. It deletes the resource identified by the given URI, effectively removing it from the server.

PATCH: The PATCH method is used to apply partial modifications to a resource. It sends data to the server to update the resource's state with the provided partial representation, allowing clients to update specific fields of a resource without replacing the entire resource.

(Nolle, 2023)

3 IMPLEMENTATION

This section is divided into two parts: the website overview provides an overview of how the website operates, and the back-end implementation demonstrates how the project was constructed.

3.1 Website overview

The website contains two basic roles, including admin and user. The admin here is the shelter organization, they can create, update, or delete animals' information. With the application form, they can check all the forms with users' information and then do the process for adoption and reply to them through email. The users can read animals' information, fill in the form, and donate to the shelter organization.

The website is called **Petner**, this name embodies the essence of the platform dedicated to animal adoption. Inspired by the profound link between humans and animals, I imagined a space where a "pet" goes beyond companionship to become a treasured friend and valued "partner" in life's journey. "Petner" hopes to renew the traditional concept of pet ownership by establishing ties based on care, compassion, and companionship, as well as recognizing the vital role that animals play in our lives.



Figure 5. Website logo

Figure 5 shows the logo for this website, featuring the word "Petner" alongside a paw icon, which embodies the essence of our platform. Paws represent the feet of animals, especially dogs and cats, this iconic image will give the website a recognizable look.

The main functions of this website are the following :

- Register and log in: The users can create an account and log in to this website.
- Users can read animal information by categories.
- Adoption application: To adopt an animal, the users must fill in the form and provide all required information.
- Donate: Users can donate to the organization through the donation function.

3.2 Back-end implementation

In the back-end implementation section, we delve into the crucial components of constructing a robust and efficient server architecture. This segment encompasses two primary facets: API planning and server implementation.

3.2.1 API planning

The API planning involves meticulously designing the endpoints, method, and data input that facilitate communication between the client side and the server. This process entails defining the functionalities, requests, and responses expected from the server, laying the groundwork for seamless interaction with the user interface.

Table 1. API plan

Endpoint	Method	Input	Description
----------	--------	-------	-------------

/register	POST	Name, email, password	Register a new user account.
/login	POST	Email, password	Sign in to an existing user account.
/user/:id	GET	Id	Get user by id.
/user/	GET	-	Get all users.
/user/:id	PUT	Id	Update user by id.
/user/:id	DELETE	Id	Delete user by id.
/category/create	POST	Name of the category	Create a new category.
/category/	GET	-	Load all categories.
/category/:id	GET	Id	Load one category by id.
/category/:id	PUT	Id	Update one category by id.
/category/:id	DELETE	Id	Delete one category by id.
/adopt/	GET	-	Get the list of all animals.
/adopt/:id	GET	Id	Get one animal's information by id.
/adopt/animals	POST	Name, category, breed, age, sex, size, description.	Admin creates new animal.
adopt/animals/:id	PUT	-	Admin updates an existing animal by id.
adopt/animals/:id	DELETE	-	Admin deletes an existing animal by id.
/application/	POST	Fill in the adoption form.	Users create an application form.
/application/	GET	-	Admin gets all applications.
/donate	POST	-	Do a donate
/donate/success	GET	-	Payment success
/donate/cancel	GET	-	Cancel payment

Table 1 offers a diverse range of endpoints and methods crucial for the project's operations. They feature clear and distinct endpoints, methods, and descriptions, aimed at facilitating smooth integration into the project. This detailed API plan serves as the groundwork for effortless interaction between users and the application, fostering efficient communication and functionality.

3.2.2 Server implementation

To start with the server side, there are some modules and frameworks that need to be installed: `express`, `nodemon`, `body-parser`, `cors`, `dotenv`, `jsonwebtoken`, and `mongoose`. We use the command with the structured *`npm install`* and the name of the modules or frameworks.

Express.js is a Node.js framework designed for building web applications and APIs. It simplifies server-side development by providing features like routing, middleware, templating, and debugging. (Sharma, 2023)

Nodemon is a handy utility that automatically monitors changes in the Node.js application files. It detects modifications and swiftly restarts the server, eliminating the need for manual intervention. This streamlined process enhances efficiency during development, as we can focus on coding without the interruption of stopping and restarting the server after each update. (Kouchi, 2024)

CORS stands for Cross-Origin Resource Sharing, which means sharing resources from many different sources. Same-origin policy is a very important security mechanism, implemented in the browser to block JS code that can make requests to sources other than the source from which it is returned. Clients from different sources cannot access each other's resources without permission. (Armstrong, 2021)

Dotenv is a zero-dependency module that loads environment variables from a `.env` file into `process.env`. The Twelve-Factor App technique is the foundation for

storing settings in the environment apart from code. In this project, the `.env` file contains sensitive information including database credentials and JWT secret keys. (Dotenv, 2024)

Bcryptjs is a JavaScript module designed for securely storing passwords in backend applications. Bcrypt is favored for its ability to enhance password security by making it computationally intensive to generate and compare hashes, thereby protecting user credentials from unauthorized access. It is widely used to ensure the safe storage of passwords in databases and is considered a reliable method for safeguarding sensitive user information. (Grigutyte, 2023)

JSON Web Token (JWT) is a compact and self-contained method for securely transmitting information between parties as a JSON object. It is commonly used for authorization purposes, allowing users to access routes and resources based on the permissions granted by the token. Additionally, JWTs are useful for securely exchanging information between parties, as they can be signed to verify the integrity of the claims contained within them. They offer benefits such as compactness, security, and ease of use across different platforms, making them a popular choice for information exchange and authentication in web and mobile applications. (Introduction to JSON Web Tokens, n.d)

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It is used to manage the relationships between data, provides schema validation, and is used to translate between objects in code and MongoDB. (Sazib, 2022)

To build the server side, we have to own a database and in this project I use MongoDB. The database has four collections: animal, user, application, and category shown in Figure 6:

Petner

animals
applications
categories
users

Figure 6. Collections on MongoDB Database

- Animals: Store all details of the animals in the shelter.
- Applications: Store all pet adoption applications.
- Categories: Main categories of animals.
- Users: Store all data of users including name, email, and password.

MongoDB Atlas is the cloud server I use to connect with the database MongoDB. There are two main ways to connect the database to the server, they are “connect to my application” and “access your data through tools” like MongoDB compass, MongoDB Shell, MongoDB for VS Code, and Atlas SQL. In this project, I will connect the database using the “connect to my application” way which accesses the Atlas data using MongoDB’s native driver Node.js by connection string MONGODB_URI with the format:
mongodb+srv://<username>:<password>@[database].cjtuddt.mongodb.net/?retryWrites=true&w=majority&appName=[database].

```
console.log(process.env.MONGODB_URI)

const connectDatabase = async () => {
  try {
    await mongoose.connect(process.env.MONGODB_URI, { dbName: 'Petner' })
    console.log(`MongoDB Connected`)
  } catch (error) {
    console.log(error.message);
  }
}
```

Figure 7. MongoDB connecting by using mongoose

Mongoose is imported to establish a connection to the MongoDB database as shown in Figure 7. The argument `console.log(process.env.MONGODB_URI)` logs the MongoDB connection string to the console for debugging purposes. The `connectDatabase` function, declared asynchronously, attempts to establish a connection to the MongoDB database specified in the connection string stored in the `.env` file with the database name set to "Petner". Once connected successfully, it will write "MongoDB Connected" to the console. In case of any connection error, the function will catch the error and log the corresponding message.

After establishing the database foundation, the schema is a JSON object that delineates the structure and content of the collection in the project. Many types will be defined, for example, the type of data, required, enum, etc.

```
const animalSchema = new mongoose.Schema({
  name: { type: String, required: true },
  category:{type: String, required: true, ref: 'Category'},
  age: { type: Number, default: 0 },
  gender: { type: String, enum: ['Male','Female'],required: true },
  description: { type: String, required: true },
  image: { type: String, required: true },
}, {
  timestamps: true
},{ collection: 'animals' });
```

Figure 8. Animal Schema

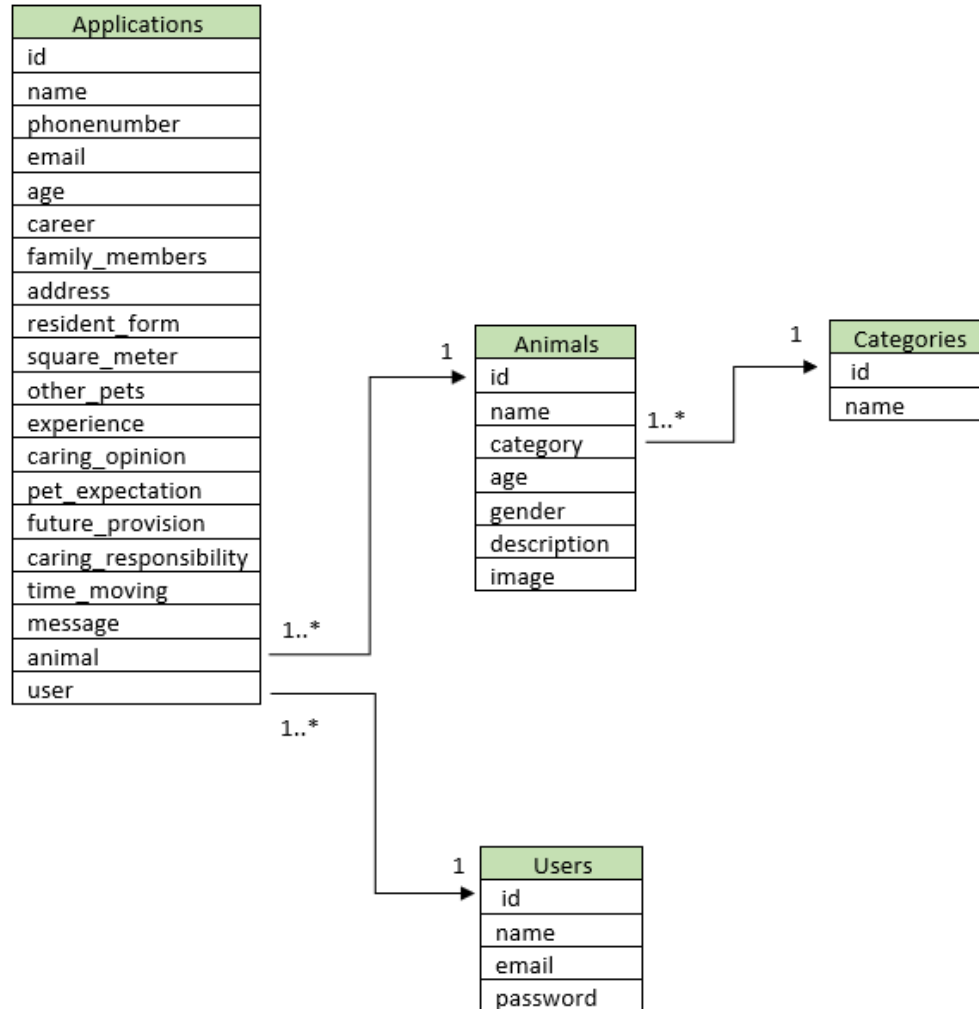
In figure 8, the animal schema is the collection of the types for the data that are stored for animals such as name, category of this animal, age, gender, description, and image for each pet. To use the Schema in the collection, we have to convert it into Model, and figure 9 displays how to convert the `animalSchema` to Animal model.

```
const Animal = mongoose.model('Animal', animalSchema);
```

Figure 9. Animal Model

The remaining collections include categories, applications, and users that have a similar structure to the animal schema and animal model. For more details, table 2 provides a comprehensive overview of the project's data structure and the relationships that bind them together.

Table 2: Relational table of the project



After defining the database collections and the models for each of them, I utilize Express to define routes for handling various operations related to collection within the application. I use four basic API methods: POST, GET, PUT, and DELETE following the table 1.

```

router.post('/create', categoryController.createCategory);
router.get('/', categoryController.getCategory);
router.get('/:id', categoryController.getCategoryById);
router.put('/:id', categoryController.updateCategory);
router.delete('/:id', categoryController.deleteCategory);

```

Figure 10. Category Route

Figure 10 displays how to create the route for the category collection based on the API planning in table 1. Using the POST method to create the category of animal following the path `/category/create`, the GET method to load the category, the PUT method to update the existing data, and the DELETE method to delete the existing category. The `/:id` after the prefixes is the method required input to work with certain data. In this situation, this `/:id` becomes the keyword for the route to edit the information by input the id of specified category. The route forwards requests to the appropriate controller function for processing.

```

exports.createCategory = async (req, res, next) => {
  try {
    const newCategory = await Category.create(req.body);
    res.status(201).json({
      status: 'Create new category successfully!',
      category: {
        _id: newCategory._id,
        name: newCategory.name,
      }
    });
  } catch (error) {
    next(error);
  }
};

```

Figure 11. Using POST method to create a new category

Figure 11 shows the code defining a controller function named `createCategory`, tasked with creating a new category in the database. Inside the function, there's a *try-catch* block. This is a common pattern for error handling in JavaScript. The code inside the try block is executed, and if any errors occur, they are caught and handled in the catch block. It begins by using `Category.create()` method, which takes the data provided in the request body `req.body`. When the `Category.create()` returns a promise, the `await` keyword is used to pause the execution of the function until the category creation operation is complete. If the

category creation is successful, it responds with a status code of 201 (Created) and sends a JSON response containing details about the newly created category, including its id and name.

```
exports.createAnimal = async (req, res, next) => {
  try {
    const newAnimal = await Animal.create(req.body);
    res.status(201).json({
      status: 'Created new animal information successfully',
      animal: {
        _id: newAnimal._id,
        name: newAnimal.name,
        category: newAnimal.category,
        age: newAnimal.age,
        gender: newAnimal.gender,
        description: newAnimal.description,
        image: newAnimal.image
      }
    });
  } catch (error) {
    next(error);
  }
};
```

Figure 12. Using POST method to create a new animal

The same POST method is used to create an animal with a list of requested data including the name, category, age, gender, description and image of each animal as shown in figure 12. In case of any errors during the process, such as database connection issues or validation errors, the error is caught and passed to the error-handling middleware using the next() function for further handling.

```
class createError extends Error {
  constructor(message, statusCode) {
    super(message);
    this.statusCode = statusCode;
    this.status = `${statusCode}`.startsWith('4') ? 'fail' : 'error';
    Error.captureStackTrace(this, this.constructor);
  }
}
module.exports = createError;
```

Figure 13. Custom error by createError class

Figure 13 displays code that defines a custom error class named *createError*, extending the built-in JavaScript Error class. It accepts two parameters in its constructor: *message* and *statusCode*, which are used to set properties of the error instance. The *statusCode* property is used to determine whether the error status is 'fail' or 'error', based on whether the status code falls within the range of client errors (4xx). Additionally, it utilizes *Error.captureStackTrace()* to capture and store a stack trace for the error instance. The *createError* class is exported at the end of the file, so it can be imported and used in other parts of my application to throw custom errors such as when an animal is not found for showing, updating, or deleting shown in figure 13 below. These errors are then caught in the catch block of the try-catch statement and passed to the next middleware function, which is likely an error-handling middleware that sends an appropriate response to the client.

```
exports.getallAnimal = async (req, res, next) => {
  try {
    const animals = await Animal.find();

    if (!animals || animals.length === 0) {
      return next(new createError('No animals found', 404));
    }
    res.status(200).json({
      status: 'success',
      animals: animals
    });
  } catch (error) {
    next(error);
  }
};
```

Figure 14. GET method to load all animals

Figure 14 shows the code defining a controller function named *getAnimal*, responsible for managing requests to retrieve all animals from the database. To get all categories, I use the controller function name *getCategory* with the same structure. The code uses Mongoose's *find* method to get all documents from the categories collection in the database. The *await* keyword is used to pause the execution of the function until the promise returned by *Animal.find()* is resolved. If the animal array is empty, it creates a new *createError* object with an error

message `No animals found` and a status code of 404. Opposite, it sends a response with a status code of 200 and a JSON body containing a status property with a value of 'success' and an animal property with the animals array. In case of any errors occurring during the execution of the function, the *catch* block calls the *next* function with the caught error, passing it to the error-handling middleware for further processing.

```
const animal = await Animal.findById(req.params.id);
```

Figure 15. Get a single animal by id

Each category and animal has their unique ID so in case one wants to get data on one specified category or animal, use the *findById* method as following figure 15 above. The *.findById(req.params.id)* is a Mongoose method that finds a single document in the collection by its ID. The ID is passed as a parameter in the URL of the HTTP request, and it's accessed through *req.params.id*. In the code above, I use this function to get a single animal by ID, to get another document in the other collections like categories, applications, or users, I also use the same structure.

```
exports.updateAnimal = async (req, res, next) => {
  try {
    const updatedAnimal = await Animal.findByIdAndUpdate(req.params.id, req.body, {
      new: true,
      runValidators: true
    });
    if (!updatedAnimal) {
      return next(new createError('Animal not found', 404));
    }
    res.status(200).json({
      status: 'success',
      data: {
        animal: updatedAnimal
      }
    });
  } catch (error) {
    next(error);
  }
};
```

Figure 16. Update animal information

During the operation and management of the rescue center, animal information can be continuously updated and edited. To update new information for animals, I use the method *.findByIdAndUpdate* shown in figure 16. This method finds a

document in the collection by its ID and updates it. The ID is passed as a parameter in the URL of the HTTP request, and it's accessed through `req.params.id`. The new data for the document is taken from `req.body`, which is the body of the HTTP request. The options object `{ new: true, runValidators: true }` shows that the updated document should be returned (`new: true`) and that the update should be validated against the schema (`runValidators: true`). If an animal with the given ID is found and updated in the database, *updatedAnimal* will be an object representing that animal. If no animal is found, *updatedAnimal* will return "Animal not found".

```
exports.deleteAnimal = async (req, res, next) => {
  try {
    const deletedAnimal = await Animal.findByIdAndDelete(req.params.id);
    if (!deletedAnimal) {
      return next(new createError('Animal not found', 404));
    }
    res.status(200).json({
      status: 'success',
      message: 'Animal deleted successfully'
    });
  } catch (error) {
    next(error);
  }
};
```

Figure 17. Delete animal

If organizations have the ability to create, receive, and update data, they also possess the capability to delete it. Figure 17 demonstrates the method *findByIdAndDelete(req.params.id)*, which locates a specific document in the collection by its ID and removes it. This function is typically executed for animals that have been adopted or are no longer under the care of the organization.

3.3 Home page

The home page serves as the gateway to a website, offering users a welcoming interface and intuitive navigation options. It provides quick access to essential features like account registration, login, and the latest updates. With a clean design and user-friendly layout, the home page ensures visitors can easily find

the information they need. Engaging visuals and clear call-to-action buttons guide users through their journey, enhancing their overall experience on the site.

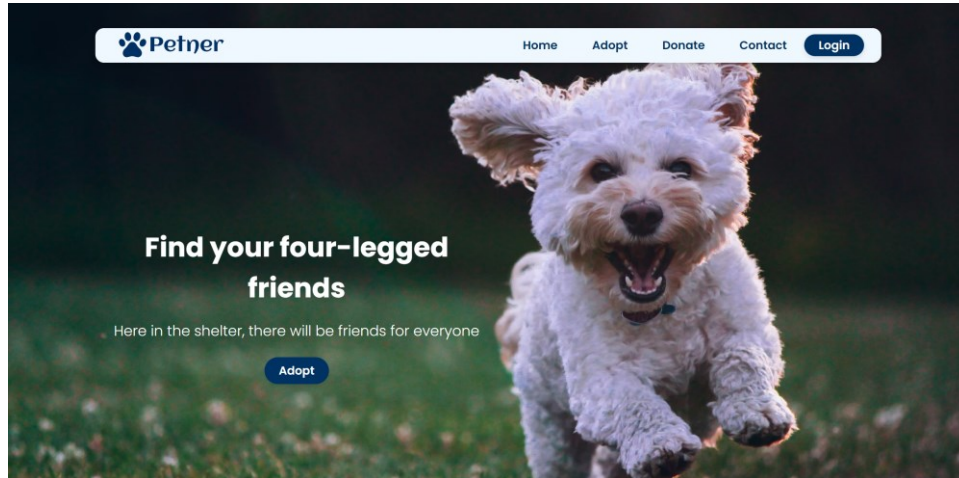


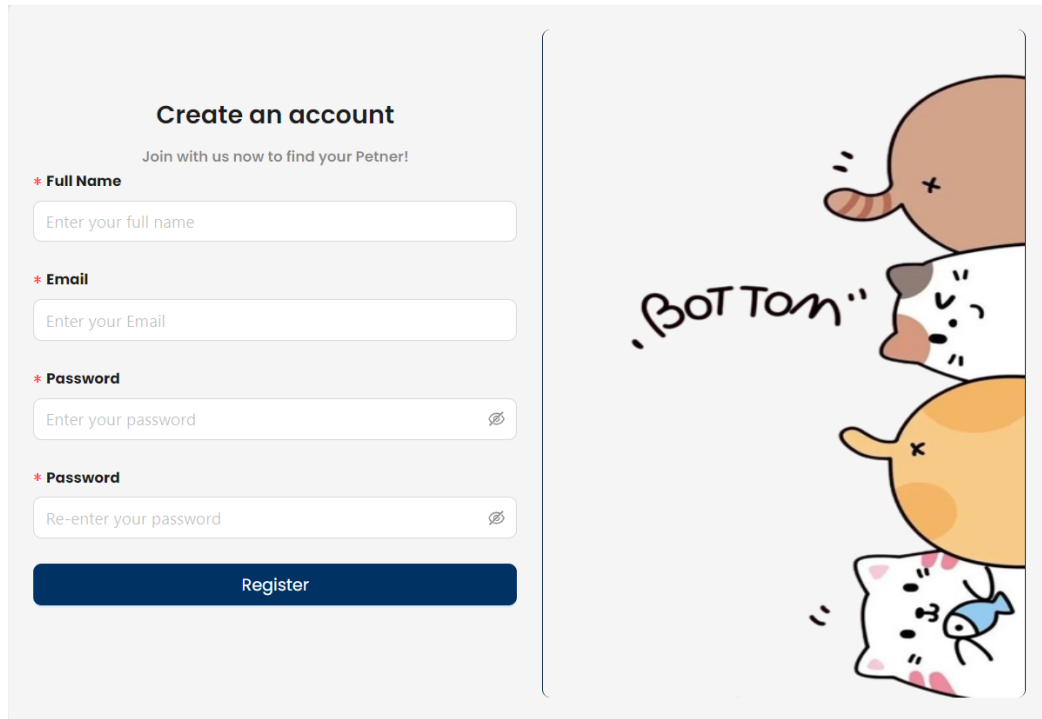
Figure 18. Home page

Figure 18 presents users with a welcoming interface and a toolbar offering seamless navigation to various functions of the website. Five primary functions are accessible: "Home" directs users to the main page, "Adopt" facilitates the animal adoption process, "Donate" enables users to contribute financially to the organization, "Contact" provides organization information, and "Login" allows user identification. This intuitive layout ensures easy access to key features, enhancing user experience and engagement with the platform.

3.4 Register page

The register page is designed for users to initiate the creation of a new account. To proceed with the registration process, users are required to input their full name, email, and password. Once all the necessary information is provided, users can click on the "Register" button to submit their data to the backend endpoint `/register` for validation. Users need to adhere to the specified criteria for email formatting and password complexity to successfully create their

accounts. In case of any incorrect inputs, users will receive immediate error messages displayed above the register button, prompting them to correct their entries. Figure 19 illustrates the register page.



Create an account

Join with us now to find your Petner!

* **Full Name**

Enter your full name

* **Email**

Enter your Email

* **Password**

Enter your password

* **Password**

Re-enter your password

Register

Bottom

Figure 19. Register page

On the server side, using the POST method to request the `/register` the controller first checks if a user with the provided email already exists in the database. The `.findOne` method will check the existing email in the users collection, if a user with the same email is found, it creates a new error with the message “User already exists”. Otherwise, the `User.create` function will create a new user with the data in `req.body` including name, email, and password.

```

userSchema.pre('save', async function(next) {
  if (!this.isModified('password')) {
    return next();
  }
  try {
    const salt = await bcrypt.genSalt(10);
    this.password = await bcrypt.hash(this.password, salt);
    next();
  } catch (error) {
    next(error);
  }
});

```

Figure 20. Hash the password

Figure 20 shows the execution of the *userSchema.pre('save')* middleware function before saving the user to the database. This function verifies whether the password field has been modified. If the password has been modified, it generates a salt using *bcrypt.genSalt*, then hashes the password using *bcrypt.hash* with the generated salt. The resulting hash is then stored as the password for the user instance before proceeding with the save operation.

```

exports.register = async (req, res, next) => {
  try{
    const user= await User.findOne({email:req.body.email});
    if(user){
      return next(new createError('User already exists', 400));
    }
    const newUser = await User.create(req.body);

    const token = jwt.sign(
      {id: newUser._id},
      process.env.tokenSecret, {expiresIn: '30d'});

    res.status(201).json({
      status: 'success',
      message: 'User registered successfully',
      token,
      user: {
        _id: newUser._id,
        name: newUser.name,
        email: newUser.email,
        isAdmin: newUser.isAdmin,
      },
    });
  } catch(error){
    next(error);
  }
};

```

Figure 21. Register new users by POST method

Figure 21 displays the code for the register function, which handles user registration. After successfully creating the user, a JSON Web Token (JWT) is crafted using the user's ID and a secret token. This token, coupled with user details, is then provided in the response. Utilizing the *jwt.sign* method, the code incorporates the user's ID (`newUser._id`) into the token payload, signs it with a designated secret key loaded from the `.env` file, and establishes a 30-day expiration period (`expiresIn: '30d'`). Subsequently, the generated token is stored in the `token` variable for authentication purposes.

3.5 Log in page

After registering for an account, users can enter their email and password to authenticate their identity on subsequent visits. Figure 22 illustrates the log in page.

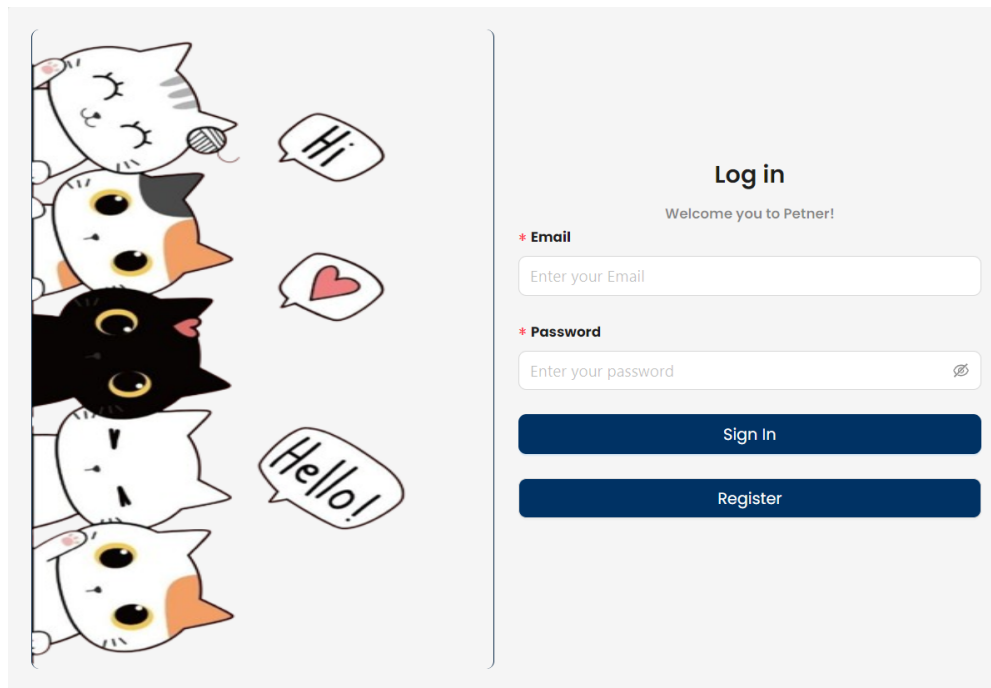


Figure 22. Log in page

Upon submitting the login credentials, the data is sent to the backend endpoint `/login` for validation using the POST method. Similar to the registration process, the controller first queries the database to find a user with the provided email using the `.findOne` method. If no user is found, an error message "User not found" is generated and returned to the user interface. If a user is found, the controller proceeds to verify the entered password against the hashed password stored in the database using the `userSchema.methods.matchPassword` function, which meticulously compares the provided password with its hashed equivalent. If the passwords match, a JSON Web Token (JWT) is generated using the user's ID and a secret token, and this token, along with the user details, is returned in the response. Conversely, if the passwords do not match, an error message "Invalid email or password" is returned, signaling an unsuccessful login attempt.

```
userSchema.methods.matchPassword = async function (enteredPassword) {  
  return await bcrypt.compare(enteredPassword, this.password);  
}
```

Figure 23. Compare method in the login process

Figure 23 showcases the code snippet depicting the `matchPassword` method within the `userSchema` of a Mongoose model. This asynchronous function is responsible for comparing the entered password with its hashed counterpart in the database using `bcrypt.compare()`. The function execution is paused using the `await` keyword until the comparison completes, which then returns either `true` or `false`. This method plays a critical role in authenticating user credentials during the login process.

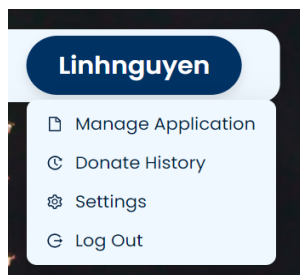


Figure 24. User menu

Upon successful login, the user's name will automatically be displayed. Figure 24 shows a user menu with several functions, including managing applications, donation history, settings, and logout. In the current version of this project, only the logout function is operational, with plans to enhance the other features in future updates.

3.6 Adopt page

The main function of this website is to facilitate animal adoption, allowing users to access and find a companion that matches their preferences. Through an intuitive interface, users can browse available pets, filter by categories such as All, Cat, or Dog, and view detailed information about each animal. This streamlined process ensures that finding a new pet is both efficient and enjoyable.

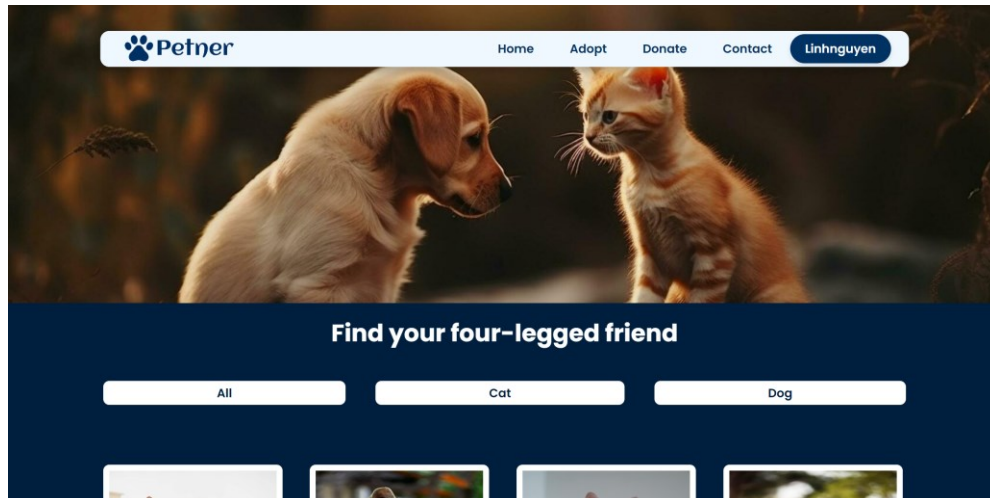


Figure 25. Adopt page

Figure 25 illustrates the Adopt page, which provides a user-friendly interface for potential pet adopters to browse a list of pets available at the shelter. This React component utilizes state management to store pet data fetched from an API endpoint. Upon loading, it retrieves the pet data using Axios and updates both the complete and filtered lists of pets. The page includes a dynamic filtering function

that enhances the browsing experience by allowing users to refine their search by category—All, Cat, or Dog—through clearly labeled buttons. When a filter button is clicked, the list of pets updates to show only those matching the selected category. Figure 26 shows all dogs in the Dog category, demonstrating how the filter function personalizes the user experience.

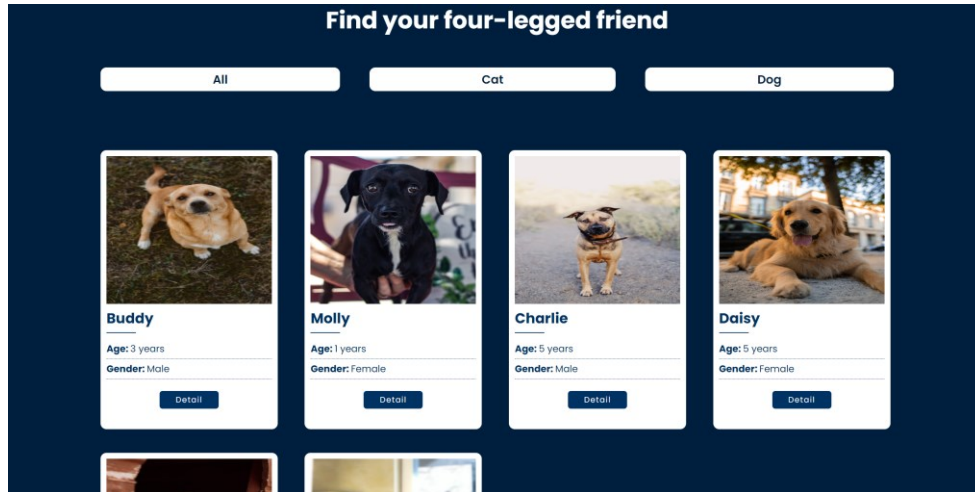


Figure 26. Filter dog category

To improve surfing performance, the website contains a Pagination component that divides the list of pets into digestible groups of twelve on each page. This component uses state management to track the current page and determines the total number of pages required depending on the amount of elements on each page. It cuts the array of objects and displays only the appropriate portion for the current page. The produced output maps over the current items to create individual item displays, each with an image, name, age, gender, and a detail button that takes the user to a detailed view of the selected item via React Router's `useNavigate` hook.

Users can flip between pages using the radio buttons below the item display. The buttons are designed to identify the current page and are deactivated when already chosen, providing clear visual feedback and improving the user experience. The combination of dynamic filtering and effective pagination allows

visitors to traverse the available pets, locate important information, and have a smooth browsing experience.

3.7 Animal detail page

The animal detail page provides a comprehensive view of individual pets available for adoption, enhancing the user's ability to make informed decisions about adopting a pet. This React component utilizes `useState` and `useEffect` hooks to fetch and display detailed information about a selected pet from the API endpoint. Upon loading, the pet's details, such as name, age, gender, and description, are displayed, along with an image. The page also includes a section on adoption fees, specifying what is covered for dogs and cats. This structured presentation ensures that users have all the necessary information to consider before adopting.



Figure 27. Animal detail page

Figure 27 shows the animal details page. The website has an easy-to-navigate style with a "Her Detail" or "His Detail" headline dependent on the pet's gender and a button to start the adoption process. When users click the "Adopt This Little Friend!" button, a dialog box displays, describing crucial adoption considerations and actions. This dialog emphasizes key points such as age requirements, the

importance of thoughtful decision-making in pet adoption, and considerations regarding personal resources and living arrangements. Additionally, it underscores the absence of a trial period and encourages a thorough review of the pet's description to ensure compatibility with the prospective adopter's family and lifestyle. It concludes by stressing the significance of readiness to receive the pet upon contacting the organization due to limited foster home availability. After evaluating the factors and determining a match, users may begin filling out the adoption application.

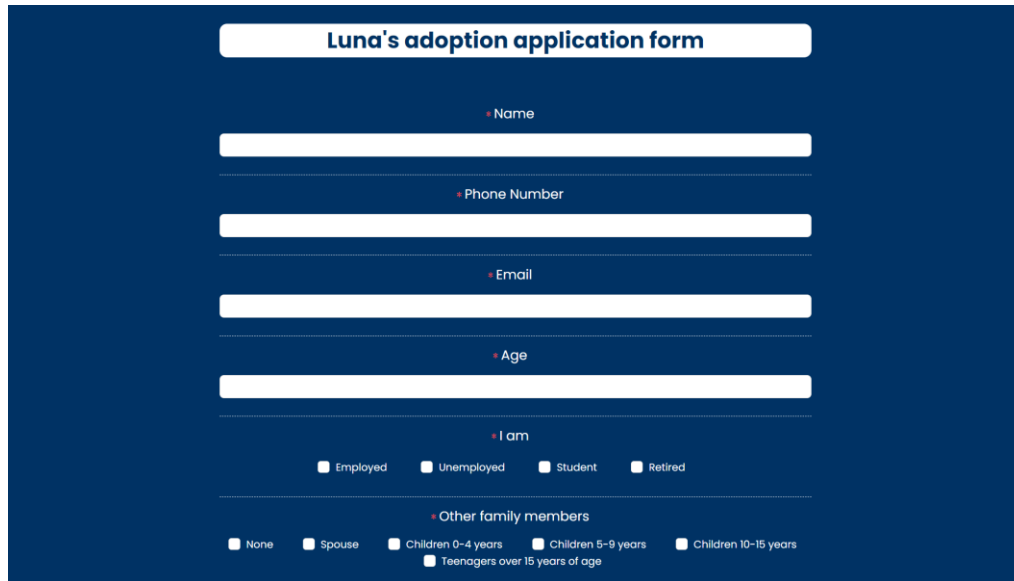
3.8 Adoption application form

The main function of this website is to adopt animals by filling out an adoption application. Continue using the *create()* function to create a new application using the POST method, which creates a new adoption application asynchronously using the data provided in the request body as shown in figure 28.

```
exports.createApplication = async (req, res, next) => {
  try {
    const newApplication = await Application.create(req.body);
    res.status(201).json({
      status: 'success',
      message:
        'Congratulations, your application has been successfully submitted! We will contact you soon.',
      application: {
        _id: newApplication._id,
        name: newApplication.name,
        phoneNumber: newApplication.phoneNumber,
        email: newApplication.email,
        age: newApplication.age,
        career: newApplication.career,
        family_members: newApplication.family_members,
        address: newApplication.address,
        resident_form: newApplication.resident_form,
        square_meter: newApplication.square_meter,
        other_pets: newApplication.other_pets,
        experience: newApplication.experience,
        caring_opinion: newApplication.caring_opinion,
        pet_expectation: newApplication.pet_expectation,
        future_provision: newApplication.future_provision,
        caring_responsibility: newApplication.caring_responsibility,
        time_moving: newApplication.time_moving,
        message: newApplication.message,
        animalId: newApplication.animal,
        userId: newApplication.user,
      }
    });
  } catch (error) {
    next(error);
  }
};
```

Figure 28. Create a new application by POST method

Figure 29 shows the application form where the applicant must be over 20 years old and must complete all required information. Once the application is successfully created and stored in the database, the application responds with a JSON object containing a success status and a congratulatory message confirming delivery to the screen.



Luna's adoption application form

* Name

* Phone Number

* Email

* Age

* I am

☐ Employed ☐ Unemployed ☐ Student ☐ Retired

* Other family members

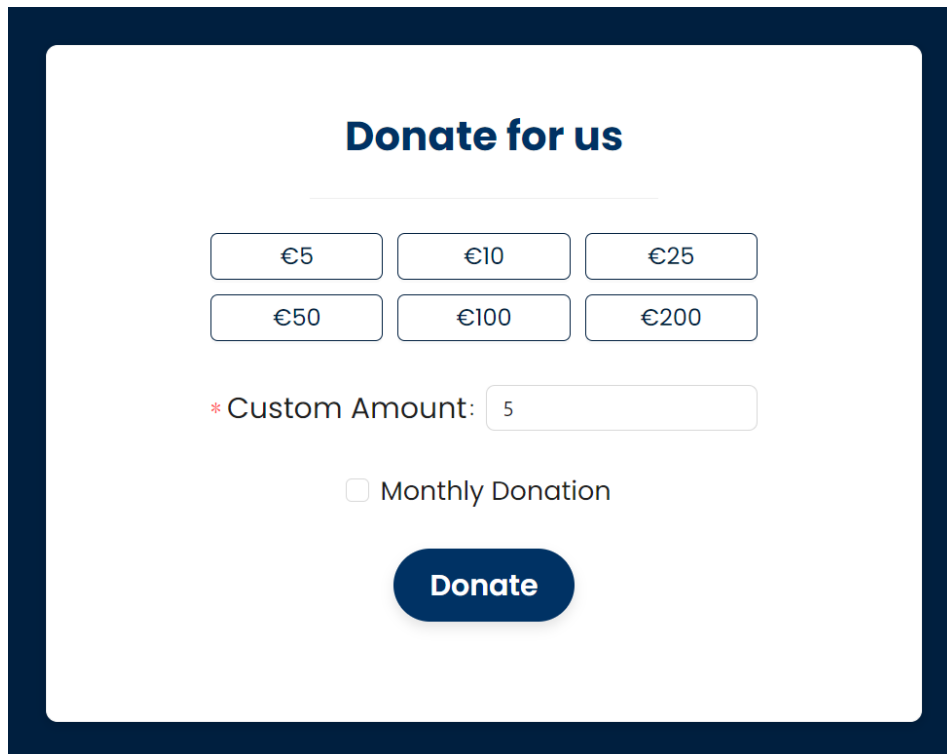
☐ None ☐ Spouse ☐ Children 0-4 years ☐ Children 5-9 years ☐ Children 10-15 years ☐ Teenagers over 15 years of age

Figure 29. Adoption application form

Once completed, applications will be sent directly to the organization's email. The organization then proceeds with the adoption process by thoroughly examining each application and evaluating the applicant's details to determine if they are a match for the desired animal. Following this assessment, the organization will notify the results to the applicant via email and conduct a phone interview. In this communication, additional questions may be included to determine the candidate's compatibility with the responsibilities associated with pet care, thereby facilitating the process. comprehensive review.

3.9 Donate page

The website facilitates both pet adoption and donations to support the organization's mission. Users have the option to contribute financially through PayPal, selecting between one-time or monthly donations. This functionality not only fosters community engagement but also provides vital resources for maintaining the shelter's operations. By leveraging PayPal's secure payment platform, users can conveniently support the organization's cause, ensuring the well-being of sheltered animals and enabling continued outreach efforts. In Figure 30, the donation interface is depicted, showing the user a structured layout of suggested donation amounts between 5 and 200 euros. Additionally, users have the flexibility to manually enter a custom donation amount.

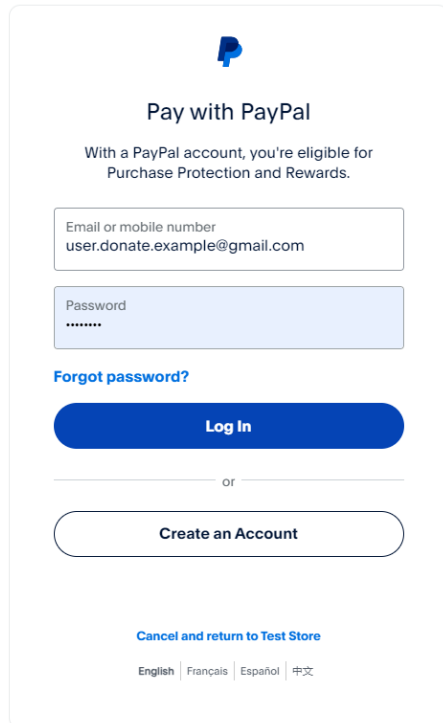


The image shows a donation form titled "Donate for us" in a dark blue header. Below the title, there are six buttons for suggested donation amounts: €5, €10, €25, €50, €100, and €200, arranged in two rows of three. Below these buttons is a label "* Custom Amount:" followed by a text input field containing the number "5". Underneath the input field is a checkbox labeled "Monthly Donation". At the bottom of the form is a large, dark blue button with the word "Donate" in white text.

Figure 30. Donation box

The project uses the PayPal REST SDK in combination with the PayPal sandbox environment to provide secure and frictionless contribution transactions. By interacting with the PayPal API, the server-side code orchestrates payment generation and execution, assuring reliable payment processing. On the client

side, the DonatePage component allows users to enter their contribution selections and starts the donation process via asynchronous communication with the server.



The image shows a PayPal login and account creation interface. At the top is the PayPal logo. Below it, the text "Pay with PayPal" is displayed, followed by a note: "With a PayPal account, you're eligible for Purchase Protection and Rewards." There are two input fields: "Email or mobile number" with the example "user.donate.example@gmail.com" and "Password" with masked characters. A link "Forgot password?" is located below the password field. A blue "Log In" button is positioned below the input fields. Below the button is a horizontal line with the word "or" in the center. Underneath the line is a "Create an Account" button. At the bottom, there is a link "Cancel and return to Test Store" and a row of language options: "English", "Français", "Español", and "中文".

Figure 31. Make a donation

Figure 31 shows the PayPal payment interface, users can log in to their PayPal account and initiate payment.

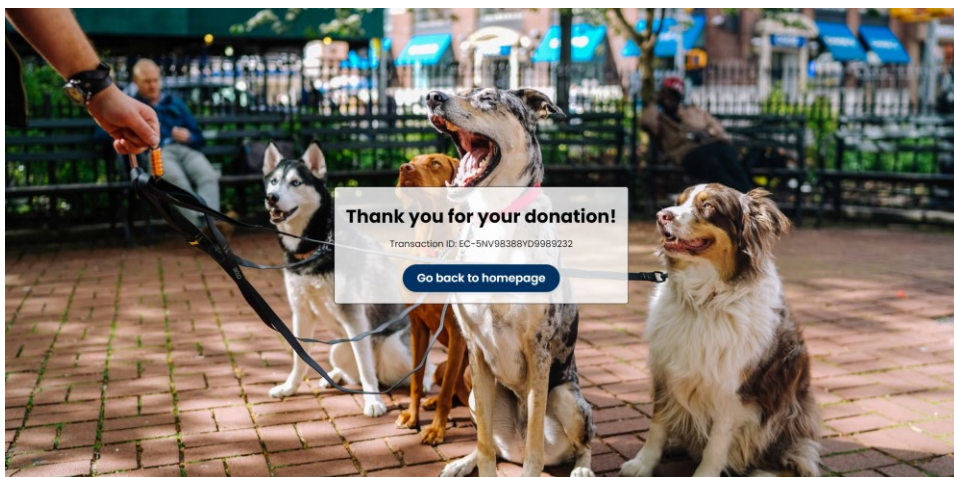


Figure 32. Payment successfully

The DonateSuccess and DonateCancel components are designed to handle the outcomes of a user's donation attempt on the website. Both components utilize React Router's `useLocation` and `useNavigate` hooks to manage URL query parameters and facilitate navigation. When a contribution is successfully completed, the DonateSuccess component displays a thank-you message and the transaction ID in figure 32; on the other hand, the DonateCancel component notifies the user that their donation has been canceled and keeps the same transaction ID for reference in figure 33.

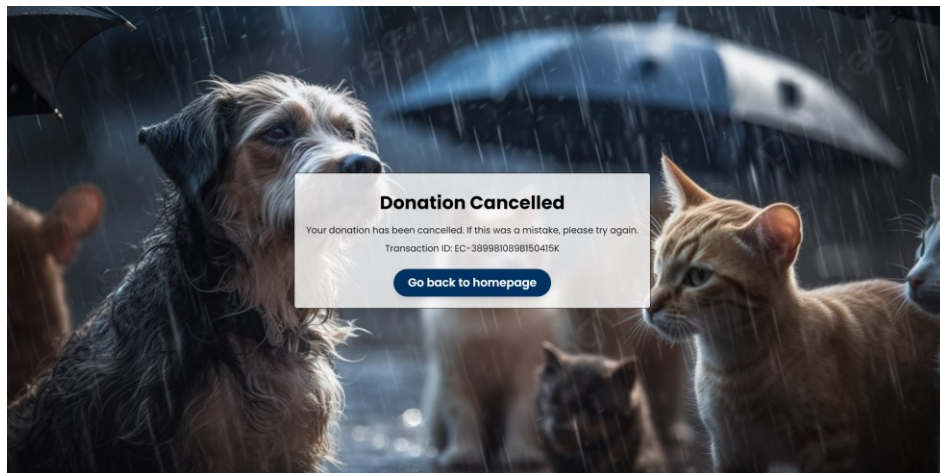


Figure 33. Payment cancelled

Both components offer a button that redirects users back to the homepage, ensuring a seamless user experience regardless of the donation outcome. This implementation ensures that users receive clear feedback on their donation attempts and have an easy way to return to the main page.

3.10 Contact page

The Contact component serves as a crucial interface for fostering communication between users and the animal shelter. By providing a user-friendly form and displaying contact information prominently, the component ensures that users can easily reach out with their inquiries, feedback, or suggestions. This interaction is vital for maintaining transparency, addressing concerns, and encouraging community engagement. It reflects the shelter's commitment to

being accessible and responsive, ultimately helping to build trust and strengthen relationships with potential adopters, donors, and volunteers.

CONTACT

Sent us a message 🐾

Please get in touch via the form or find our contact information below. Your feedback, questions and suggestions are important to the maintenance and growth of the organization.

☎ (+358)255987563

✉ petner@gmail.com

📍 Raviradantie 777, Mikkeli, Finland

Your name

Enter your name

Phone number

Enter your phone number

Your email

Enter your email

Your message

Enter your message

Send →

Figure 34. Contact form

Figure 34 shows the contact form. When a user fills out the form and hits send, the fetch API sends the form data to the Web3Forms endpoint via a POST request. This data includes user input such as name, phone number, email, and message. The Web3Forms service then processes this data and sends it directly to the organization's email, ensuring that all communications are received promptly. This setup not only simplifies the submission process but also enhances reliability and security, ensuring that user messages are efficiently conveyed to the shelter's team.

4 FUTURE DEVELOPMENT

In addition to the existing functionalities, the project has significant potential for future development. One key aspect of this is the implementation of an admin dashboard, empowering managers to perform operations with animal and user data more intuitively. This dashboard will streamline tasks such as managing animal profiles, updating pet statuses, and evaluating adoption applications. Moreover, integrating an online adoption application processing feature will

revolutionize the adoption process. Admins will be able to view efficiently, process, and give feedback on applications directly through the platform, eliminating the need for manual email correspondence. Furthermore, implementing an organization's volunteer management system will streamline volunteer recruitment and coordination, and encourage community participation and support for the organization. Additionally, the project could incorporate an e-commerce platform to sell pet-related items, providing a new revenue stream to support the shelter's financial needs. The donation feature can be extended to include various payment methods such as credit and debit cards, digital wallets, and bank transfers. This provides users with flexibility in choosing their preferred payment method.

In improving user experience, detailed development of features for User Profiles such as allowing them to manage adoption application forms, track online application progress, view donation history contributions, review the application history, and set up the profile information. Hosting virtual adoption events is a feature similar to donating, but here users can choose exactly which animal and for how long they want to support financially. Moreover, providing educational resources about pet ownership is an idea that the organization partners with schools, providing hands-on and educational experiences for students about loving animals. Through these efforts, the project will not only enhance user engagement but also streamline organizational processes, ultimately contributing to the success of animal adoptions and welfare efforts.

5 CONCLUSION

The goal of this project was to build an animal adoption website for animal rescue organizations, thereby finding more connections between pets and people, to spread kindness and love for animals.

This endeavor utilized a comprehensive stack of technologies, necessitating proficiency in ReactJS, NodeJS, HTML, and CSS. NodeJS functioned as the server-side component, facilitating the exchange of data between the client and the database. Meanwhile, ReactJS served as the client-side application, empowering users to engage with the website's interface seamlessly. HTML was employed to structure the front-end design, providing the foundational framework for the website's layout. Additionally, CSS was utilized to enhance the visual appeal of the web pages, ensuring an aesthetically pleasing user experience. MongoDB was employed to manage the database operations effectively.

Looking ahead, the future of the project holds immense promise and potential for growth. By continuing to innovate and expand upon its existing functionalities, the project aims to further solidify its position as a valuable resource for animal rescue organizations and prospective adopters alike. Through the implementation of advanced features such as the admin dashboard, online adoption application processing, and enhanced user profiles, the project will continue to evolve to meet the changing needs of its users and stakeholders. Additionally, by fostering community engagement through social sharing capabilities and virtual adoption events, the project will continue to raise awareness about animal welfare and facilitate meaningful connections between pets and people. With a steadfast commitment to excellence and ongoing development, the project is poised to make a lasting impact in the realm of animal adoption and welfare for years to come.

REFERENCES

Anderson, L. 2023. The History of React.js: A Story of Innovation and Community. Web page. Available at: <https://www.linkedin.com/pulse/history-reactjs-story-innovation-community-l-anderson/> [Accessed 5 March 2024].

Armstrong, T. 2021. What is CORS?. Web page. Available at: <https://www.stackhawk.com/blog/what-is-cors/> [Accessed 20 March 2024].

CHAUHAN, P. 2023. How does Virtual DOM actually work in Reactjs?. Web page. Available at: <https://copperchips.com/how-does-virtual-dom-actually-work-in-reactjs/> [Accessed 9 March 2024].

Corbeel, B. 2024. React Development Diving Deep into JSX: Syntax, Components, and Best Practices. Web page. Available at: <https://www.linkedin.com/pulse/diving-deep-jsx-syntax-components-best-practices-brecht-corbeel-lwace/> [Accessed 10 March 2024].

Daigle, K. 2023. Octoverse: The state of open source and rise of AI in 2023. Web page. Available at: <https://github.blog/2023-11-08-the-state-of-open-source-and-ai/> [Accessed 5 March 2024].

Decpk. 2021. React JSX in Depth. Web page. Available at: <https://www.geeksforgeeks.org/react-jsx-in-depth/> [Accessed 10 March 2024].

DeGroat, T. 2019. The History of JavaScript: Everything You Need to Know. Springboard. Web page. Available at: <https://www.springboard.com/blog/data-science/history-of-javascript/> [Accessed 5 March 2024].

Dotenv. 2024. NPM. Web page. Available at: <https://www.npmjs.com/package/dotenv> [Accessed 20 March 2024].

Evenson, B. 2023. JavaScript. New York: Wilson Publishers.

Express.js Tutorial. 2024. GeeksforGeeks. Web page. Available at:
<https://www.geeksforgeeks.org/express-js/> [Accessed 21 March 2024].

Grigutyte, M. 2023. What is bcrypt and how does it work?. Web page. Available at: <https://nordvpn.com/fr/blog/what-is-bcrypt/#:~:text=Bcrypt%20is%20a%20valuable%20tool,to%20break%20the%20bcrypt%20hash>. [Accessed 1 April 2024].

Introduction to JSON Web Tokens. n.d. JWT.io. Web page. Available at:
<https://jwt.io/introduction> [Accessed 1 April 2024].

Kouchi, B. 2024. How to restart your node.js apps automatically with nodemon. Web page. Available at:
<https://www.digitalocean.com/community/tutorials/workflow-nodemon> [Accessed 1 April 2024].

MongoDB Advantages & Disadvantages. 2023. GeekforGeek. Web page. Available at: <https://www.geeksforgeeks.org/mongodb-advantages-disadvantages/> [Accessed 23 March 2024].

Nodemon reload, automatically. n.d. Nodemon.io. Web page. Available at:
<https://nodemon.io/> [Accessed 1 April 2024].

Nolle, T. 2023. The 5 essential HTTP methods in RESTful API development. Web page. Available at:
<https://www.techtarget.com/searchapparchitecture/tip/The-5-essential-HTTP-methods-in-RESTful-API-development> [Accessed 28 March 2024].

REST API: Key Concepts, Best Practices, and Benefits. 2022. AltexSoft:

Software r&d engineering. Web page. Available at:

<https://www.altexsoft.com/blog/rest-api-design/> [Accessed 28 March 2024].

Sazib, M. 2022. Mongoose Discusses Schema and Model Work. Web page.

Available at: <https://medium.com/@musasazib/mongoose-discusses-schema-and-model-work-81c5b4b73975> [Accessed 24 March 2024].

Sharma, A. 2023. Express JS Tutorial. Web page. Available at:

<https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js>
[Accessed 21 March 2024].

Sufiyan, T. 2023. What is Node.js: A Comprehensive Guide. Web page.

Available at: <https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs>
[Accessed 20 March 2024].

Top Features of MongoDB. n.d. Board Infinity. Web page. Available at:

<https://www.boardinfinity.com/blog/top-features-of-mongodb/>
[Accessed 23 March 2024].

LIST OF FIGURES

Figure 1. Rated of JavaScript language	6
Figure 2. DOM process	7
Figure 3. Using JSX in React	8
Figure 4. How REST API processes between clients and servers	13
Figure 5. Website logo.....	15
Figure 6. Collections on MongoDB Database.....	20
Figure 7. MongoDB connecting by using mongoose	20
Figure 8. Animal Schema	21
Figure 9. Animal Model.....	21
Figure 10. Category Route	23
Figure 11. Using POST method to create a new category	23
Figure 12. Using POST method to create a new animal.....	24
Figure 13. Custom error by createError class.....	24
Figure 14. GET method to load all animals	25
Figure 15. Get a single animal by id	26
Figure 16. Update animal information.....	26
Figure 17. Delete animal	27
Figure 18. Home page.....	28
Figure 19. Register page	29
Figure 20. Hash the password.....	30
Figure 21. Register new users by POST method	30
Figure 22. Log in page.....	31
Figure 23. Compare method in the login process	32
Figure 24. User menu.....	32
Figure 25. Adopt page	33
Figure 26. Filter dog category.....	34
Figure 27. Animal detail page.....	35
Figure 28. Create a new application by POST method.....	36
Figure 29. Adoption application form.....	37
Figure 30. Donation box	38
Figure 31. Make a donation.....	39

Figure 32. Payment successfully	39
Figure 33. Payment cancelled	40
Figure 34. Contact form	41