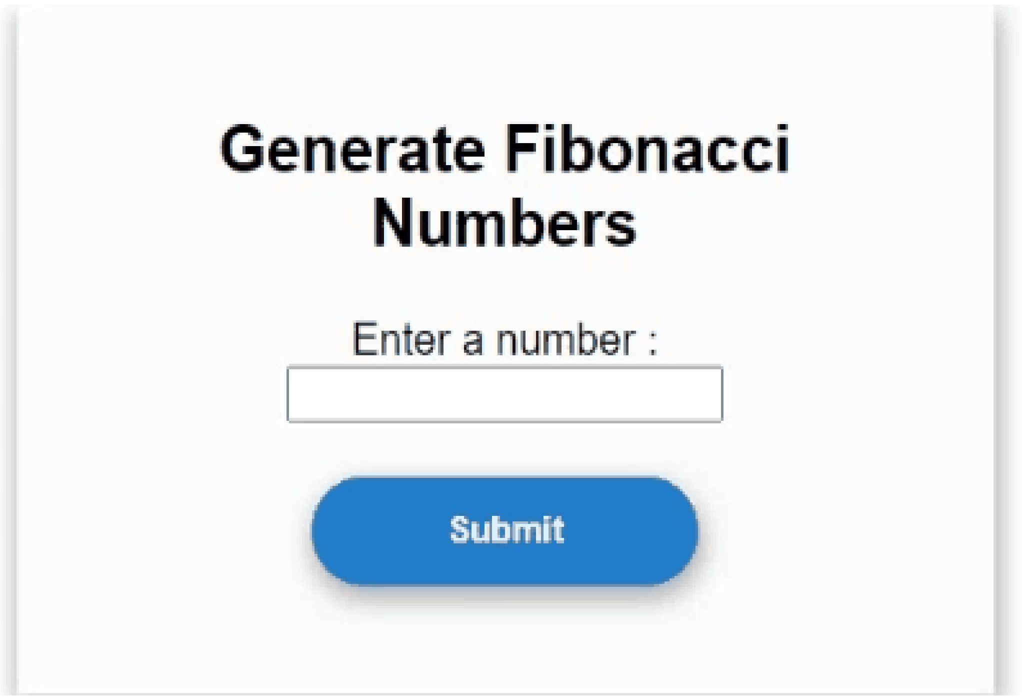
# Take-home challenge for fullstack software engineering at CZ Biohub SF

By Prashasti Karlekar

## Introduction

The Fibonacci Web App is a simple web application that allows users to generate the first 'n' Fibonacci numbers, where 'n' is an integer provided by the user. The app is built using React.js for the frontend and Node.js with Express.js for the backend. The backend stores computed Fibonacci numbers in a PostgreSQL database to avoid recomputation and improve performance.

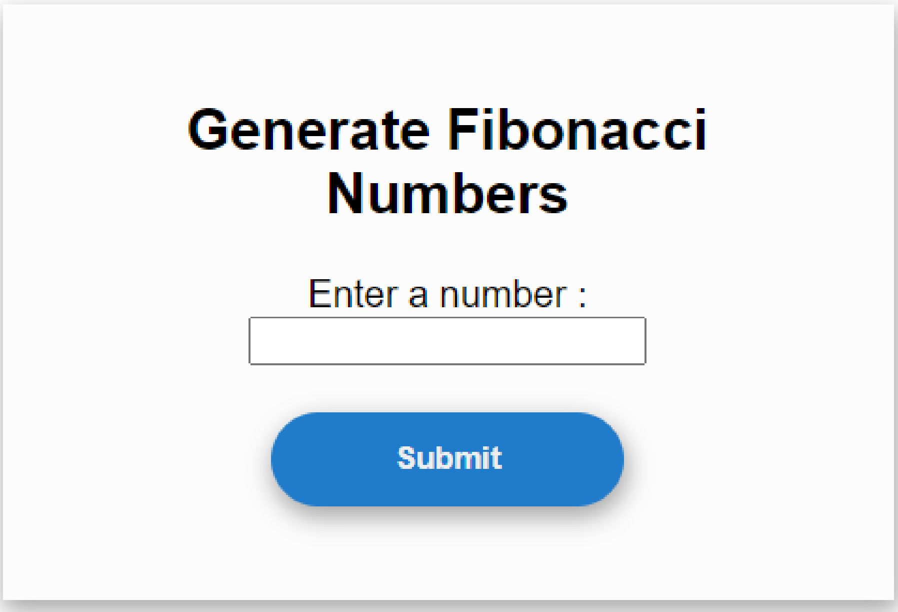


## Logic and Design

### Frontend

#### Step 1: User Input

The frontend presents the user with a simple form on the first page. The form contains a single input field where the user can enter the value of 'n', which represents the number of Fibonacci numbers to generate. A 'Submit' button triggers the form submission.

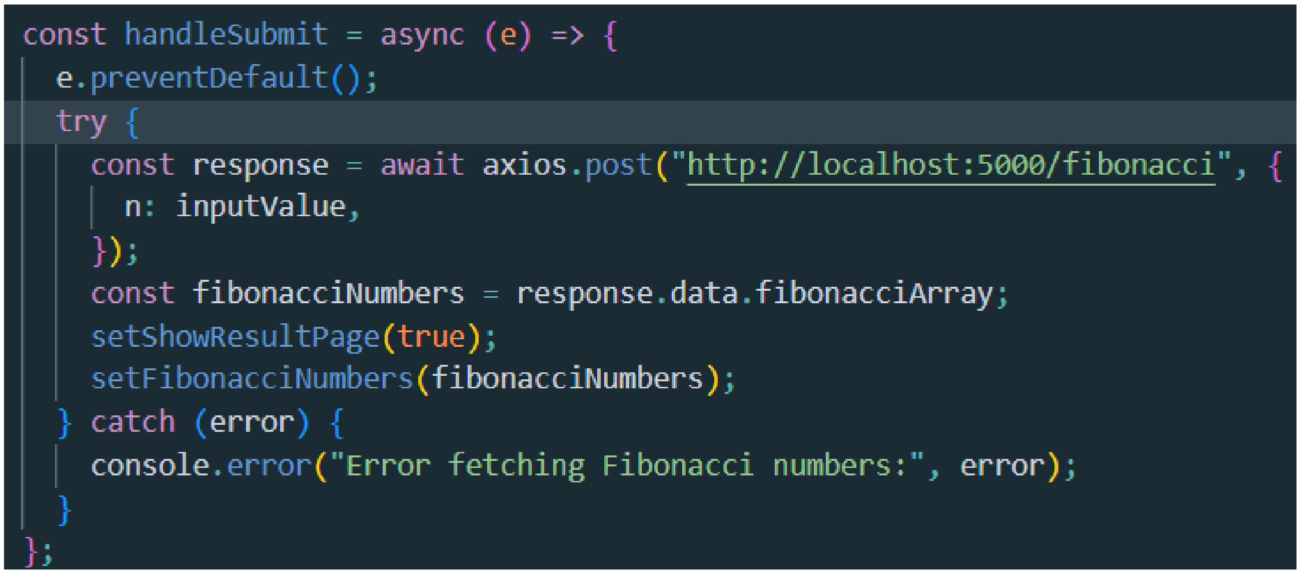


Step 2: Form Submission

Upon form submission, the frontend sends an API request to the backend to calculate the first 'n' Fibonacci numbers.

Step 3: API Request

The frontend uses Axios, an HTTP client library, to make a POST request to the backend's '/fibonacci' endpoint. The request includes the 'n' value entered by the user.



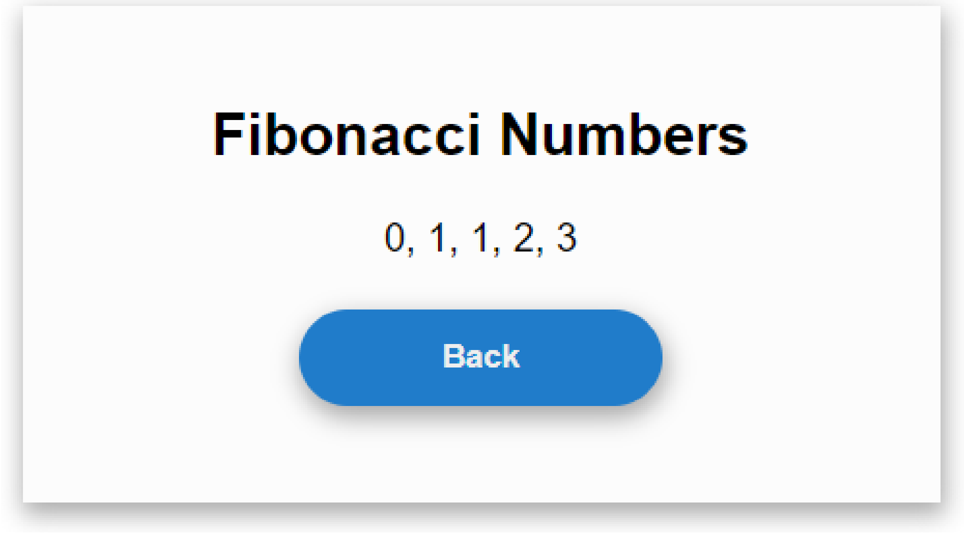
```
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const response = await axios.post("http://localhost:5000/fibonacci", {
      n: inputValue,
    });
    const fibonacciNumbers = response.data.fibonacciArray;
    setShowResultPage(true);
    setFibonacciNumbers(fibonacciNumbers);
  } catch (error) {
    console.error("Error fetching Fibonacci numbers:", error);
  }
};
```

Step 4: API Response

The backend computes the first 'n' Fibonacci numbers and sends them back as a response to the frontend.

Step 5: Display Fibonacci Numbers

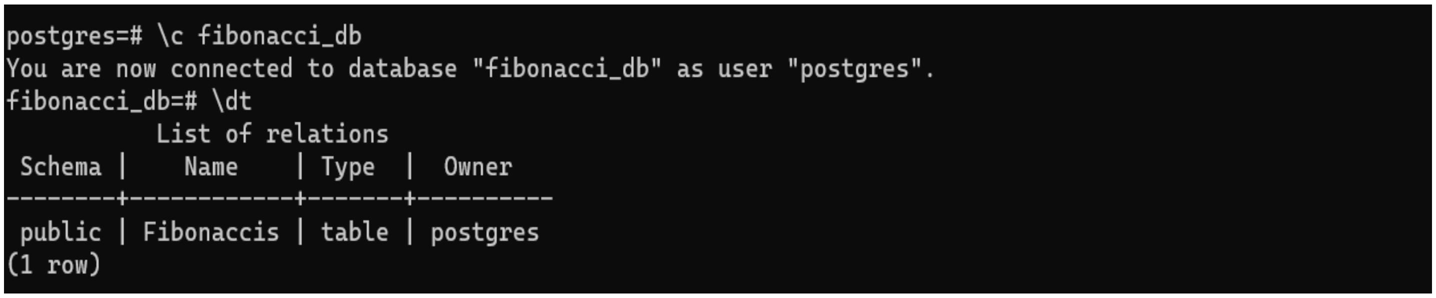
The frontend receives the API response containing the computed Fibonacci numbers. It then redirects the user to the second page - Result, where the Fibonacci numbers are displayed as a comma-separated list.



## Backend

Step 1: PostgreSQL Database

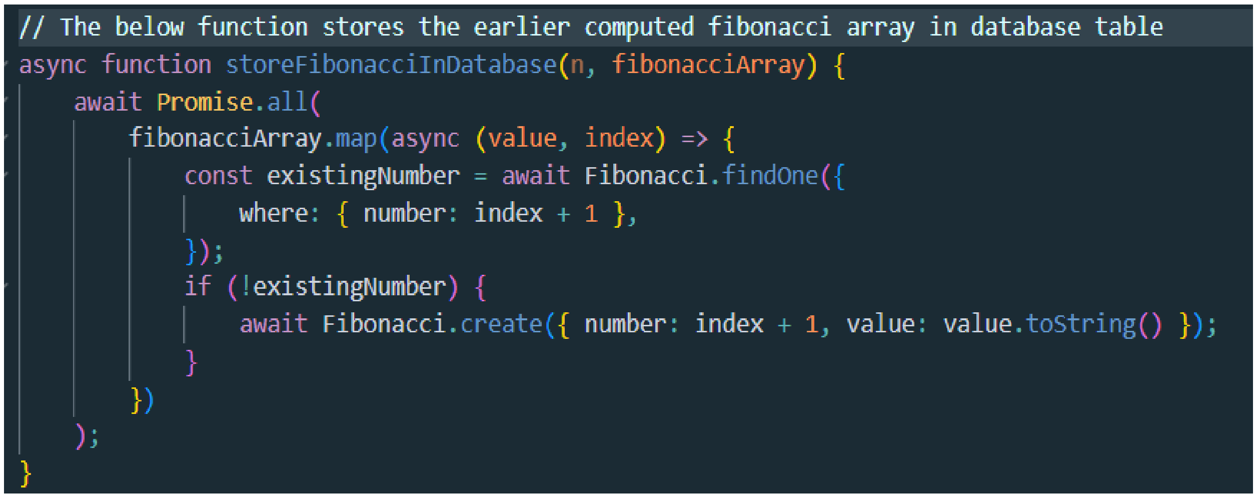
The backend is set up with Express.js and uses Sequelize as an ORM to interact with a PostgreSQL database. The database is responsible for storing the computed Fibonacci numbers to prevent redundant calculations.



```
postgres=# \c fibonacci_db
You are now connected to database "fibonacci_db" as user "postgres".
fibonacci_db=# \dt
           List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | Fibonaccis | table | postgres
(1 row)
```

## Step 2: Compute Fibonacci Numbers

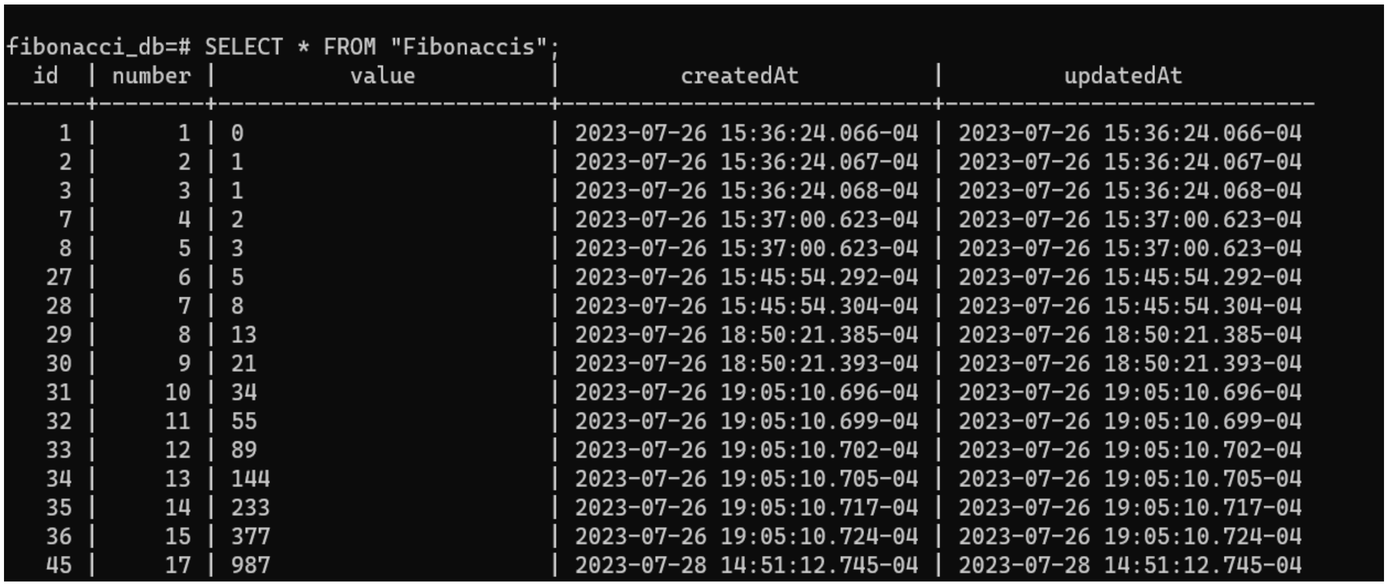
When the backend receives a POST request to '/fibonacci', it first checks if the Fibonacci numbers for the given 'n' value already exist in the database. If they do, it retrieves them from the database and sends them back as a response.



```
// The below function stores the earlier computed fibonacci array in database table
async function storeFibonacciInDatabase(n, fibonacciArray) {
    await Promise.all(
        fibonacciArray.map(async (value, index) => {
            const existingNumber = await Fibonacci.findOne({
                where: { number: index + 1 },
            });
            if (!existingNumber) {
                await Fibonacci.create({ number: index + 1, value: value.toString() });
            }
        })
    );
}
```

## Step 3: Store Fibonacci Numbers

If the Fibonacci numbers for the given 'n' value do not exist in the database, the backend calculates them using a recursive algorithm. It then stores the computed Fibonacci numbers in the database, associating each number with its corresponding 'n' value.



```
fibonacci_db=# SELECT * FROM "Fibonaccis";
 id  | number |           value           |          createdAt           |          updatedAt
-----+--------+---------------------------+------------------------------+----------------------------
   1 |      1 | 0                         | 2023-07-26 15:36:24.066-04 | 2023-07-26 15:36:24.066-04
   2 |      2 | 1                         | 2023-07-26 15:36:24.067-04 | 2023-07-26 15:36:24.067-04
   3 |      3 | 1                         | 2023-07-26 15:36:24.068-04 | 2023-07-26 15:36:24.068-04
   7 |      4 | 2                         | 2023-07-26 15:37:00.623-04 | 2023-07-26 15:37:00.623-04
   8 |      5 | 3                         | 2023-07-26 15:37:00.623-04 | 2023-07-26 15:37:00.623-04
  27 |      6 | 5                         | 2023-07-26 15:45:54.292-04 | 2023-07-26 15:45:54.292-04
  28 |      7 | 8                         | 2023-07-26 15:45:54.304-04 | 2023-07-26 15:45:54.304-04
  29 |      8 | 13                        | 2023-07-26 18:50:21.385-04 | 2023-07-26 18:50:21.385-04
  30 |      9 | 21                        | 2023-07-26 18:50:21.393-04 | 2023-07-26 18:50:21.393-04
  31 |     10 | 34                        | 2023-07-26 19:05:10.696-04 | 2023-07-26 19:05:10.696-04
  32 |     11 | 55                        | 2023-07-26 19:05:10.699-04 | 2023-07-26 19:05:10.699-04
  33 |     12 | 89                        | 2023-07-26 19:05:10.702-04 | 2023-07-26 19:05:10.702-04
  34 |     13 | 144                       | 2023-07-26 19:05:10.705-04 | 2023-07-26 19:05:10.705-04
  35 |     14 | 233                       | 2023-07-26 19:05:10.717-04 | 2023-07-26 19:05:10.717-04
  36 |     15 | 377                       | 2023-07-26 19:05:10.724-04 | 2023-07-26 19:05:10.724-04
  45 |     17 | 987                       | 2023-07-28 14:51:12.745-04 | 2023-07-28 14:51:12.745-04
```

## Reasons for Choosing this Tech Stack

Using React.js for the frontend, Node.js with Express.js, PostgreSQL and Sequelize for the backend was a good option for this Fibonacci web app due to the following reasons:

- Node.js follows an asynchronous, non-blocking I/O model, making it highly efficient for handling concurrent requests. This is beneficial for this web application since it may experience multiple simultaneous user interactions.
- Node.js is built on Chrome's V8 JavaScript engine, which provides high-performance execution. It is lightweight compared to other server-side technologies, making it suitable for small to medium-sized applications like the Fibonacci web app.
- Express.js is a minimalist web framework for Node.js which provides essential features for building web applications, such as routing, middleware support, and simplified HTTP request handling, without adding unnecessary overhead.
- I used Sequelize, which is an ORM for Database Interaction that allows to work with databases using JavaScript objects instead of raw SQL queries, making the code more maintainable and expressive.
- PostgreSQL is a robust, scalable, and open-source relational database management system, which offers excellent performance, and support for large datasets, making it a good choice for storing and managing Fibonacci numbers.
- In terms of modularity and maintainability, Node.js and Express.js support modular code organization by allowing to break the application into separate modules (e.g., routes, services, models). This makes it easier to understand, maintain, and scale the codebase as the app grows.'
- Using React.js for frontend was a good option for this app because it easily provided a dynamic and interactive user interface, facilitated efficient state management and reusable components, enhancing the user experience and development productivity.

## Constraints and Considerations

- Input Validation: Both the frontend and backend must validate user inputs to ensure 'n' is a positive integer. Invalid inputs should be handled with appropriate error messages.
- Duplicate Number Handling: The backend must check whether the Fibonacci numbers for a given 'n' value already exist in the database to avoid recomputation and prevent duplicate entries.
- PostgreSQL's performance can degrade for extremely large 'n' values due to the recursive nature of the Fibonacci computation, potentially causing slower response times.
- High read and write loads from multiple concurrent users generating Fibonacci numbers may affect database performance, necessitating scaling strategies to handle increased traffic.
- Performance Optimization: Caching mechanisms and database indexing can be implemented to enhance the app's performance and reduce the need for frequent database queries.

- Security Measures: Sanitize user inputs to prevent SQL injection and configure CORS on the backend to allow requests only from trusted origins.
- Scalability and Deployment: Load balancing and database sharding can be considered for potential scaling needs. CI/CD pipelines should be set up for automated testing and deployment.
- Testing: Writing unit tests for critical parts of the application, such as the Fibonacci computation function and implementing end-to-end tests to cover the entire user flow.

## Conclusion

The Fibonacci Web App provides a straightforward way for users to generate the first 'n' Fibonacci numbers. The frontend allows users to input 'n' and triggers an API request to the backend for computation. The backend efficiently computes and stores Fibonacci numbers in a database, ensuring optimal performance and avoiding unnecessary recomputation. The app's design and logic ensure a user-friendly experience while adhering to constraints and best practices for performance, security, and maintainability.