

# Armstrong Number Verification Web Application

December 17, 2024

BHAGYA SURESH KUMAR

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Project Features</b>	<b>3</b>
<b>3</b>	<b>System Design</b>	<b>4</b>
3.1	Database Schema . . . . .	4
3.2	API Endpoints . . . . .	4
3.3	System Overview . . . . .	5
<b>4</b>	<b>Implementation Details</b>	<b>6</b>
4.1	Frontend Implementation . . . . .	6
4.2	Backend Implementation with Database Queries . . . . .	19
4.3	CORS Handling . . . . .	24
<b>5</b>	<b>Challenges and Solutions Implemented</b>	<b>25</b>
5.1	User Identification Across Pages . . . . .	25
5.2	Preventing Unauthorized Access Through URL Manipulation . . . . .	25
5.3	Handling Invalid Inputs . . . . .	25
5.4	Handling Large Tables with Pagination . . . . .	25
<b>6</b>	<b>Testing and Performance</b>	<b>26</b>
6.1	Testing . . . . .	26
6.2	Performance Optimizations . . . . .	26
<b>7</b>	<b>Future Enhancements</b>	<b>26</b>
<b>8</b>	<b>Conclusion</b>	<b>27</b>

# 1 Introduction

The focus of this application is to verify Armstrong numbers, manage user-specific data, and provide meaningful insights through a well-integrated system. Armstrong numbers, also known as narcissistic numbers, hold mathematical significance as they are numbers that are equal to the sum of their own digits each raised to the power of the number of digits. Identifying these numbers programmatically while ensuring seamless data management is a key objective of the system.

The application offers the following core functionalities:

- **Verification of Armstrong numbers:** Users can input a number to check whether it is an Armstrong number. .
- **User-specific data management:** Armstrong numbers verified by users are stored securely in a database. This allows users to view their previously verified numbers.
- **Global data display:** The system provides an overview of all users and their verified Armstrong numbers.

To achieve these objectives, a modern full-stack technology stack has been adopted. The user interface is built using **React**, a popular JavaScript library for developing dynamic and responsive frontend applications. React enables smooth navigation, clean user interactions, and an intuitive design. On the server side, **GoLang** (also referred to as Go) is utilized for its performance, concurrency support, and simplicity. GoLang efficiently handles API requests, processes logic for Armstrong number verification, and interacts with the database layer. For reliable and structured data storage, **MySQL** is employed as the database. MySQL ensures efficient data retrieval, storage, and management, supporting both user-specific and global data queries.

By combining mathematical computations with modern software development practices, the application not only verifies Armstrong numbers but also manages and displays user-specific data in an organized manner. The detailed system design, implementation, and performance evaluation will be discussed in the subsequent sections.

## 2 Project Features

The application incorporates several features to ensure a seamless and efficient user experience. The functionalities provided are as follows:

- **User Registration:**
  - Allows users to create accounts using their email.
  - Basic form validation ensures that user inputs are valid and complete.
- **Armstrong Number Verification:**
  - An input field is provided for users to enter a number.
  - A check button allows users to verify whether the entered number is an Armstrong number.
  - Verified Armstrong numbers are saved to the database for future retrieval.
- **User Dashboard:**
  - Displays a list of Armstrong numbers verified by the logged-in user.
  - Pagination is implemented to handle large datasets efficiently.
- **Global Users Dashboard:**
  - Displays a list of all registered users along with their verified Armstrong numbers.
  - Filtering options, such as showing only positive or negative numbers, are provided.
  - Pagination is implemented to manage large datasets seamlessly.
- **Error Handling:**
  - Input validation is performed on both the frontend and backend to ensure data integrity.
  - User-friendly error messages guide users when invalid inputs or unexpected errors occur.
- **Bonus Features:**
  - Users can input a range (e.g., 10-100), and the page returns the Armstrong numbers within that range.

The above features collectively ensure that the application meets its objectives while providing a smooth and intuitive interface for users.

## 3 System Design

The system design focuses on creating a robust and efficient architecture for verifying Armstrong numbers, storing and retrieving user-specific data, and providing global insights. This section details the database schema, API endpoints, and overall system structure.

### 3.1 Database Schema

The database schema is designed to support both user-specific and global data management efficiently. It consists of two main tables: **Users** and **Numbers**.

- **Users Table:**

- **user\_id** (Primary Key): A unique string identifier for each user.
- **name**: The user's full name.
- **email** (Unique): The user's email, used for registration and login.
- **created\_at**: Timestamp indicating when the user account was created.

- **Numbers Table:**

- **id** (Primary Key): A unique identifier for each record in the numbers table.
- **user\_id** (Foreign Key): References the **user\_id** in the Users table to associate numbers with a specific user.
- **number**: The number entered by the user, regardless of whether it's Armstrong or not.
- **result**: Stores "positive" if the number is Armstrong and "negative" if it is not.
- **created\_at**: Timestamp of when the number was entered and stored.

The database structure ensures that each number entered by a user, along with its result, is linked to that user, allowing for easy retrieval of the data and verification.

### 3.2 API Endpoints

The backend provides a set of API endpoints that allow interaction with the frontend. These endpoints include user management, Armstrong number verification, and data retrieval for both individual users and global users.

- **User Authentication and Management:**

- **POST /users**: Handles user authentication and registration. If the user is new, it enters the user details (including a unique **user\_id**, which is a unique string) into the database.

- **Armstrong Number Verification:**
  - `POST /verify`: Takes the number entered by the user, checks whether the user has already entered that number before. If not, it checks whether the number is Armstrong, and then inputs the number details (positive or negative result), `user_id`, and `created_at` fields into the `Numbers` table.
- **User-Specific Data Retrieval:**
  - `GET /getuserdet`: Receives the `user_id` from the frontend (via `UserContext`) and checks the `Numbers` table to return the total number of searches, total positives, and total negatives for the user.
- **Global Users Data Retrieval:**
  - `GET /getremainingusers`: Returns the total number of searches, `user_id`, total positives and negatives, and the `name` of all users other than the one who is logged in.
- **User Numbers Data Retrieval:**
  - `GET /getusernumbers`: Takes the `user_id` from the frontend (via the URL) and returns all the numbers entered by the user, along with the result ("positive" or "negative") and the `created_at` field, which the frontend displays.

These API endpoints facilitate the core functionalities of the application, including user management, Armstrong number verification, and the retrieval of user-specific or global data.

### 3.3 System Overview

The frontend communicates with the backend through API requests. The user interacts with the frontend application, which collects input data, sends requests to the backend for verification and storage, and retrieves relevant data for display.

- **Frontend (React):** The React application provides an interactive interface, including forms for user registration, number verification, and dashboards for displaying Armstrong numbers.
- **Backend (GoLang):** The Go server handles API requests, performs the Armstrong number verification, and interacts with the MySQL database to store and retrieve data.
- **Database (MySQL):** The MySQL database stores user and number data, supporting efficient queries, filtering, and pagination.

This architecture ensures that the system is scalable, responsive, and able to handle large datasets effectively while providing a seamless user experience.

## 4 Implementation Details

This section outlines the implementation of the system, covering frontend and backend components, CORS handling, and database queries.

### 4.1 Frontend Implementation

The frontend of the system is built using React, providing an interactive interface for user registration, Armstrong number verification, and data display. Below are the key components and sample code placeholders for each:

- **login.js:** Handles user authentication and login.

```
1 import React, { useState, useEffect } from "react";
2 import { useUser } from "../context/UserContext"; // Access
  the user context
3 import { useNavigate } from "react-router-dom"; // Import
  useNavigate
4
5 const LoginForm = () => {
6   const { setUserDetails, user } = useUser(); // Access the
    user context and setUserDetails function
7   const [name, setName] = useState("");
8   const [email, setEmail] = useState("");
9   const navigate = useNavigate(); // Hook for navigation
10
11   useEffect(() => {
12     // Clear the user context when the login page loads
13     console.log("Resetting UserContext");
14     setUserDetails({ user_id: null, name: "", email: "" });
15     localStorage.removeItem("user_id");
16     localStorage.removeItem("name");
17     localStorage.removeItem("email");
18   }, []); // Runs only when the component mounts
19
20   const handleLogin = async (e) => {
21     e.preventDefault();
22
23     console.log("Name entered:", name);
24     console.log("Email entered:", email);
25
26     // Input validation
27     if (!name.trim()) {
28       alert("Name is required");
29       return;
30     }
31     if (!email.trim()) {
32       alert("Email is required");
33       return;
34     }
35
36     try {
```

```

37     const response = await fetch("http://localhost:8080/
38         users", {
39         method: "POST",
40         headers: {
41             "Content-Type": "application/json",
42         },
43         body: JSON.stringify({ name, email }),
44     });
45
46     const data = await response.json();
47     console.log("Backend response:", data);
48
49     if (response.ok) {
50         // Update useContext and localStorage with the new
51         // user_id
52         setUserDetails({
53             user_id: data.user_id, // Use the unique 'user_id'
54             name: data.name,
55             email: data.email,
56         });
57
58         console.log("UserContext updated:", {
59             user_id: data.user_id,
60             name: data.name,
61             email: data.email,
62         });
63
64         // Persist user details in localStorage
65         localStorage.setItem("user_id", data.user_id);
66         localStorage.setItem("name", data.name);
67         localStorage.setItem("email", data.email);
68
69         alert("User successfully logged in!");
70         navigate("/home"); // Navigate to the Armstrong check
71         // page
72     } else {
73         alert(data.error || "Login failed");
74     }
75 } catch (error) {
76     console.error("Error during login:", error);
77     alert("An error occurred. Please try again later.");
78 }
79
80 return (
81     <div className="flex items-center justify-center min-h-
82         screen bg-custom-blue">
83         <div className="md:w-2/6 md:h-2/3 bg-white p-8 rounded-
84             lg shadow-lg">
85             <form onSubmit={handleLogin}>
86                 <h2 className="text-4xl font-bold mb-4 text-center

```



```

      text-custom-blue">Welcome</h2>
83 <div className="mb-4">
84   <label htmlFor="name" className="block text-xl
      font-bold text-custom-blue">
85     Name
86   </label>
87   <input
88     type="text"
89     id="name"
90     name="name"
91     value={name}
92     onChange={(e) => setName(e.target.value)} //
      Updates 'name' state
93     className="mt-1 block w-full px-4 py-2 bg-gray
      -200 text-gray-900 rounded-md focus:outline-
      none focus:ring-2 focus:ring-custom-blue
      focus:border-transparent"
94   />
95 </div>
96 <div className="mb-4">
97   <label htmlFor="email" className="block text-xl
      font-bold text-custom-blue">
98     Email
99   </label>
100  <input
101    type="email"
102    id="email"
103    name="email"
104    value={email}
105    onChange={(e) => setEmail(e.target.value)} //
      Updates 'email' state
106    className="mt-1 block w-full px-4 py-2 bg-gray
      -200 text-gray-900 rounded-md focus:outline-
      none focus:ring-2 focus:ring-custom-blue
      focus:border-transparent"
107  />
108 </div>
109 <div className="flex justify-center">
110   <button
111     type="submit"
112     className="bg-custom-blue text-xl text-white
      font-bold py-2 px-4 rounded shadow hover:bg-
      opacity-90 focus:outline-none focus:ring
      focus:ring-blue-500"
113   >
114     LOGIN
115   </button>
116 </div>
117 </form>
118 </div>
119 </div>

```

```

120 |   );
121 | };
122 |
123 | export default LoginForm;

```

Listing 1: login.js

**checkArm.js:** A page for verifying if a number is Armstrong.

```

1 import React, { useState } from "react";
2 import { useUser } from "../context/UserContext"; // Access user
  data from the context
3 import thumbsUp from "../static/thumbsUp.gif";
4 import thumbsDown from "../static/thumbsDown.gif";
5 import NavBar from "../navbar"; // Import NavBar component
6
7 const CheckNumber = () => {
8   const { user } = useUser(); // Access user data from the
  context
9   const [number, setNumber] = useState("");
10  const [result, setResult] = useState("");
11
12  const handleCheck = async () => {
13    if (!/^\d+$/.test(number)) {
14      alert("Please enter a valid numeric value");
15      return;
16    }
17
18    const num = parseInt(number, 10);
19
20    // Check if the number is Armstrong
21    const isArmstrongNumber = isArmstrong(num);
22    const resultText = isArmstrongNumber ? "positive" : "negative
  ";
23
24    try {
25      // Send the number, user id, and result to the backend
26      const response = await fetch("http://localhost:8080/verify
  ", {
27        method: "POST",
28        headers: {
29          "Content-Type": "application/json",
30        },
31        body: JSON.stringify({
32          user_id: user.user_id, // Pass the UID from context
33          number: num,
34          result: resultText, // Send the result as positive or
  negative
35        }),
36      });
37
38      if (response.ok) {

```

```

39     const data = await response.json();
40     console.log("Backend Response:", data);
41     // Set result based on the backend response
42     setResult({
43         message: isArmstrongNumber
44             ? 'Congrats! Number ${num} is an Armstrong number.'
45             : 'Number ${num} is not an Armstrong number.',
46         gif: isArmstrongNumber ? thumbsUp : thumbsDown,
47     });
48 } else {
49     const error = await response.json();
50     alert(error.error || "An error occurred");
51     setResult({
52         message: 'An error occurred while checking the number
53             .',
54         gif: thumbsDown,
55     });
56 } catch (error) {
57     console.error("Error:", error);
58     alert("An error occurred. Please try again.");
59 }
60 };
61
62 const isArmstrong = (num) => {
63     const digits = num.toString().split("").map(Number);
64     const sum = digits.reduce((acc, digit) => acc + Math.pow(
65         digit, digits.length), 0);
66     return sum === num;
67 };
68
69 return (
70     <div className="bg-gray-100 min-h-screen flex flex-col"> {/*
71         Ensured full height with min-h-screen */}
72     <NavBar /> {/* Fixed NavBar at the top */}
73
74     <div className="flex flex-col items-center justify-start
75         flex-1 pt-28"> {/* pt-28 to ensure spacing for NavBar
76         */}
77     <div className="bg-custom-blue text-white p-8 px-0
78         rounded-lg shadow-md w-[30%] pb-15 h-[40%] flex flex-
79         col opacity-90 items-center justify-center">
80         {/* Increased height and added flex-grow to parent
81         container */}
82         <div className="flex flex-col justify-between h-full w-
83             full p-8 px-0">
84             <div className="flex flex-col items-center">
85                 <h1 className="text-2xl font-bold mb-4 text-center
86                     pb-2">Let's Check a number</h1>
87                 <label htmlFor="number" className="block text-2xl
88                     mb-2 text-center pb-5">

```

```

79         Enter a number
80     </label>
81     <input
82         id="number"
83         type="text"
84         value={number}
85         onChange={(e) => setNumber(e.target.value)}
86         className="w-[70%] p-2 rounded bg-white text-
            custom-blue text-center text-xl border focus:
            outline-none focus:ring-2 focus:ring-blue-500"
87     />
88 </div>
89 <button
90     onClick={handleCheck}
91     className="mt-4 bg-white text-xl font-bold text-
        custom-blue px-4 py-2 rounded hover:text-bold
        transition self-center"
92 >
93     Check
94 </button>
95 </div>
96 </div>
97
98 {result && (
99     <div className="flex items-center pt-5 w-[20%] justify-
        center space-x-0">
100         <img src={result.gif} alt="Result" className="w-[120
            px] h-[100px]" />
101         <span className="text-xl text-center font-bold text-
            custom-blue">{result.message}</span>
102     </div>
103 )}
104 </div>
105 </div>
106 );
107 };
108
109 export default CheckNumber;

```

Listing 2: checkArm.js

**userDetails.js:** Displays user-specific details like searches, positives, and negatives.

```

1 import React, { useEffect, useState } from "react";
2 import { useNavigate } from "react-router-dom"; // Import
    useNavigate from react-router-dom
3 import UserBox from "../userBox";
4 import { useUser } from "../context/UserContext";
5 import NavBar from "../navbar";
6
7 const UserDetails = () => {
8     const [userDetails, setUserDetails] = useState(null);

```

```

9   const [otherUsers, setOtherUsers] = useState([]); // Initialize
    as an empty array
10  const { user } = useUser();
11  const navigate = useNavigate(); // Initialize the navigate
    function
12
13  useEffect(() => {
14    if (user && user.user_id) {
15      console.log('Attempting to fetch user details for ID: ${
        user.user_id}');
16
17      fetch('http://localhost:8080/getuserdet?user_id=${
        encodeURIComponent(user.user_id)}', {
18        method: "GET",
19        headers: {
20          Accept: "application/json",
21        },
22      })
23        .then((response) => response.json())
24        .then((data) => {
25          console.log("User Details fetched:", data);
26          setUserDetails(data);
27        })
28        .catch((error) => console.error("Error fetching user
        details:", error));
29
30      // Fetch remaining users
31      fetch('http://localhost:8080/getremainingusers?user_id=${
        encodeURIComponent(user.user_id)}', {
32        method: "GET",
33        headers: {
34          Accept: "application/json",
35        },
36      })
37        .then((response) => response.json())
38        .then((data) => {
39          console.log("Other Users fetched:", data);
40          setOtherUsers(data); // Store the list of other users
41        })
42        .catch((error) => console.error("Error fetching other
        users:", error));
43    } else {
44      console.warn("User ID is not available. Cannot fetch
        details.", user);
45    }
46  }, [user]);
47
48  // Ensure that 'otherUsers' is an array before attempting to
    use '.map()'
49  if (!Array.isArray(otherUsers)) {
50    console.error("otherUsers is not an array", otherUsers);

```

```

51     return <div>Loading...</div>;
52 }
53
54 const handleUserClick = (userID) => {
55     // Navigate to the dynamic user details page with the user ID
56     navigate(`/user-details/${userID}`);
57 };
58
59 if (!userDetails) {
60     return <div>Loading...</div>;
61 }
62
63 return (
64     <div className=" min-h-screen text-white flex flex-col">
65         {/* Navbar at the top */}
66         <NavBar />
67
68         {/* Main content container */}
69         <div className="flex flex-col bg-custom-blue items-center
70             justify-start flex-grow pt-20 px-10">
71             <h1 className="text-4xl font-semi-bold mb-8">Hey {user.
72                 name}, Let's meet our users</h1>
73
74             {/* Current user details */}
75             <div className="flex items-center w-full pt-10 pb-8
76                 border-b-[0.1px] border-white/40">
77                 <span className="mr-8 pr-10 font-bold text-xl">You</
78                     span>
79                 <UserBox
80                     name={userDetails.name || user.name}
81                     searches={userDetails.searches}
82                     positives={userDetails.positives}
83                     negatives={userDetails.negatives}
84                     onClick={() => handleUserClick(user.user_id)} // Pass
85                     the user ID to handle the click
86                 />
87             </div>
88
89             {/* Other users */}
90             <div className="flex flex-wrap justify-start w-full gap-6
91                 pt-10">
92                 {otherUsers.length > 0 ? (
93                     otherUsers.map((otherUser) => (
94                         <UserBox
95                             key={otherUser.user_id}
96                             name={otherUser.name}
97                             searches={otherUser.searches}
98                             positives={otherUser.positives}
99                             negatives={otherUser.negatives}
100                             onClick={() => handleUserClick(otherUser.user_id)}
101                         >

```

```

95         />
96     ))
97     ) : (
98         <p>No other users found.</p>
99     )}
100     </div>
101 </div>
102 </div>
103 );
104 };
105
106 export default UserDetails;
```

### Listing 3: userDetails.js

**userPage.js:** Displays a table which has details about the numbers entered by the user like result, date, and time.

```

1 import React, { useEffect, useState, useCallback } from "react";
2 import { useParams } from "react-router-dom";
3
4 const UserPage = () => {
5   const { user_id } = useParams();
6   const [userDetails, setUserDetails] = useState(null);
7   const [numberDetails, setNumberDetails] = useState([]);
8
9   const [currentPage, setCurrentPage] = useState(1);
10  const [totalPages, setTotalPages] = useState(0);
11  const [totalCount, setTotalCount] = useState(0);
12  const [pageSize, setPageSize] = useState(5); // Default to 5
13    rows per page
14
15  const [filter, setFilter] = useState("all"); // New filter
16    state
17
18  // Fetch user and number details
19  const fetchUserDetails = useCallback(() => {
20    if (user_id) {
21      console.log(`Fetching details for user ID: ${user_id}`);
22
23      // Fetch user details
24      fetch(
25        `http://localhost:8080/getuserdet?user_id=${
26          encodeURIComponent(user_id)
27        }`,
28        {
29          method: "GET",
30          headers: { Accept: "application/json" },
31        }
32      )
33        .then((response) => response.json())
34        .then((data) => {
35          console.log("User Details fetched:", data);
36          setUserDetails(data);
37        });
38    }
39  });
40
41  // Fetch number details
42  const fetchNumberDetails = useCallback(() => {
43    if (user_id) {
44      console.log(`Fetching details for user ID: ${user_id}`);
45
46      // Fetch number details
47      fetch(
48        `http://localhost:8080/getnumdet?user_id=${
49          encodeURIComponent(user_id)
50        }`,
51        {
52          method: "GET",
53          headers: { Accept: "application/json" },
54        }
55      )
56        .then((response) => response.json())
57        .then((data) => {
58          console.log("Number Details fetched:", data);
59          setNumberDetails(data);
60        });
61    }
62  });
63
64  // Fetch total count
65  const fetchTotalCount = useCallback(() => {
66    if (user_id) {
67      console.log(`Fetching details for user ID: ${user_id}`);
68
69      // Fetch total count
70      fetch(
71        `http://localhost:8080/gettotalcount?user_id=${
72          encodeURIComponent(user_id)
73        }`,
74        {
75          method: "GET",
76          headers: { Accept: "application/json" },
77        }
78      )
79        .then((response) => response.json())
80        .then((data) => {
81          console.log("Total Count fetched:", data);
82          setTotalCount(data);
83        });
84    }
85  });
86
87  // Fetch total pages
88  const fetchTotalPages = useCallback(() => {
89    if (user_id) {
90      console.log(`Fetching details for user ID: ${user_id}`);
91
92      // Fetch total pages
93      fetch(
94        `http://localhost:8080/gettotalpages?user_id=${
95          encodeURIComponent(user_id)
96        }`,
97        {
98          method: "GET",
99          headers: { Accept: "application/json" },
100        }
101      )
102        .then((response) => response.json())
103        .then((data) => {
104          console.log("Total Pages fetched:", data);
105          setTotalPages(data);
106        });
107    }
108  });
109
110  // Fetch current page
111  const fetchCurrentPage = useCallback(() => {
112    if (user_id) {
113      console.log(`Fetching details for user ID: ${user_id}`);
114
115      // Fetch current page
116      fetch(
117        `http://localhost:8080/getcurrentpage?user_id=${
118          encodeURIComponent(user_id)
119        }`,
120        {
121          method: "GET",
122          headers: { Accept: "application/json" },
123        }
124      )
125        .then((response) => response.json())
126        .then((data) => {
127          console.log("Current Page fetched:", data);
128          setCurrentPage(data);
129        });
130    }
131  });
132
133  // Fetch filter
134  const fetchFilter = useCallback(() => {
135    if (user_id) {
136      console.log(`Fetching details for user ID: ${user_id}`);
137
138      // Fetch filter
139      fetch(
140        `http://localhost:8080/getfilter?user_id=${
141          encodeURIComponent(user_id)
142        }`,
143        {
144          method: "GET",
145          headers: { Accept: "application/json" },
146        }
147      )
148        .then((response) => response.json())
149        .then((data) => {
150          console.log("Filter fetched:", data);
151          setFilter(data);
152        });
153    }
154  });
155
156  // Fetch rows per page
157  const fetchRowsPerPage = useCallback(() => {
158    if (user_id) {
159      console.log(`Fetching details for user ID: ${user_id}`);
160
161      // Fetch rows per page
162      fetch(
163        `http://localhost:8080/getrowsperpage?user_id=${
164          encodeURIComponent(user_id)
165        }`,
166        {
167          method: "GET",
168          headers: { Accept: "application/json" },
169        }
170      )
171        .then((response) => response.json())
172        .then((data) => {
173          console.log("Rows per page fetched:", data);
174          setPageSize(data);
175        });
176    }
177  });
178
179  // Fetch user details
180  const fetchUserDetails2 = useCallback(() => {
181    if (user_id) {
182      console.log(`Fetching details for user ID: ${user_id}`);
183
184      // Fetch user details
185      fetch(
186        `http://localhost:8080/getuserdet?user_id=${
187          encodeURIComponent(user_id)
188        }`,
189        {
190          method: "GET",
191          headers: { Accept: "application/json" },
192        }
193      )
194        .then((response) => response.json())
195        .then((data) => {
196          console.log("User Details fetched:", data);
197          setUserDetails(data);
198        });
199    }
200  });
201
202  // Fetch number details
203  const fetchNumberDetails2 = useCallback(() => {
204    if (user_id) {
205      console.log(`Fetching details for user ID: ${user_id}`);
206
207      // Fetch number details
208      fetch(
209        `http://localhost:8080/getnumdet?user_id=${
210          encodeURIComponent(user_id)
211        }`,
212        {
213          method: "GET",
214          headers: { Accept: "application/json" },
215        }
216      )
217        .then((response) => response.json())
218        .then((data) => {
219          console.log("Number Details fetched:", data);
220          setNumberDetails(data);
221        });
222    }
223  });
224
225  // Fetch total count
226  const fetchTotalCount2 = useCallback(() => {
227    if (user_id) {
228      console.log(`Fetching details for user ID: ${user_id}`);
229
230      // Fetch total count
231      fetch(
232        `http://localhost:8080/gettotalcount?user_id=${
233          encodeURIComponent(user_id)
234        }`,
235        {
236          method: "GET",
237          headers: { Accept: "application/json" },
238        }
239      )
240        .then((response) => response.json())
241        .then((data) => {
242          console.log("Total Count fetched:", data);
243          setTotalCount(data);
244        });
245    }
246  });
247
248  // Fetch total pages
249  const fetchTotalPages2 = useCallback(() => {
250    if (user_id) {
251      console.log(`Fetching details for user ID: ${user_id}`);
252
253      // Fetch total pages
254      fetch(
255        `http://localhost:8080/gettotalpages?user_id=${
256          encodeURIComponent(user_id)
257        }`,
258        {
259          method: "GET",
260          headers: { Accept: "application/json" },
261        }
262      )
263        .then((response) => response.json())
264        .then((data) => {
265          console.log("Total Pages fetched:", data);
266          setTotalPages(data);
267        });
268    }
269  });
270
271  // Fetch current page
272  const fetchCurrentPage2 = useCallback(() => {
273    if (user_id) {
274      console.log(`Fetching details for user ID: ${user_id}`);
275
276      // Fetch current page
277      fetch(
278        `http://localhost:8080/getcurrentpage?user_id=${
279          encodeURIComponent(user_id)
280        }`,
281        {
282          method: "GET",
283          headers: { Accept: "application/json" },
284        }
285      )
286        .then((response) => response.json())
287        .then((data) => {
288          console.log("Current Page fetched:", data);
289          setCurrentPage(data);
290        });
291    }
292  });
293
294  // Fetch filter
295  const fetchFilter2 = useCallback(() => {
296    if (user_id) {
297      console.log(`Fetching details for user ID: ${user_id}`);
298
299      // Fetch filter
300      fetch(
301        `http://localhost:8080/getfilter?user_id=${
302          encodeURIComponent(user_id)
303        }`,
304        {
305          method: "GET",
306          headers: { Accept: "application/json" },
307        }
308      )
309        .then((response) => response.json())
310        .then((data) => {
311          console.log("Filter fetched:", data);
312          setFilter(data);
313        });
314    }
315  });
316
317  // Fetch rows per page
318  const fetchRowsPerPage2 = useCallback(() => {
319    if (user_id) {
320      console.log(`Fetching details for user ID: ${user_id}`);
321
322      // Fetch rows per page
323      fetch(
324        `http://localhost:8080/getrowsperpage?user_id=${
325          encodeURIComponent(user_id)
326        }`,
327        {
328          method: "GET",
329          headers: { Accept: "application/json" },
330        }
331      )
332        .then((response) => response.json())
333        .then((data) => {
334          console.log("Rows per page fetched:", data);
335          setPageSize(data);
336        });
337    }
338  });
339
340  // Fetch user details
341  const fetchUserDetails3 = useCallback(() => {
342    if (user_id) {
343      console.log(`Fetching details for user ID: ${user_id}`);
344
345      // Fetch user details
346      fetch(
347        `http://localhost:8080/getuserdet?user_id=${
348          encodeURIComponent(user_id)
349        }`,
350        {
351          method: "GET",
352          headers: { Accept: "application/json" },
353        }
354      )
355        .then((response) => response.json())
356        .then((data) => {
357          console.log("User Details fetched:", data);
358          setUserDetails(data);
359        });
360    }
361  });
362
363  // Fetch number details
364  const fetchNumberDetails3 = useCallback(() => {
365    if (user_id) {
366      console.log(`Fetching details for user ID: ${user_id}`);
367
368      // Fetch number details
369      fetch(
370        `http://localhost:8080/getnumdet?user_id=${
371          encodeURIComponent(user_id)
372        }`,
373        {
374          method: "GET",
375          headers: { Accept: "application/json" },
376        }
377      )
378        .then((response) => response.json())
379        .then((data) => {
380          console.log("Number Details fetched:", data);
381          setNumberDetails(data);
382        });
383    }
384  });
385
386  // Fetch total count
387  const fetchTotalCount3 = useCallback(() => {
388    if (user_id) {
389      console.log(`Fetching details for user ID: ${user_id}`);
390
391      // Fetch total count
392      fetch(
393        `http://localhost:8080/gettotalcount?user_id=${
394          encodeURIComponent(user_id)
395        }`,
396        {
397          method: "GET",
398          headers: { Accept: "application/json" },
399        }
400      )
401        .then((response) => response.json())
402        .then((data) => {
403          console.log("Total Count fetched:", data);
404          setTotalCount(data);
405        });
406    }
407  });
408
409  // Fetch total pages
410  const fetchTotalPages3 = useCallback(() => {
411    if (user_id) {
412      console.log(`Fetching details for user ID: ${user_id}`
```

```

32     })
33     .catch((error) => console.error("Error fetching user
34         details:", error));
35
36     // Fetch paginated and filtered number details
37     fetch(
38         'http://localhost:8080/getusernumbers?user_id=${
39             encodeURIComponent(
40                 user_id
41             )}&page=${currentPage}&page_size=${pageSize}&filter=${
42                 filter}', // Include filter in query
43         {
44             method: "GET",
45             headers: { Accept: "application/json" },
46         })
47         .then((response) => response.json())
48         .then((data) => {
49             console.log("Number Details fetched:", data);
50             setNumberDetails(data.number_details);
51             setTotalPages(data.total_pages);
52             setTotalCount(data.total_count);
53         })
54         .catch((error) =>
55             console.error("Error fetching user number details:",
56                 error)
57         );
58     }, [user_id, currentPage, pageSize, filter]);
59
60     useEffect(() => {
61         fetchUserDetails();
62     }, [fetchUserDetails]);
63
64     // Pagination handlers
65     const handleNextPage = () => {
66         if (currentPage < totalPages) {
67             setCurrentPage(currentPage + 1);
68         }
69     };
70
71     const handlePrevPage = () => {
72         if (currentPage > 1) {
73             setCurrentPage(currentPage - 1);
74         }
75     };
76
77     // Page size change handler
78     const handlePageSizeChange = (e) => {

```



```

79     setPageSize(parseInt(e.target.value, 10));
80     setCurrentPage(1); // Reset to the first page when the page
      size changes
81 };
82
83 // Filter change handler
84 const handleFilterChange = (e) => {
85     setFilter(e.target.value);
86     setCurrentPage(1); // Reset to the first page when the filter
      changes
87 };
88
89 if (!userDetails) {
90     return <div className="text-center text-lg mt-10">Loading
      user details...</div>;
91 }
92
93 return (
94     <div className="p-4 max-w-5xl mx-auto">
95         <div className="flex items-center justify-between mb-20 mt
      -20">
96             <button
97                 onClick={() => window.history.back()}
98                 className="text-custom-blue font-bold flex items-center
      "
99             >
100                 <span className="mr-2 font-bold">&larr;</span> Back
101             </button>
102
103             <h1 className="text-3xl font-bold text-custom-blue mx-
      auto">
104                 Searches by {userDetails.name}
105             </h1>
106
107             {/* Filter Dropdown */}
108             <select
109                 value={filter}
110                 onChange={handleFilterChange}
111                 className="border text-custom-blue font-bold rounded px
      -2 py-1"
112             >
113                 <option value="all">All</option>
114                 <option value="positive">Positives</option>
115                 <option value="negative">Negatives</option>
116             </select>
117         </div>
118
119         <div className="overflow-x-auto">
120             <table className="table-auto border-separate border w-
      full">
121                 <thead>

```

```

122         <tr className="bg-custom-blue text-white">
123             <th className="px-4 py-2 text-left text-xl rounded-
                tl-lg">NUMBER</th>
124             <th className="px-4 py-2 text-xl text-left">RESULT
                </th>
125             <th className="px-4 py-2 text-xl text-left">DATE</
                th>
126             <th className="px-4 py-2 text-left text-xl rounded-
                tr-lg">TIME</th>
127         </tr>
128     </thead>
129     <tbody>
130         {numberDetails.map((entry) => (
131             <tr key={entry.id} className="bg-custom-blue">
132                 <td className="border px-4 py-2 text-white">{
                    entry.number}</td>
133                 <td className="border px-4 py-2 text-white">{
                    entry.result}</td>
134                 <td className="border px-4 py-2 text-white">
135                     {new Date(entry.created_at).toLocaleDateString
                        ()}<
                    /td>
136                 <td className="border px-4 py-2 text-white">
137                     {new Date(entry.created_at).toLocaleTimeString
                        ()}<
                    /td>
138                 </td>
139             </tr>
140         ))}
141     </tbody>
142 </table>
143
144
145     {/* Pagination Controls */}
146     <div className="flex justify-between items-center mt-4">
147         <div className="flex items-center">
148             <label htmlFor="pageSize" className="mr-2 font-bold
                text-custom-blue">Show:</label>
149             <select
150                 id="pageSize"
151                 value={pageSize}
152                 onChange={handlePageSizeChange}
153                 className="border text-custom-blue font-bold
                    rounded px-2 py-1"
154             >
155                 <option value={5}>5</option>
156                 <option value={10}>10</option>
157                 <option value={25}>25</option>
158                 <option value={50}>50</option>
159             </select>
160         </div>
161
162         <div className="text-center font-bold text-custom-blue

```

```

163         ">
164         Page {currentPage} of {totalPages} (Total {totalCount
165         } entries)
166     </div>
167     <div className="flex space-x-2">
168         <button
169             onClick={handlePrevPage}
170             disabled={currentPage === 1}
171             className={`px-4 py-2 border rounded ${
172                 currentPage === 1
173                     ? 'bg-gray-200 font-bold text-custom-blue
174                       cursor-not-allowed'
175                     : 'bg-custom-blue text-white font-bold hover:
176                       opacity-90'
177             }`}
178         >
179             Previous
180         </button>
181         <button
182             onClick={handleNextPage}
183             disabled={currentPage === totalPages}
184             className={`px-4 py-2 border rounded ${
185                 currentPage === totalPages
186                     ? 'bg-gray-200 font-bold text-custom-blue
187                       cursor-not-allowed'
188                     : 'bg-custom-blue text-white font-bold hover:
189                       opacity-90'
190             }`}
191         >
192             Next
193         </button>
194     </div>
195 </div>
196 </div>
197 );
198 };
199
200 export default UserPage;

```

Listing 4: userPage.js

## 4.2 Backend Implementation with Database Queries

The backend is built using Go, which handles API requests for user authentication, Armstrong number verification, and data retrieval. Below are the handler functions corresponding to each route.

- **POST /users:** Handles user authentication and registration.

```
1 package handlers
2
3 import (
4     "backend/config"
5     "backend/models"
6     "encoding/json"
7     "net/http"
8     "regexp"
9     "strings"
10    "time"
11
12    "github.com/google/uuid"
13 )
14
15 func LoginUser(w http.ResponseWriter, r *http.Request) {
16     var input struct {
17         Name  string `json:"name"`
18         Email string `json:"email"`
19     }
20     if err := json.NewDecoder(r.Body).Decode(&input); err !=
21         nil {
22         http.Error(w, "Invalid request payload", http.
23             StatusBadRequest)
24         return
25     }
26
27     // Validate inputs
28     if strings.TrimSpace(input.Name) == "" {
29         http.Error(w, "Name is required", http.
30             StatusBadRequest)
31         return
32     }
33     if strings.TrimSpace(input.Email) == "" {
34         http.Error(w, "Email is required", http.
35             StatusBadRequest)
36         return
37     }
38
39     // Validate email format using regex
40     emailRegex := `^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$`
41     matched, err := regexp.MatchString(emailRegex, input.Email)
42     if err != nil || !matched {
43         w.Header().Set("Content-Type", "application/json")
44         w.WriteHeader(http.StatusBadRequest)
45         json.NewEncoder(w).Encode(map[string]string{"error"
46             : "Invalid email format"})
47         return
48     }
49 }
```

```

45 // Check if the user already exists
46 var existingUser models.User
47 if err := config.DB.Where("email = ?", input.Email).First(&
    existingUser).Error; err == nil {
48     // Return the existing user's details
49     w.Header().Set("Content-Type", "application/json")
50     w.WriteHeader(http.StatusOK)
51     json.NewEncoder(w).Encode(map[string]interface{}{
52         "message": "User already exists",
53         "user_id": existingUser.UserID, // Send
            unique user_id
54         "name":     existingUser.Name,
55         "email":     existingUser.Email,
56     })
57     return
58 }
59
60 // Generate a unique UserID
61 newUserID := uuid.New().String()
62
63 // Add new user to the database
64 newUser := models.User{
65     UserID:     newUserID,
66     Name:       input.Name,
67     Email:      input.Email,
68     CreatedAt:  time.Now(),
69 }
70
71 if err := config.DB.Create(&newUser).Error; err != nil {
72     http.Error(w, "Error creating user", http.
        StatusInternalServerError)
73     return
74 }
75
76 // Return the new user's details
77 w.Header().Set("Content-Type", "application/json")
78 w.WriteHeader(http.StatusCreated)
79 json.NewEncoder(w).Encode(map[string]interface{}{
80     "message": "User created successfully",
81     "user_id": newUser.UserID, // Send unique user_id
82     "name":     newUser.Name,
83     "email":     newUser.Email,
84 })
85 }

```

Listing 5: GoLang Login User Handler

- **POST /verify:** Verifies if a number is Armstrong and stores the result.

```

1 package handlers
2
3 import (
4     "backend/config"
5     "backend/models"
6     "encoding/json"
7     "net/http"
8     "time"
9 )

```

```

10
11 // VerifyArmstrong checks if a number is Armstrong and adds it to
    the database
12 func VerifyArmstrong(w http.ResponseWriter, r *http.Request) {
13     // Decode JSON request body
14     var data struct {
15         UserID string 'json:"user_id"' // Use string to match UUID
            format
16         Number int 'json:"number"'
17         Result string 'json:"result"' // Result will be either "
            positive" or "negative"
18     }
19
20     if err := json.NewDecoder(r.Body).Decode(&data); err != nil {
21         http.Error(w, "Invalid request payload", http.
            StatusBadRequest)
22         return
23     }
24
25     // Validate inputs
26     if data.UserID == "" {
27         http.Error(w, "User ID is required", http.StatusBadRequest)
28         return
29     }
30     if data.Number <= 0 {
31         http.Error(w, "Invalid number", http.StatusBadRequest)
32         return
33     }
34
35     // Check if the number already exists for the user
36     var existingEntry models.ArmstrongNumber
37     if err := config.DB.Where("user_id = ? AND number = ?", data.
        UserID, data.Number).First(&existingEntry).Error; err == nil
        {
38         // If the entry exists, return a conflict response
39         w.WriteHeader(http.StatusConflict)
40         json.NewEncoder(w).Encode(map[string]string{
41             "error": "Number already checked by this user",
42         })
43         return
44     }
45
46     // Add new entry to the database
47     newEntry := models.ArmstrongNumber{
48         UserID:    data.UserID, // Store the user_id as string
49         Number:    data.Number,
50         Result:    data.Result, // Store the result (positive or
            negative)
51         CreatedAt: time.Now(),
52     }
53
54     if err := config.DB.Create(&newEntry).Error; err != nil {
55         http.Error(w, "Error saving entry", http.
            StatusInternalServerError)
56         return
57     }
58
59     // Send a successful response with the new entry

```

```

60     w.Header().Set("Content-Type", "application/json")
61     w.WriteHeader(http.StatusCreated)
62     json.NewEncoder(w).Encode(newEntry)
63 }

```

Listing 6: GoLang Armstrong Verification Handler

- **GET /getusernumbers:** Returns the details of the numbers entered by the users like result and timestamp of when the number was entered.

```

1  package handlers
2
3  import (
4      "backend/config"
5      "encoding/json"
6      "log"
7      "net/http"
8      "strconv"
9  )
10
11 // NumberDetails represents a single entry in the armstrong_numbers
12 // table
13 type NumberDetails struct {
14     ID          uint   `json:"id"`
15     Number      int    `json:"number"`
16     Result      string `json:"result"`
17     CreatedAt   string `json:"created_at"`
18 }
19
20 // PaginatedNumberDetailsResponse contains paginated number details
21 type PaginatedNumberDetailsResponse struct {
22     NumberDetails []NumberDetails `json:"number_details"`
23     TotalCount    int64            `json:"total_count"`
24     Page          int              `json:"page"`
25     PageSize      int              `json:"page_size"`
26     TotalPages    int              `json:"total_pages"`
27 }
28
29 // GetUserNumbers retrieves paginated numbers entered by the user
30 func GetUserNumbers(w http.ResponseWriter, r *http.Request) {
31     log.Println("Received request to fetch user numbers")
32
33     // Get user ID from query parameter
34     userID := r.URL.Query().Get("user_id")
35     log.Printf("Received user_id: %s", userID)
36
37     if userID == "" {
38         log.Println("Missing user ID")
39         http.Error(w, `{"error": "Missing user ID"}`, http.
40             StatusBadRequest)
41         return
42     }
43
44     // Parse pagination parameters
45     pageStr := r.URL.Query().Get("page")
46     pageSizeStr := r.URL.Query().Get("page_size")
47     filter := r.URL.Query().Get("filter") // New filter parameter

```

```

47 // Default values
48 page := 1
49 pageSize := 5
50
51 // Convert page and page_size to integers
52 if pageStr != "" {
53     if p, err := strconv.Atoi(pageStr); err == nil && p > 0 {
54         page = p
55     }
56 }
57 if pageSizeStr != "" {
58     if ps, err := strconv.Atoi(pageSizeStr); err == nil && ps >
59         0 && ps <= 100 {
60         pageSize = ps
61     }
62 }
63 // Calculate offset
64 offset := (page - 1) * pageSize
65
66 // Prepare response struct
67 var response PaginatedNumberDetailsResponse
68
69 // Query for total count
70 var totalCount int64
71 countQuery := "SELECT COUNT(*) FROM armstrong_numbers WHERE
72     user_id = ?"
73 countParams := []interface{}{userID}
74
75 // Apply filter if present
76 if filter == "positive" || filter == "negative" {
77     countQuery += " AND result = ?"
78     countParams = append(countParams, filter)
79 }
80 config.DB.Raw(countQuery, countParams...).Scan(&totalCount)
81
82 // Calculate total pages
83 totalPages := (int(totalCount) + pageSize - 1) / pageSize
84
85 // Query for paginated data
86 var numberDetails []NumberDetails
87 dataQuery := `
88     SELECT
89         id,
90         number,
91         result,
92         created_at
93     FROM armstrong_numbers
94     WHERE user_id = ?
95 `
96 dataParams := []interface{}{userID}
97
98 if filter == "positive" || filter == "negative" {
99     dataQuery += " AND result = ?"
100     dataParams = append(dataParams, filter)
101 }
102

```



```

103 dataQuery += " ORDER BY created_at DESC LIMIT ? OFFSET ?"
104 dataParams = append(dataParams, pageSize, offset)
105
106 err := config.DB.Raw(dataQuery, dataParams...).Scan(&
    numberDetails).Error
107
108 if err != nil {
109     log.Printf("Error querying armstrong_numbers table for
        user_id %s: %v", userID, err)
110     http.Error(w, '{"error": "Internal server error"}', http.
        StatusInternalServerError)
111     return
112 }
113
114 // Populate response
115 response = PaginatedNumberDetailsResponse{
116     NumberDetails: numberDetails,
117     TotalCount:    totalCount,
118     Page:         page,
119     PageSize:     pageSize,
120     TotalPages:   totalPages,
121 }
122
123 // Send the response
124 w.Header().Set("Content-Type", "application/json")
125 if err := json.NewEncoder(w).Encode(response); err != nil {
126     log.Printf("Error encoding response: %v", err)
127     http.Error(w, '{"error": "Internal server error"}', http.
        StatusInternalServerError)
128 }
129 }

```

Listing 7: GoLang GetUserNumbers Handler with Pagination

### 4.3 CORS Handling

Cross-Origin Resource Sharing (CORS) is a security feature implemented by browsers to restrict how resources on a web server can be requested from another domain. The backend is configured with the ‘github.com/rs/cors’ middleware to allow the frontend to make requests from a different origin.

```

1 corsHandler := cors.New(cors.Options{
2     AllowedOrigins: []string{"http://localhost:3000"},
3     AllowedMethods: []string{"GET", "POST", "PUT", "DELETE", "
        OPTIONS"},
4     AllowedHeaders: []string{"Content-Type", "Authorization"},
5     AllowCredentials: true,
6 }).Handler(r)

```

Listing 8: CORS Handler Setup in GoLang

## 5 Challenges and Solutions Implemented

### 5.1 User Identification Across Pages

**Challenge:** Maintaining user identification across different pages to ensure the user is authenticated throughout their session.

**Solution:** Implemented a `userContext` using the Context API in React. The `userContext` holds the user's credentials such as `name`, `user_id`, and `email`, and is accessible throughout the application to manage user authentication and authorization across different pages.

### 5.2 Preventing Unauthorized Access Through URL Manipulation

**Challenge:** A user trying to access the home page by manipulating the URL without logging in could bypass the authentication process.

**Solution:** Implemented a `PrivateRoute` component in React. This component checks if the user's `name`, `user_id`, and `email` fields are not null in the `userContext`. If any of these fields are null, the user is redirected to the login page.

### 5.3 Handling Invalid Inputs

**Challenge:** Ensuring valid user input, particularly for fields such as email and numeric input, to prevent issues during data processing.

**Solutions:**

- **Email Validation:** Implemented regular expressions (regex) to validate the email pattern, ensuring it follows a standard format (e.g., `user@example.com`).
- **Numeric Input Validation:** Implemented checks to ensure that only numbers are entered for fields that require numeric input, such as Armstrong number verification.

### 5.4 Handling Large Tables with Pagination

**Challenge:** Displaying large datasets such as Armstrong numbers in a user-friendly manner without overwhelming the interface or degrading performance.

**Solution:** Implemented pagination for the Armstrong numbers table. The table is split into pages, with each page displaying a limited number of records, allowing users to navigate through large datasets efficiently. Pagination is implemented both on the frontend (React) and backend (Go) to ensure smooth user experience and server efficiency.

## 6 Testing and Performance

### 6.1 Testing

To ensure the functionality and reliability of the application, manual testing for all features was conducted. Each feature was tested to verify its behavior under various scenarios, ensuring correct data processing and interaction between different components. The testing process involved checking:

- User registration and login flow.
- Armstrong number verification and storage.
- Pagination functionality.
- Input validation for fields like email and numbers.
- Proper routing and navigation between pages.

### 6.2 Performance Optimizations

To ensure optimal performance, the following optimizations were implemented:

- **Efficient API Pagination Logic:** Pagination was optimized to handle large data efficiently. The backend queries were optimized to return only the required set of results based on the current page and page size, thus reducing the amount of data transferred and improving API response times.

## 7 Future Enhancements

The following enhancements are planned for future versions of the project to improve functionality, scalability, and user experience:

- **Email Verification:** Implement email verification during user registration to ensure valid email addresses. A verification email will be sent to the user to confirm their email before they can log in.
- **Password Authentication:** Introduce password authentication for user accounts to improve security. This will involve adding password fields during user registration and login, using secure hashing algorithms to store passwords in the database.
- **Enhanced Pagination and Filtering:** Enhance the pagination and filtering logic to support more complex queries, such as sorting by multiple fields or filtering by additional parameters like date range or result type (positive/negative).
- **Internationalization and Localization:** Implement internationalization (i18n) to support multiple languages. This would involve translating the user interface and handling various date and number formats based on the user's region.

## 8 Conclusion

In this project, a robust web application was developed to verify Armstrong numbers, manage user data, and display results efficiently. Key features of the project include:

- **User Context:** Implemented `userContext` using the Context API to manage user details across different pages, ensuring a seamless experience.
- **Protected Routes:** Utilized private routes to restrict access to certain pages based on user credentials, enhancing security by preventing unauthorized access.
- **Pagination:** Implemented pagination to handle large datasets efficiently, ensuring smooth navigation through large tables of Armstrong numbers.
- **CORS Implementation:** Configured CORS settings to ensure secure communication between the frontend and backend, allowing specific origins and enabling safe cross-origin requests.
- **Input Validation:** Applied input validation techniques like regex for email verification and checks for valid numeric inputs, ensuring data integrity and preventing errors.

These features collectively ensure that the application is user-friendly, secure, and capable of handling large amounts of data efficiently, providing a solid foundation for further enhancements and scalability.

## Appendix: UI Screenshots

### Login Page

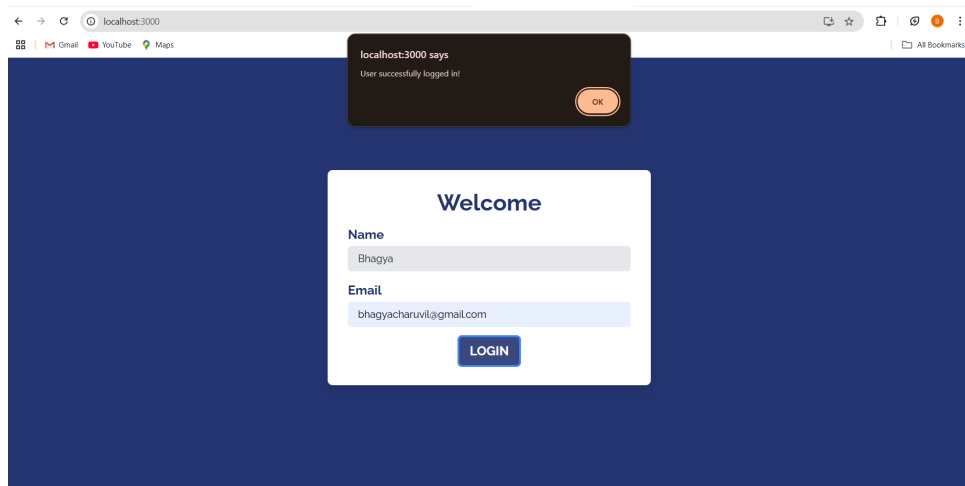


Figure 1: Login Page

### Home Page

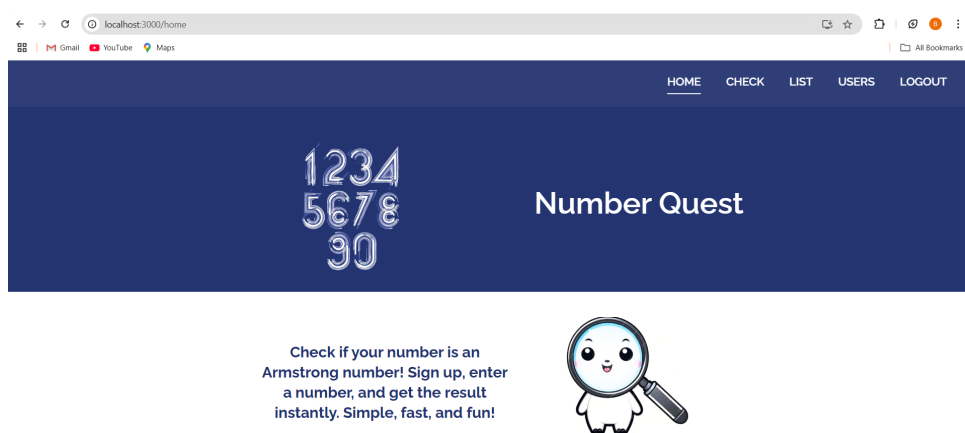


Figure 2: Landing Page

## Verification Page

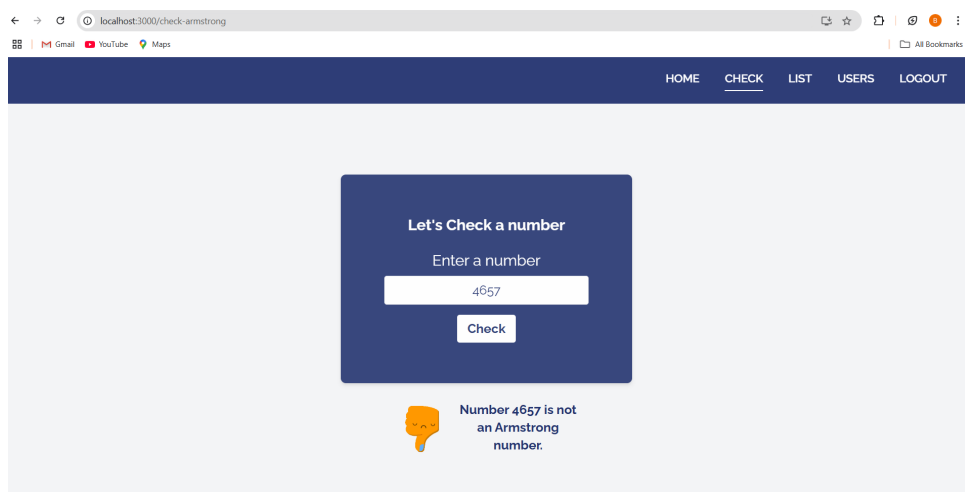


Figure 3: Verification page

## User Details Page

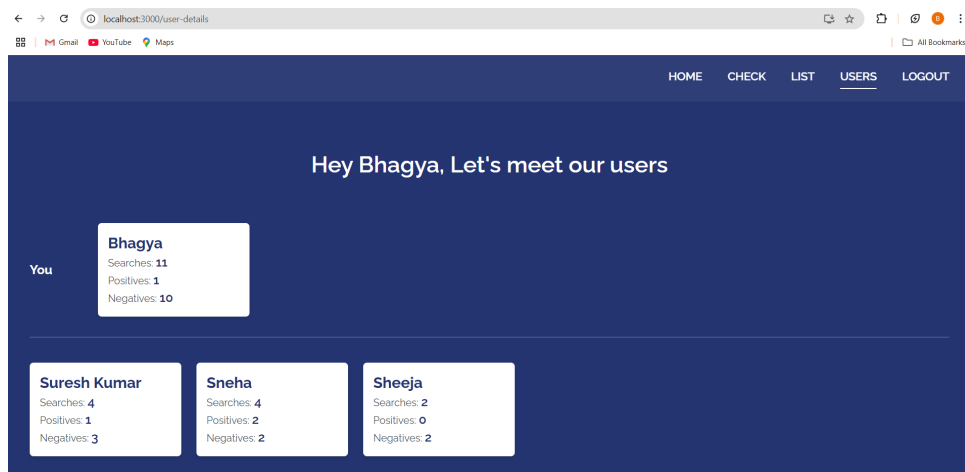
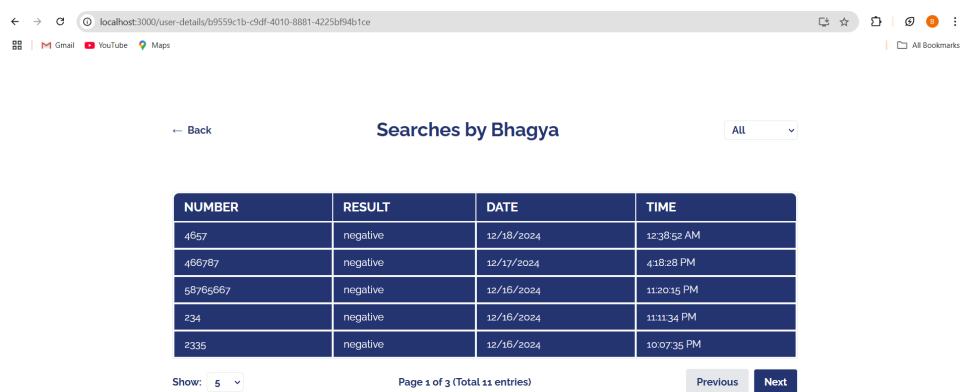


Figure 4: Page showing user details

## Number Details Page



← Back

Searches by Bhagya

All

NUMBER	RESULT	DATE	TIME
4657	negative	12/18/2024	12:38:52 AM
466787	negative	12/17/2024	4:18:28 PM
68769667	negative	12/16/2024	11:20:15 PM
234	negative	12/16/2024	11:11:34 PM
2335	negative	12/16/2024	10:07:35 PM

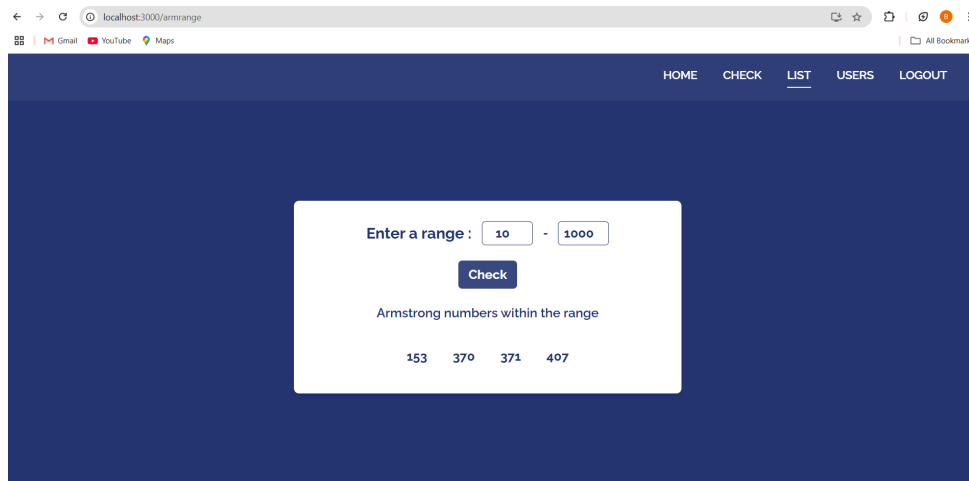
Show: 5

Page 1 of 3 (Total 11 entries)

Previous Next

Figure 5: Table showing details of numbers entered by a user with pagination

## Armstrong Nos within range page



HOME CHECK LIST USERS LOGOUT

Enter a range : 10 - 1000

Check

Armstrong numbers within the range

153 370 371 407

Figure 6: Page that returns the list of armstrong nos within range 10-1000