DOCUMENTATION

By

Hallar Ahmed Khuhro (2312367)

Muhammad Aashir Ali (2312379)

ACKNOWLEDGMENT

We are grateful because we managed to complete our project within the time given by our teachers *[Abid Ali and Usama Khalid]*. This assignment would not have been completed without the effort and co-operation from our group members, *[Hallar Ahmed and Muhammad Aashir].*

We also sincerely thank our teachers *[Abid Ali and Usama Khalid]* for the guidance and encouragement in finishing this project and for teaching us in this course.

Last but not the least, we would like to express our gratitude to our friends and classmates for the support and encouragement.

CONTENT

Introduction

The Library Management System is an easy-to-use website that has real life applications, which made us understand how a database is used in everyday life to make our lives easier and more organised. It is a website developed using JavaScript and HTML programming languages on frontend and PostgreSQL, with neon, on the backend. It is designed to manage and maintain records for libraries. The system is an interactive and user-friendly project which makes data management easier and faster. This system is an essential tool for schools, colleges, and universities to manage library data. It provides a simple and efficient way of managing information, thus helping institutions avoid errors in data management and to ensure data integrity and reliability.

Objectives of the project

The primary objectives of this project are to provide an efficient system for managing and maintaining library records, this includes storing and retrieving data in a fast and efficient manner and to provide a simple and effective method of organising manipulating and retrieving required data and a user-friendly interface that is easy to understand and operate. The website is intuitive and straightforward, allowing users to perform tasks with ease, reducing manual work.

Topic of Our Project

The topic of this project is “Design and Implementation of a Simple Library Management System in JavaScript and PostgreSQL”. This project covers various aspects of backend and frontend database management systems, including data entry and data display. The system has features like adding new records, displaying all records, and a graph presenting useful insights. Each record contains details like ID, name, and whatever is required for the table. This project provides a practical understanding of database management in programming, and it is a valuable resource for libraries for managing data. The project also helped us understand the practical implementation of databases with JavaScript programming concepts and gaining hands-on experience in developing a real-world application.

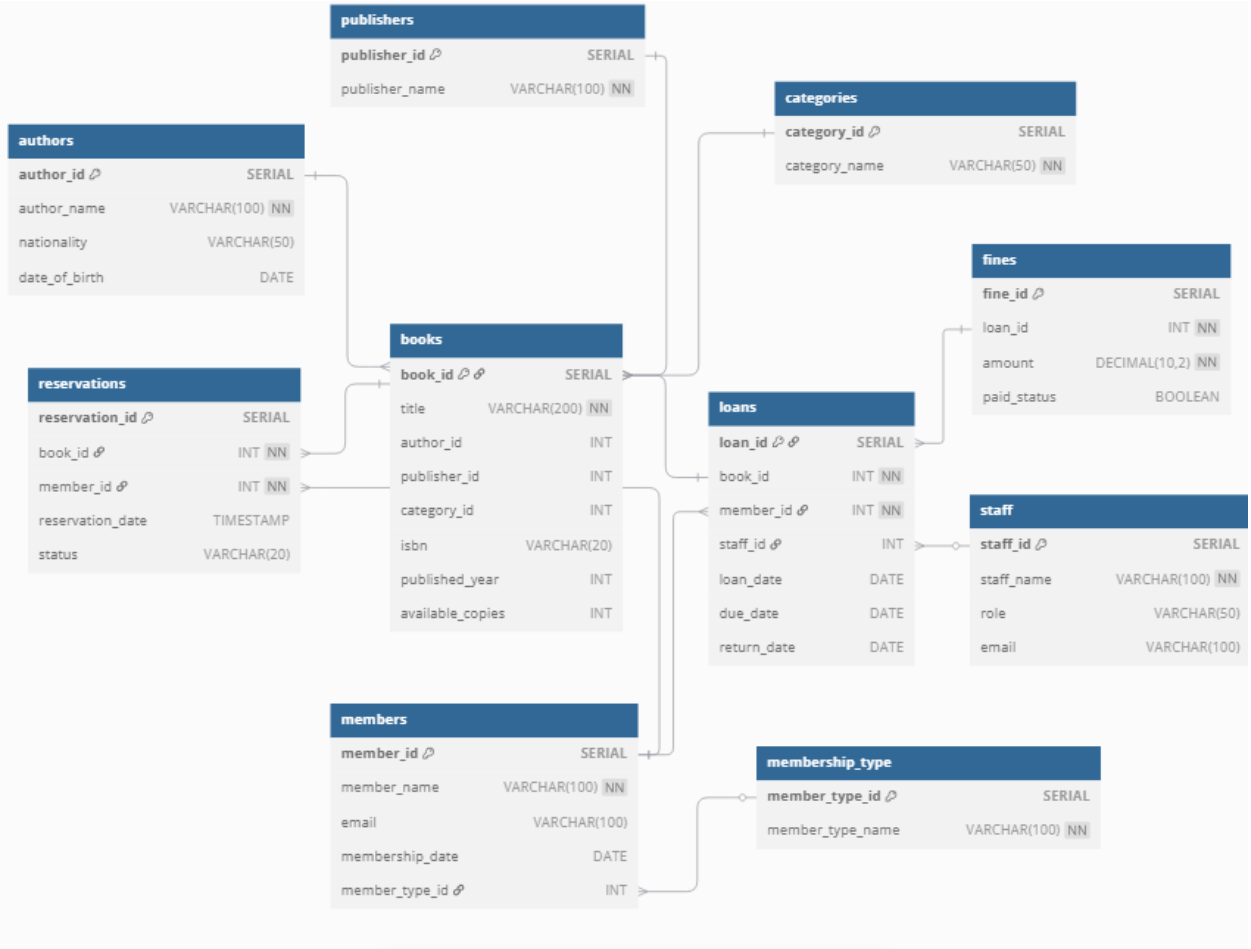
Hardware/Software Requirements

Since we developed the website in JavaScript and HTML, the essential pieces of software are a web browser and an IDE, which might be one of several different ones like Google Chrome, Microsoft Edge, Safari, Firefox, or VS Code, notepad++, IntelliJ IDEA, etc. However, Chrome is the browser and VS Code is the IDE we used. A laptop or PC compatible with the IDE and browser, in this case VS Code and Chrome, and sufficient storage space are the only requirements for the website's basic hardware.

Work Analysis

Task	Aashir	Hallar
Analysis	✓	
Design	✓	
Coding	✓	✓
Testing		✓
Documentation		✓

ERD


publishers
publisher_id SERIAL
publisher_name VARCHAR(100) NN
categories
category_id SERIAL
category_name VARCHAR(50) NN
authors
author_id SERIAL
author_name VARCHAR(100) NN
nationality VARCHAR(50)
date_of_birth DATE
fines
fine_id SERIAL
loan_id INT NN
amount DECIMAL(10,2) NN
paid_status BOOLEAN
books
book_id SERIAL
title VARCHAR(200) NN
author_id INT
publisher_id INT
category_id INT
isbn VARCHAR(20)
published_year INT
available_copies INT
reservations
reservation_id SERIAL
book_id INT NN
member_id INT NN
reservation_date TIMESTAMP
status VARCHAR(20)
loans
loan_id SERIAL
book_id INT NN
member_id INT NN
staff_id INT
loan_date DATE
due_date DATE
return_date DATE
staff
staff_id SERIAL
staff_name VARCHAR(100) NN
role VARCHAR(50)
email VARCHAR(100)
members
member_id SERIAL
member_name VARCHAR(100) NN
email VARCHAR(100)
membership_date DATE
member_type_id INT
membership_type
member_type_id SERIAL
member_type_name VARCHAR(100) NN

Code Snippets

GET and POST:

```
app.get("/books", async (req, res) => {
  try {
    const result = await pool.query("select * from books");
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ Error: err.message });
  }
});
```

```
app.post("/books", async (req, res) => {
  try {
    const {
      book_id,
      title,
      author_id,
      publisher_id,
      category_id,
      isbn,
      published_year,
      available_copies,
    } = req.body;

    const newBook = await pool.query(
      `insert into books (book_id, title, author_id, publisher_id, category_id, isbn, published_year, available_copies)
       values($1, $2, $3, $4, $5, $6, $7, $8) returning *`,
      [
        book_id,
        title,
        author_id,
        publisher_id,
        category_id,
        isbn,
        published_year,
        available_copies,
      ]
    );

    res.status(201).json(newBook.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

The GET function sends the query and gets the books' data from the Postgres table and converts it into JSON format. The POST gets the data from the request into an object and then sends the query with the data and catches an error if it fails. The process is the same for all tables. These are create and read from CRUD.

Loading:

```
function load_books() {
  fetch(BOOKS_API_LINK)
    .then((response) => {
      if (!response.ok) throw new Error("Failed to fetch data");
      return response.json();
    })
    .then((data) => {
      const tbody = document.querySelector("#bookstable tbody");
      tbody.innerHTML = "";

      data.forEach((books) => {
        const row = document.createElement("tr");
        row.innerHTML = `
          <td>${books.book_id}</td>
          <td>${books.title}</td>
          <td>${books.author_id}</td>
          <td>${books.publisher_id}</td>
          <td>${books.category_id}</td>
          <td>${books.isbn}</td>
          <td>${books.published_year}</td>
          <td>${books.available_copies}</td>
        `;
        tbody.appendChild(row);
      });
    })
    .catch((err) => {
      console.log(err.message);
    });
}
```

This code gets all the data from the link and fetches it. Then it checks if response is okay, if yes, then it returns the JSON. The data goes through a foreach loop so each row can be displayed in the table.

Adding:

```
document.getElementById("bookForm").addEventListener("submit", async (e) => {
  e.preventDefault();

  const formData = new FormData(e.target);
  const book = Object.fromEntries(formData.entries());

  try {
    const response = await fetch(BOOKS_API_LINK, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(book),
    });

    if (!response.ok) throw new Error("Failed to add book");

    e.target.reset();
    load_books();
  } catch (err) {
    console.error(err.message);
    alert("Invalid Input!");
  }
});

load_books();
```

This code gets all the data from the input forms using JavaScript's built in functions then fetches the API link and then posts the data after converting it into JSON. The input forms are then reset, and the load_books method is called so the newly added book data is shown immediately.

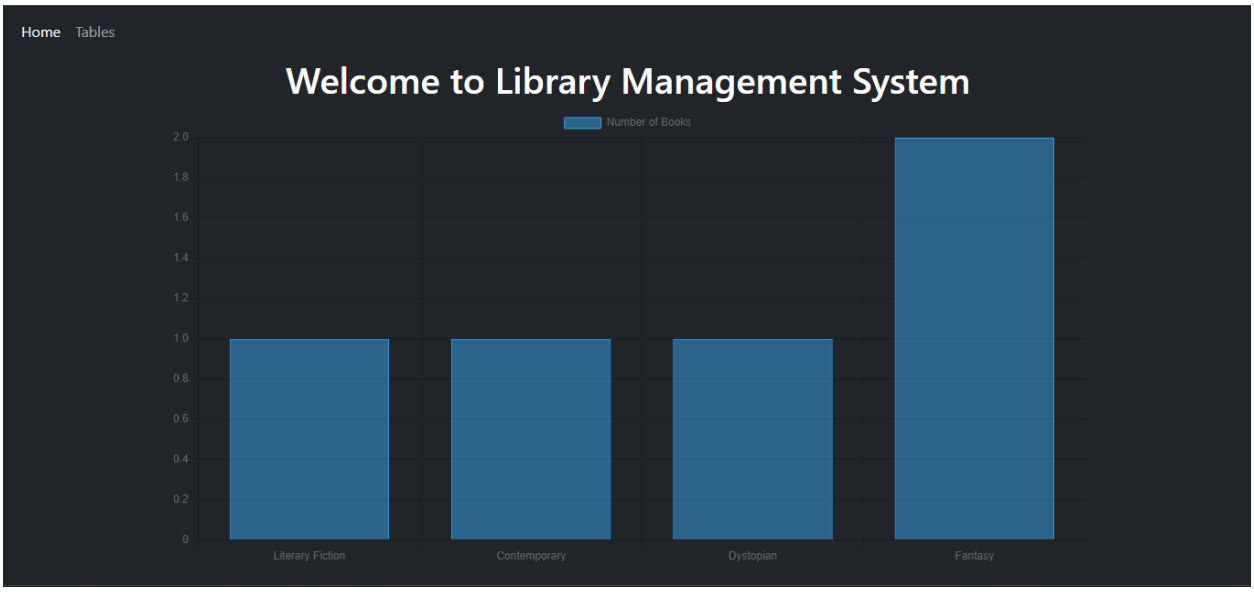
Chart:

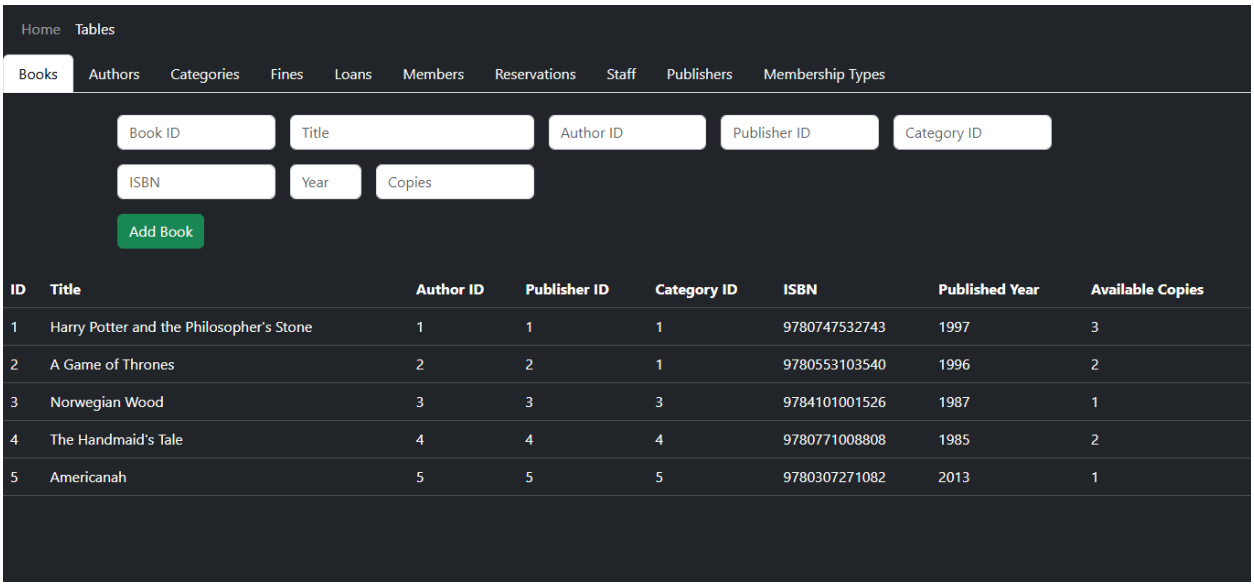
```
function load_chart() {
  fetch(datalink)
    .then((response) => {
      if (!response.ok) throw new Error("Failed to fetch data");
      return response.json();
    })
    .then((data) => {
      new Chart(ctx, {
        type: "bar",
        data: {
          labels: data.map((row) => row.category_name),
          datasets: [
            {
              label: "Number of Books",
              data: data.map((row) => row.count),
              borderWidth: 1,
            },
          ],
        },
        options: {
          scales: {
            y: {
              beginAtZero: true,
            },
          },
        },
      });
    })
    .catch((err) => {
      console.log(err.message);
    });
}

load_chart();
```

This is the code for the chart on the homepage. We used chart.js, a popular charting library, for the bar chart. This code fetches the data from the link and then maps the data in the chart after conversion.

Screenshots





GitHub Link

https://github.com/Hallar57/Library-Management-System