

CODEBLOOM: CODING PLATFORM APPLICATION

*Submitted in partial fulfillment of the requirements for the award of the degree of
Bachelor of Technology
In
Information Technology*



Submitted by
Priyamvada Priyadarshani IIB2020037

Under the
Supervision of
Prof. Nabajyoti Mazumdar

Information Technology
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY
ALLAHABAD
Prayagraj -211015

TABLE OF CONTENTS

Abstract.....	2
I. Introduction.....	2
II. Tools, Framework, Technologies, and Some Important Libraries Used.....	3
III. Architectural Design and Functionality.....	6
Authentication & Recoil:.....	7
Firebase:.....	8
Backend.....	10
Database Schemas:.....	14
Hydration error:.....	17
IV. Deployment.....	17
Conclusion.....	18
Future Scope/ Guide to Implement for a Start-up.....	19
Supplementary Material.....	20
References.....	20

Abstract

Code Bloom, an innovative coding practice platform, addresses the challenge of search engine visibility in a saturated coding website landscape. Focusing on SEO friendliness, the platform employs advanced page-building strategies to pre-render content, optimizing search engine indexing and enhancing rankings.

The platform strategically balances rendering issues by incorporating Static Site Generation (SSG) for infrequently changing data and Server-Side Rendering (SSR) for dynamic content. A hybrid approach ensures periodic content refresh while maintaining the benefits of a static site.

Trade-offs between user experience and load time, security and convenience (leveraging Firebase), and managing local and dynamic databases are successfully planned. The platform also tackles the hydration error through a thoughtful implementation, ensuring compatibility with React's expectations during rendering.

Code Bloom stands out with a minimalistic approach, offering a concise and intuitive coding experience. With a user-friendly interface, support for multiple languages, and robust authentication and database management using Firebase, Code Bloom provides a seamless environment for users to express preferences, bookmark challenges, and experience the joy of coding.

Keyword: CodeBloom, Coding, SEO, SSG, SSR

I. Introduction

In response to industry challenges, CodeBloom addresses the crucial issue of search engine visibility. By prioritizing Search Engine Optimization (SEO) friendliness, the platform employs innovative page-building strategies, ensuring pre-rendered content to enhance search engine indexing and rankings. It offers a unique blend of Static Site Generation (SSG) and Server-Side Rendering (SSR) to balance speed and dynamic content. A hybrid approach, incorporating a revalidate key in `getStaticProps`, allows periodic content refresh while maintaining the advantages of a static site.

Trade-offs were successfully planned, balancing user experience with load time and security with convenience. The platform uses a mixed rendering approach, storing static data locally for faster page generation through SSG and handling dynamic data through SSR using databases or Firestore for optimal performance.

The solution provided by CodeBloom is not only technically robust but also addresses practical challenges, such as solving hydration errors and maintaining code readability for easy modifications.

CodeBloom, is a coding platform that aims to redefine the coding platform utilities experience. Driven by the necessity for a contemporary, user-friendly, and collaborative coding environment, the journey of creating CodeBloom began. Leveraging the capabilities of React, Next.js, Tailwind CSS, and Firebase, CodeBloom provides coders with a seamless, secure, and intuitive interface to enhance their coding skills. The platform boasts a meticulously curated collection of coding challenges, facilitating efficient solution submissions. This work aims to highlight the substantial benefits of CodeBloom in skill development and its role in fostering a collaborative coding ecosystem.

Motivation:

The motivation behind CodeBloom stems from my daily coding experiences on various platforms, sparking a natural curiosity about their inner workings. This curiosity led to a deeper exploration of platform creation, providing a practical means to enhance coding skills. Beyond platform creation, CodeBloom's development sought to illuminate the intricacies of coding ecosystems, addressing considerations such as performance versus maintainability and front-end frameworks versus page load time.

To simplify and expedite user interactions, CodeBloom boasts a friendly, familiar, visually appealing, and responsive interface from the moment users engage with the platform.

Target Audience:

CodeBloom is meticulously designed for a discerning audience of coders, catering to aspiring beginners and seasoned professionals alike. Positioned as an inclusive space for skill development, the platform beckons to enthusiasts entering the coding realm and experienced developers seeking a collaborative and enriching coding environment. CodeBloom invites individuals eager to explore, learn, and contribute to a collaborative coding ecosystem.

In its early stages, CodeBloom features a curated collection of introductory Data Structures and Algorithms (DSA) problems. However, its foundational architecture is poised for expansive evolution, positioning it as a dynamic platform ready to accommodate diverse challenges transcending the boundaries of DSA. A significant aspect of its future development involves the incorporation of a multifaceted language code editor, expanding its linguistic horizons beyond the current JavaScript exclusivity.

II.Tools, Framework, Technologies, and Some Important Libraries Used

In my project, I integrated key technologies like React, Next.js, TypeScript, and Tailwind CSS to develop a robust and user-friendly coding platform. React, a widely used JavaScript library, employs a component-based approach for building interfaces, ensuring easier development and maintenance of complex UIs. By utilizing React, I established a scalable and efficient foundation for the frontend of the coding platform.

Next.js, built on top of React, provides advanced features such as server-side rendering and static site generation. I chose Next.js for its simplicity and performance optimizations, aiming to enhance the project's performance, improve SEO, and achieve smoother user experiences.

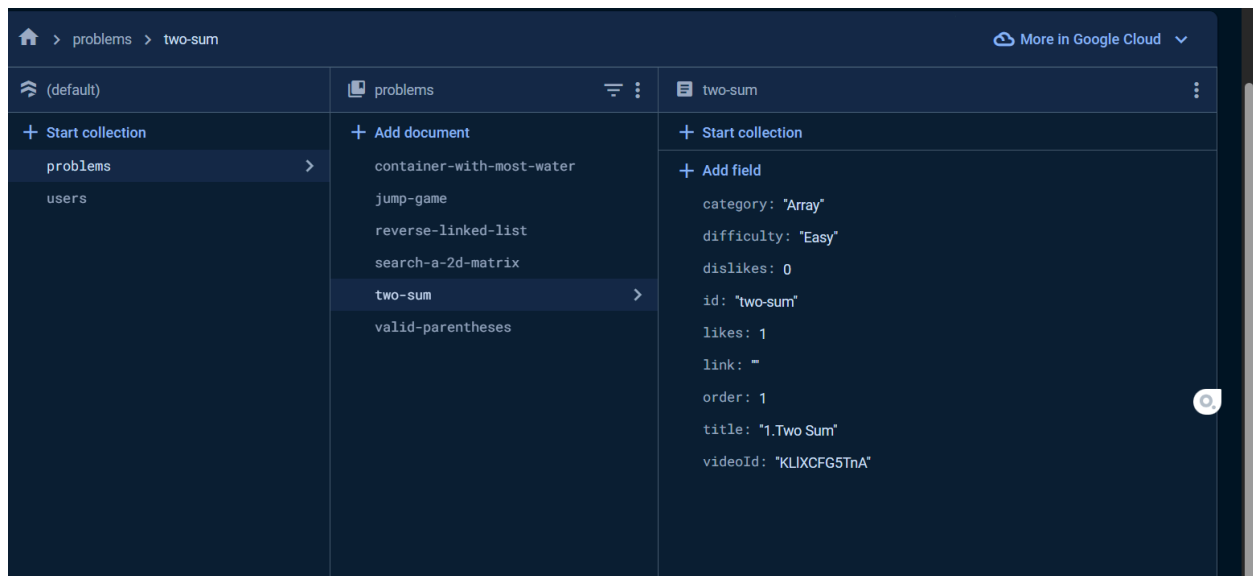
TypeScript, a typed superset of JavaScript, was incorporated to enhance code maintainability and scalability. Its strong type system and editor support contribute to error-catching early in the development process, improving code readability and navigation.

Tailwind CSS, a customizable CSS framework, takes a utility-first approach, streamlining the styling process and reducing CSS complexity. This framework facilitated the creation of a visually appealing and responsive user interface.

For security and authentication, I implemented Firebase Authentication and Firebase Database. Firebase Authentication ensures secure user management with features like email/password authentication and social media sign-in options. Firebase Database, a NoSQL cloud database, provides real-time synchronization and scalability for efficient data retrieval and storage.

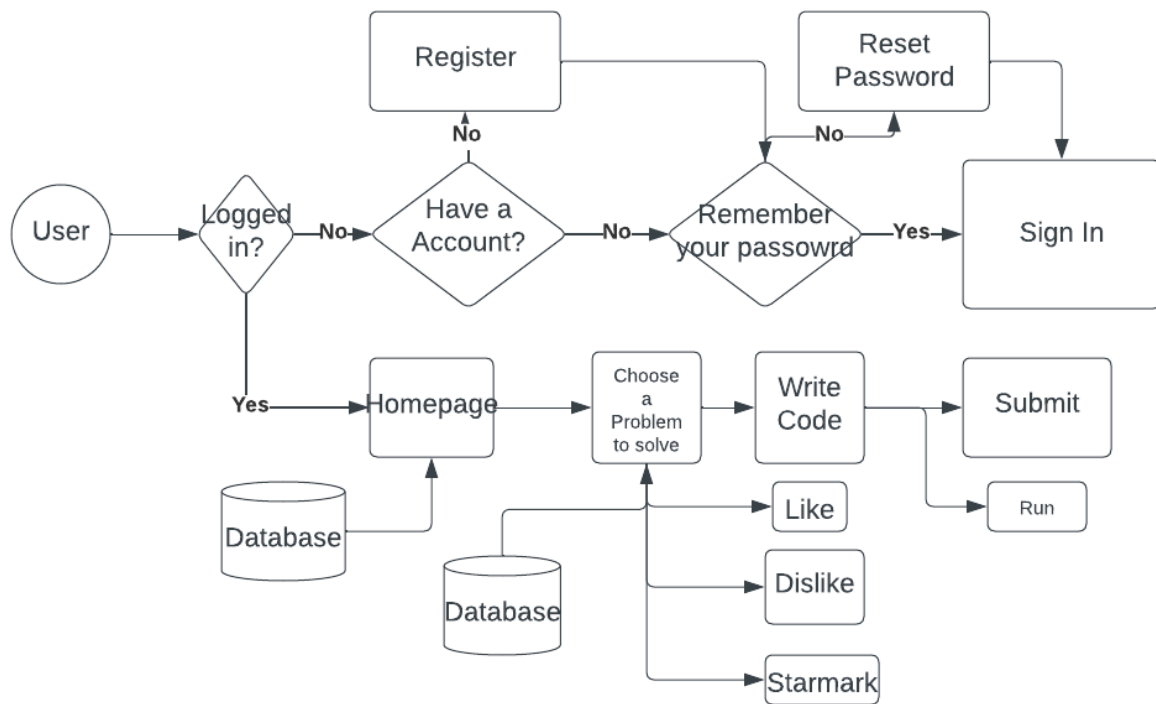
Firebase's real-time synchronization feature enables instant updates across devices, fostering collaboration in real-time coding discussions. The platform's robust infrastructure ensures scalability and reliability, crucial for handling varying levels of user activity.

Additionally, GitHub, a web-based collaboration platform, offers version control and collaboration features for software developers. It enables code repository management, change tracking, collaborative project management, and hosting in the cloud. Alternatives to GitHub include GitLab and Bitbucket. VSCode is used for code editing.



III. Architectural Design and Functionality

URL DIAGRAM/FLOWCHART:



Homepage:

CodeBloom Premium Sign In

" DEDICATION FUELS YOUR CODE, PRACTICE WITH PASSION AND WATCH IT EXPLODE! "

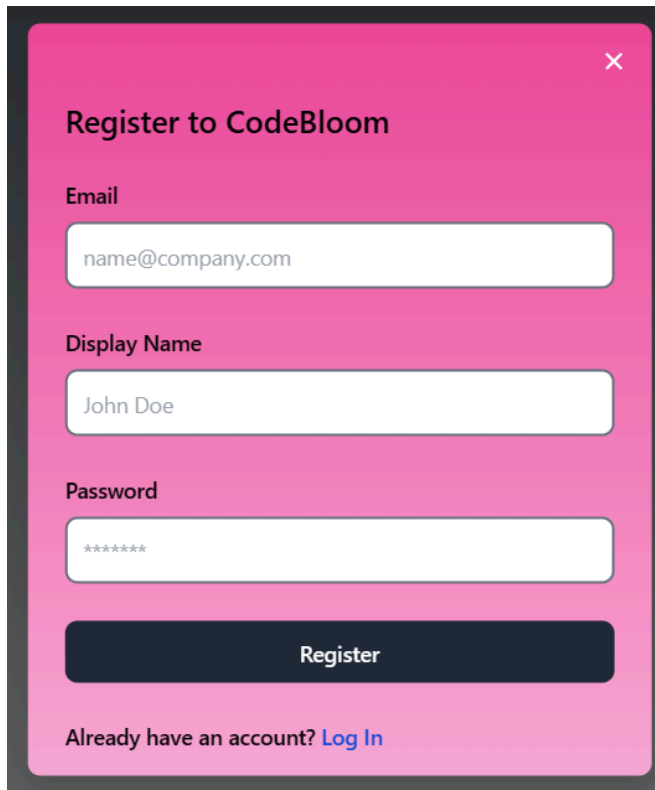
STATUS	TITLE	DIFFICULTY	CATEGORY	SOLUTION
	1.Two Sum	Easy	Array	
	2.Reverse Linked List	Hard	Linked List	Coming soon
	3. Jump Game	Medium	Dynamic Programming	Coming soon
	4. Valid Parentheses	Easy	Stack	Coming soon
	5. Search a 2D Matrix	Medium	Binary Search	Coming soon
	6. Container With Most Water	Medium	Two Pointers	Coming soon

From the Premium tab one may subscribe, etc, for now it leads to my medium account. On clicking on Sign in, you will be taken to the Register and Login pages.

Authentication & Recoil:

Since the background, overlay close button etc are common in the login/sign up/reset have a lot in common we made a modal layout of them to make things easier so that we don't have to write it over and over again. React icon <IoClose> was used.

We used global state so that user can dynamically navigate between the modals. We have in our applications different parts that are using the same state, that's why we use global state and to use it we need to use recoil(State management library for react). In recoil, we have something called atoms which represent a different state we are going to use in our application. Default type value of AuthModal state was set to login.



Register to CodeBloom

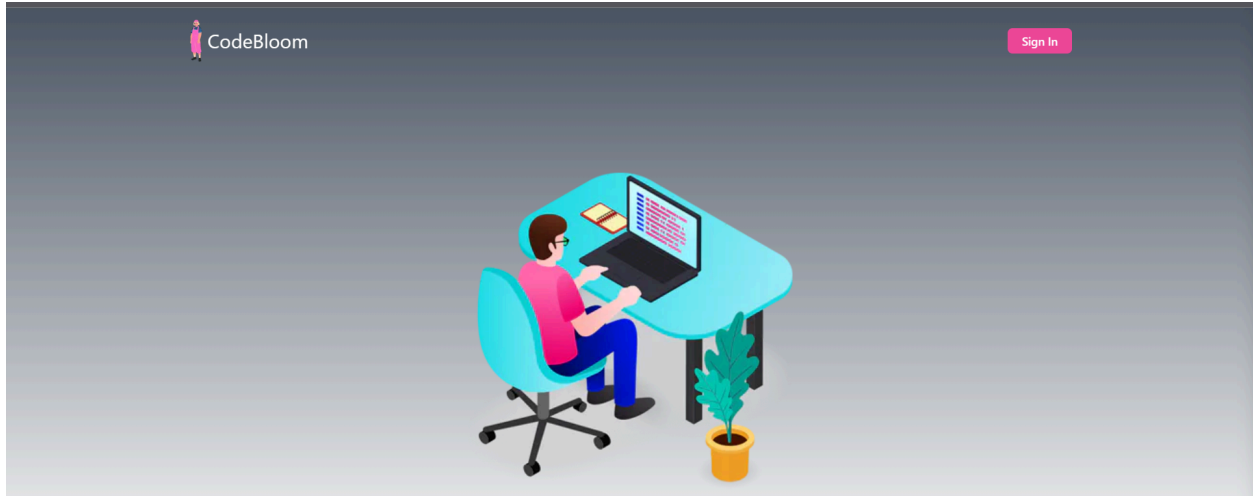
Email

Display Name

Password

Register

Already have an account? [Log In](#)



Firestore:

In building CodeBloom, integrating Firestore has been a game-changer, taking the development experience to the next level. Firestore, known for its Backend-as-a-Service (BaaS) features, allows us to create a full-stack app without dealing with the complexities of server management. This smart move not only makes the development process smoother but also ensures security through Firestore's built-in authentication and authorization.

Firestore is like a powerful toolkit that simplifies many aspects of the project, making CodeBloom more efficient and secure.

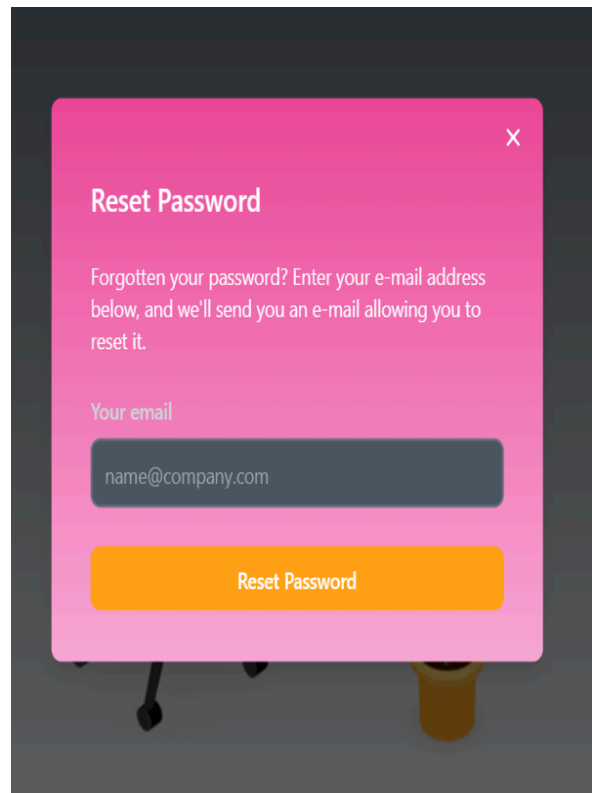
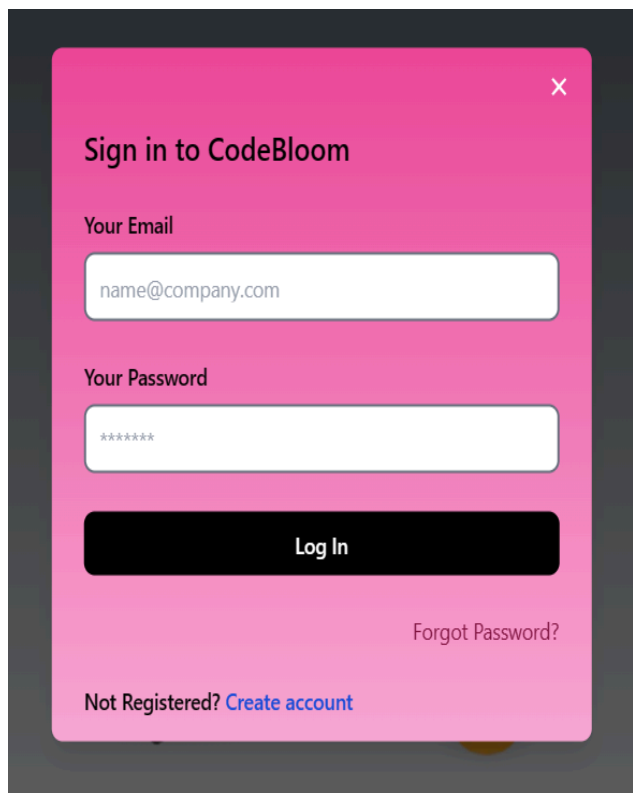
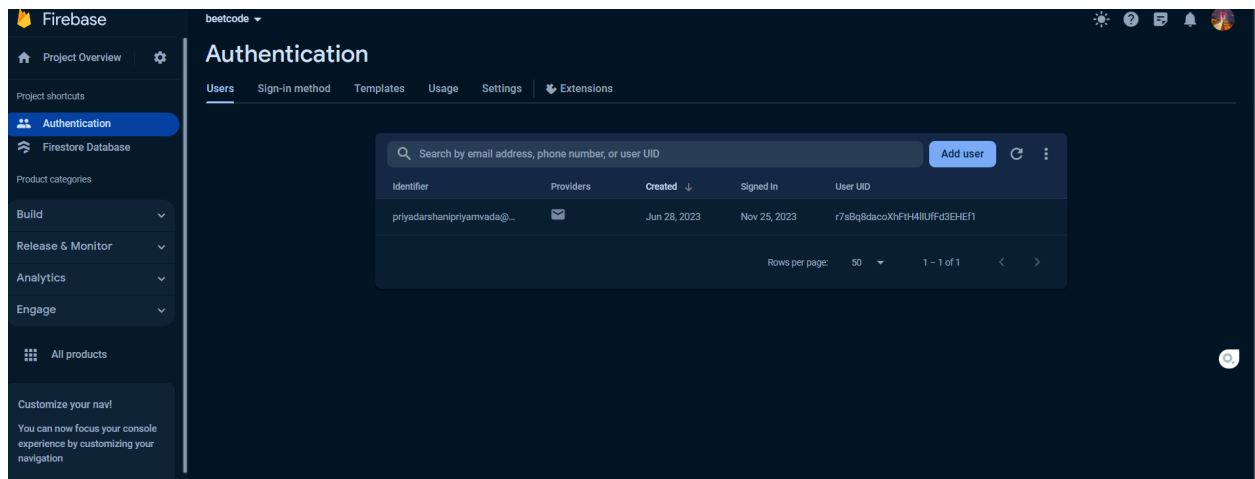
At the start of CodeBloom, I created a Firestore account using the Firestore Console. In the console, I crafted a project named CodeBloom, serving as a central hub for all our Firestore configurations and services.

The Firestore initialization code is neatly organized within a dedicated 'firebase' folder in the project structure. This decision ensures clarity and maintainability as we move through the development process.

To boost collaboration between React and Firestore, I smoothly integrated the react-firestore-hooks package using npm install. This powerful library introduces hooks that seamlessly blend Firestore functionalities into our React components. For instance, the useAuthState hook helps keep track of user authentication status in real-time, bringing dynamism to our application.

Delving into Firestore intricacies, the prioritization of establishing security rules is emphasized. These rules are thoughtfully configured in the Firestore Console, contributing to the

reinforcement of our application's security. This meticulous setup ensures that access to sensitive data is guided by a robust security structure.



Reset the password using the email. Once you enter your email, through your email you will be able to reset your password.

Backend

The public folder contains the logos and images required.

Images used in the app were taken from Google, and icons were created at Canva.

Setting up tailwind.config.js to define template material for the project.

>public

























>Src:

























>Atoms

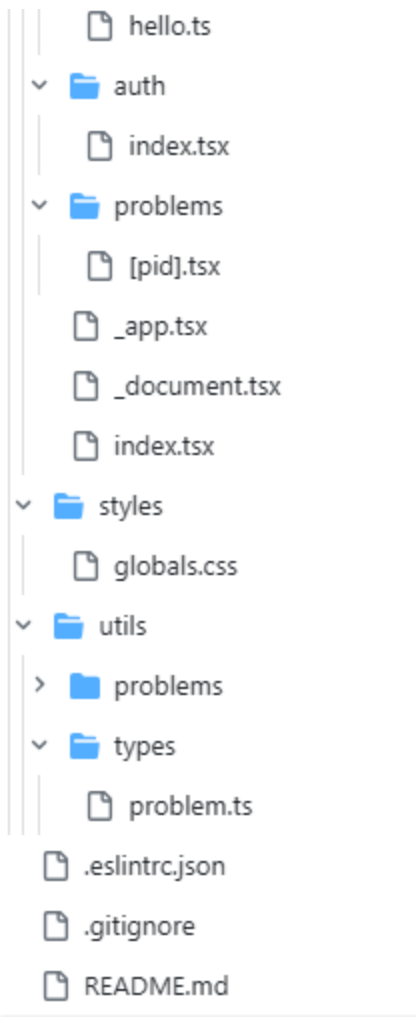
>Components [for reuse]

- Buttons
- Modals
- Navbar
- Skeletons
- Timer
- Topbar
- Workspace

- Firebase
- Hooks
- Mock problems
- Pages
 - Api
 - Auth
 - problems
- Styles: the global.css file
- Utils:
 - Problem
 - types

- >  public
- ▼  src
 - >  atoms
 - ▼  components
 - ▼  Buttons
 -  Logout.tsx
 - ▼  Modals
 -  AuthModal.tsx
 -  Login.tsx
 -  ResetPassword.tsx
 -  SettingsModal.tsx
 -  Signup.tsx
 - ▼  Navbar
 -  Navbar.tsx
 - ▼  ProblemsTable
 -  ProblemsTable.tsx
 - ▼  Skeletons
 -  CircleSkeleton.tsx
 -  RectangleSkeleton.tsx
 - ▼  Timer
 -  Timer.tsx
 - ▼  Topbar
 -  Topbar.tsx
 - ▼  Workspace

- ▼  Workspace
 - ▼  Playground
 - ▼  PreferenceNav
 -  PreferenceNav.tsx
 -  EditorFooter.tsx
 -  Playground.tsx
 - ▼  ProblemDescription
 -  ProblemDescription.tsx
 -  Workspace.tsx
 - ▼  firebase
 -  firebase.ts
 - ▼  hooks
 -  useHasMounted.ts
 -  useLocalStorage.ts
 -  useWindowSize.ts
 - ▼  mockProblems
 -  problems.ts
 - ▼  pages
 - ▼  api
 -  hello.ts
 - ▼  auth
 -  index.tsx
 - ▼  problems
 -  index.tsx



Homepage after Signing in

CodeBloom

Premium

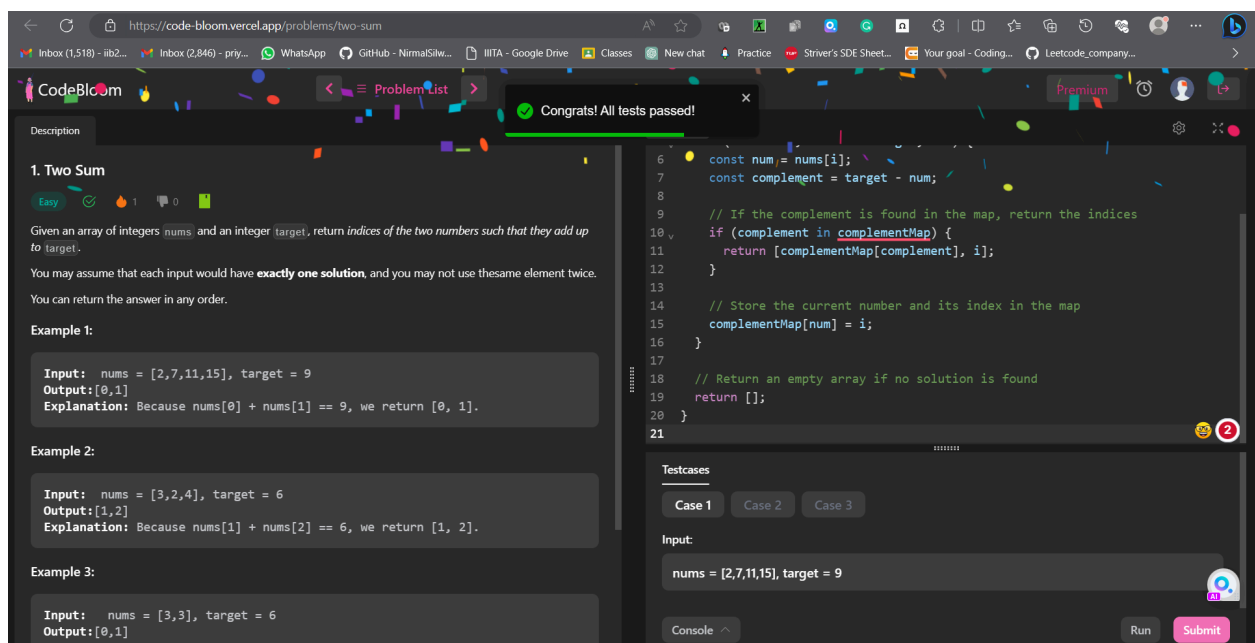
priyadarshanipriyamvada@gmail.com

"DEDICATION FUELS YOUR CODE, PRACTICE WITH PASSION AND WATCH IT EXPLODE!"

STATUS	TITLE	DIFFICULTY	CATEGORY	SOLUTION
✓	1.Two Sum	Easy	Array	
	2.Reverse Linked List	Hard	Linked List	Coming soon
	3. Jump Game	Medium	Dynamic Programming	Coming soon
✓	4. Valid Parentheses	Easy	Stack	Coming soon
	5. Search a 2D Matrix	Medium	Binary Search	Coming soon
	6. Container With Most Water	Medium	Two Pointers	Coming soon

On the homepage, a curated list of problems unfolds, currently featuring six challenges with plans to expand this repertoire. Each problem is accompanied by key details including title, difficulty level, category, solution status, and a unique click-to-solution feature. Upon activation, the solution link seamlessly directs users to a video solution hosted on a YouTube channel, with potential integration to a designated website or personal YouTube channel in the future. The forthcoming integration with a YouTube channel is a compelling enhancement that will be showcased in subsequent updates.

The mock problem folder has a file `problem.ts` which has the hardcoded description of the problems available on the app.



Problem Page: The problem page on our platform offers a rich learning experience, featuring a detailed problem description with examples, constraints, and a dedicated code editor supporting JavaScript. Users can seamlessly switch between test cases, dynamically updating the UI to enhance their coding experience.

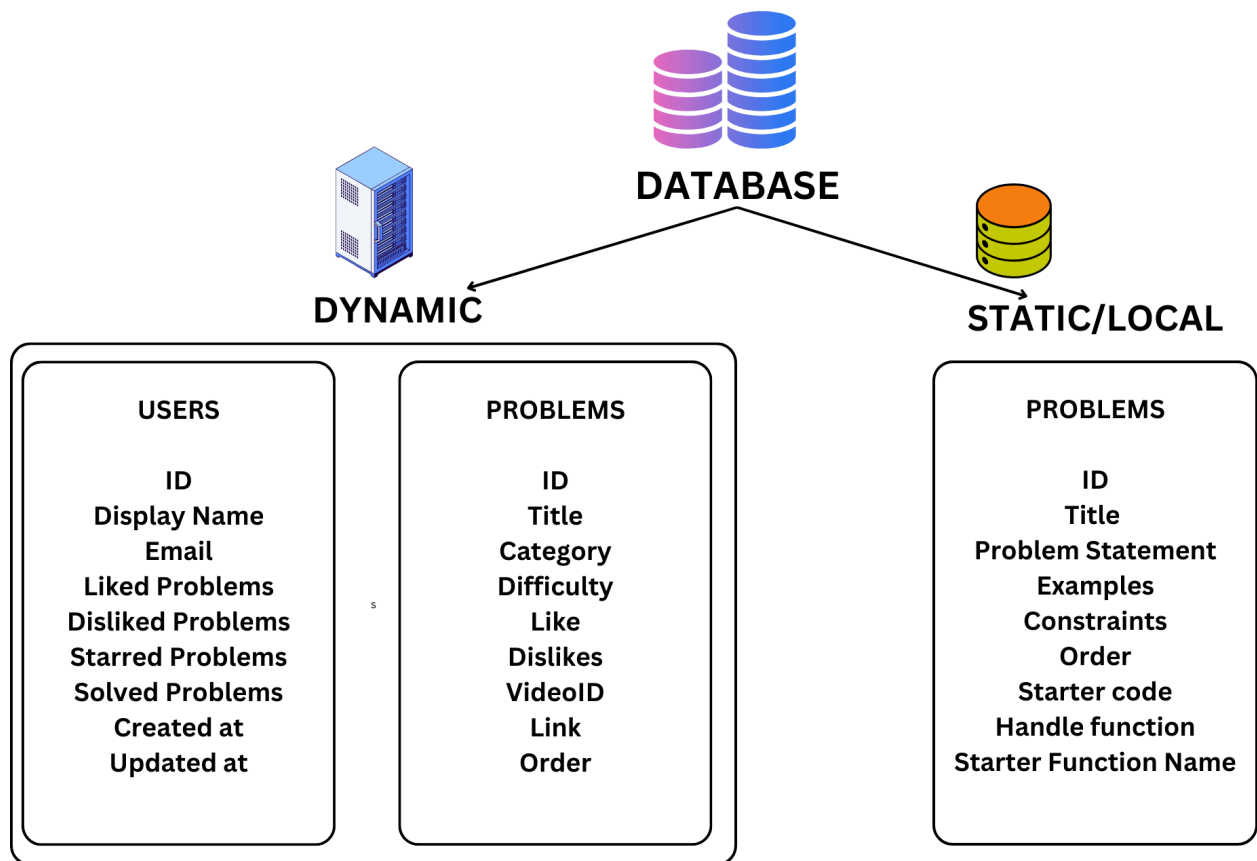
The code editor provides flexibility with JavaScript support, allowing users to write and test their solutions. Further customization is available through font size adjustments accessible from the settings menu. Additionally, users can toggle between fullscreen mode and exit at their convenience.

Interactivity is heightened with the ability to express appreciation or provide feedback through like, dislike, and submission options. However, to access these features, users are required to log in. Once logged in, if a problem has been previously solved, the platform intelligently fetches the code solution from local storage and checks its status.

A convenient timer feature is incorporated, offering users the option to open, close, or reset as needed. Navigation through different problems is simplified using arrows within the problem list.

Upon submission, the platform provides instant feedback — a congratulatory message for correct solutions and an error message for incorrect ones, fostering an engaging and rewarding coding journey for users.

Database Schemas:



```

const userData = {
  uid: newUser.user.uid,
  email: newUser.user.email,
  displayName: inputs.displayName,
  createdAt: Date.now(),
  updatedAt: Date.now(),
  likedProblems: [],
  dislikedProblems: [],
  solvedProblems: [],
  starredProblems: [],
};

export type Example = {
  id: number;
  inputText: string;
  outputText: string;
  explanation?: string;
  img?: string;
};
|
// local problem data
export type Problem = {
  id: string;
  title: string;
  problemStatement: string;
  examples: Example[];
  constraints: string;
  order: number;
  starterCode: string;
  handlerFunction: ((fn: any) => boolean) | string;
  starterFunctionName: string;
};

export type DBProblem = {
  id: string;
  title: string;
  category: string;
  difficulty: string;
  likes: number;
  dislikes: number;
  order: number;
  videoId?: string;
  link?: string;
};

export type Problem = {
  id: string;
  title: string;
  difficulty: string;
  category: string;
  order: number;
  videoId?: string;
};

export const problems: Problem[] = [
  {
    id: "two-sum",
    title: "Two Sum",
    difficulty: "Easy",
    category: "Array",
    order: 1,
    videoId: "8-k1C6ehKuw",
  },
  {
    id: "reverse-linked-list",
    title: "Reverse Linked List",
    difficulty: "Hard",
    category: "Linked List",
    order: 2,
    videoId: "",
  },
];

```

<div>+ Start collection</div> <div>problems</div> <div>users ></div>	<div>+ Add document</div> <div>r7sBq8dacoXhFtH4lIUfFd3EHEf1 ></div>	<div>+ Start collection</div> <div>+ Add field</div> <div>createdAt: 1687947929617</div> <div>dislikedProblems</div> <div>displayName: "Priyamvada2000"</div> <div>email: "priyadarshanipriyamvada@gmail.com"</div> <div>likedProblems</div> <div>0 "two-sum"</div> <div>solvedProblems</div> <div>0 "two-sum"</div> <div>1 "valid-parentheses"</div> <div>starredProblems</div> <div>0 "two-sum"</div> <div>uid: "r7sBq8dacoXhFtH4lIUfFd3EHEf1"</div> <div>updatedAt: 1687947929617</div>
<div>+ Start collection</div> <div>problems ></div> <div>users</div>	<div>+ Add document</div> <div>container-with-most-water</div> <div>jump-game</div> <div>reverse-linked-list</div> <div>search-a-2d-matrix</div> <div>two-sum ></div> <div>valid-parentheses</div>	<div>+ Start collection</div> <div>+ Add field</div> <div>category: "Array"</div> <div>difficulty: "Easy"</div> <div>dislikes: 0</div> <div>id: "two-sum"</div> <div>likes: 1</div> <div>link: ""</div> <div>order: 1</div> <div>title: "1.Two Sum"</div> <div>videoId: "8-k1C6ehKuw"</div>

Problem local data needs to be rendered on the problem page .

First, we fetch the local data created using SSG(Static Site Generation). The pages will be pre-generated on the server, it's super-fast, no loading state, is immediate render, already rendered while build. Client site fetching is for data stored in database. Both ways are mixed for better performance and maintenance.

To maintain Consistency transactions of Firebase were used:

Updating likes, liked problems, disliked problems, dislikes. Group multiple operations into a single transaction, either it is going to fail or succeed.

Hydration error:

What if the content being rendered on the server and client don't match? This leads to a hydration error. To solve this, we have a `useEffect` hook. When a React app hydrates, it assumes that the DOM structure will align.

During the initial client-side run, the React app creates a mental model of the expected DOM by mounting components. It then compares this with the existing DOM nodes on the page, attempting to seamlessly integrate them. Unlike the usual update process, it aims to align the two for correct handling of future updates.

By rendering different content based on whether we're in server-side rendering, we're essentially working around the system. We display one thing on the server but inform React to anticipate something else on the client side. Although React can sometimes manage this situation, it's a risky approach. While the hydration process is optimized for speed, it's not designed to identify and correct mismatches.

To address this, it's advised to introduce a state variable, `hasMounted`, initialized as `false`. While `false`, we avoid rendering the "real" content. Within the `useEffect` call, we trigger a re-render immediately, setting `hasMounted` to `true`. This ensures that the "real" content is rendered only after the component has mounted. This approach differs from the previous solution as `useEffect` only executes after the component has been mounted, aligning with React's expectations during the hydration process.

IV. Deployment

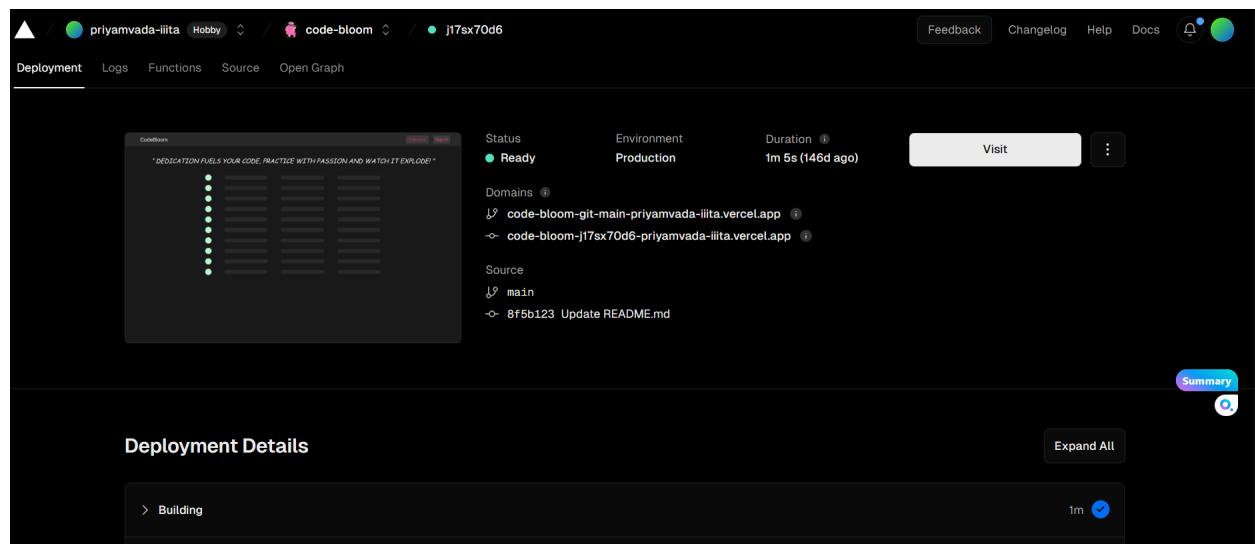
I chose Vercel for deployment in my project to ensure a seamless and efficient deployment process, as well as to leverage its powerful features for hosting and scaling web applications.

Easy Deployment Process: Vercel offers a straightforward and user-friendly deployment process. With its seamless integration with popular frameworks like Next.js, Vercel simplifies the deployment of web applications. Its intuitive interface and automated deployment workflows enable developers to quickly deploy updates and changes to their applications without unnecessary complexity or manual configurations.

Instantaneous Scaling: Vercel's infrastructure allows for effortless scalability. As the user base and demand for my coding platform grow, Vercel automatically scales the application to handle increased traffic. This scalability ensures that the platform remains accessible and performs optimally, even during peak usage periods.

Serverless Functions: Vercel provides serverless functions, allowing me to implement server-side logic for specific functionalities of my coding platform. This feature enables me to offload

server-side processing and execute code in a serverless environment, resulting in improved performance and reduced server management overhead.



Continuous Deployment and Preview Environments: Vercel offers seamless integration with Git repositories, enabling continuous deployment. With each push to the repository, Vercel automatically builds and deploys the latest version of the application. Additionally, Vercel provides preview environments, allowing me to create temporary instances of the application for testing and reviewing changes before deploying them to the production environment

Challenges/Achievements:

Why is my website preferable by user??

Crafted to address the prevailing challenge in an industry saturated with numerous coding websites, my solution targets the pivotal issue of search engine visibility. In a sea/jungle of coding platforms, the determining factor for visibility rests in the hands of the all-powerful Google search engine. To secure a prominent position in search results, a website must be tailored to be Search Engine Optimization (SEO) friendly, a practice diligently embraced by industry leaders such as Netflix and Instagram.

Conventionally:

The conventional approach to rendering web applications involves the client side, where HTML, CSS, and JS are downloaded by the browser. Subsequently, JS takes the reins to render the entire page, encapsulating all functionalities in a single bundle provided to the browser. In this process, JS handles tasks such as content painting, asset downloading, blog posting, and image population. However, since the entire content population relies on JS, it poses a challenge for search engine bots, especially the Google bot, as JS doesn't operate instantaneously. Consequently, the bot cannot read the content, leading to an inability to generate the previews

commonly seen on social media platforms. This lack of accessibility for bots renders the website non-SEO friendly, resulting in compromised search rankings. The delay in the First Contextual Paint further exacerbates the issue, adversely impacting user experience.

Focus was on:

- Backend/Storage division issue
- Fighting against slow and inappropriate rendering
- Balancing the trade-offs
- Visibility in google search

Unique/Advanced solution offered by my Websites:

- SEO friendly: In response to this SEO dilemma, we employ innovative page-building strategies. Our approach ensures that the data sent to the browser contains pre-rendered content, eliminating the reliance on JavaScript for initial content display. By adopting this method, we prioritize SEO friendliness, allowing search engine bots to index the website effectively and enhancing search rankings. This proactive measure not only mitigates the challenges posed by delayed rendering but also optimizes user experience, creating a website that stands out in the competitive landscape.

- Faster: Rendering Issues covered(SSG, SSR):

Static Site Generation (SSG): Transforms the complete website into a static form to enhance delivery speed, ideal for data that changes infrequently.

Server Side Rendering (SSR): Generates pages on the server, delivering dynamic content tailored for data that changes frequently.

The Challenge was to balance it and decide what to use where. First, we fetch the local data created using SSG(Static Site Generation). The pages will be pre-generated on the server, it's super-fast, has no loading state, is immediate render, already rendered while built. Client site fetching is for data stored in the database. Both ways are mixed for better performance and maintenance.

- Managing Local and Dynamic database:
Hybrid Approach: Incorporates a revalidate key in getStaticProps, allowing periodic content refresh while retaining the advantages of a static site.
- Transaction & Consistency:
To maintain Consistency transactions of Firebase were used:

Updating likes, liked problems, disliked problems, dislikes. Group multiple operations into a single transaction, either it is going to fail or succeed.

- Trade-offs successfully planned:
 - User Experience vs. Load Time
 - Security vs. Convenience(firebase)

I employ a mixed rendering approach for optimal results: static data is stored in a local server database and rendered using Static Site Generation (SSG) for faster page generation. Dynamic data, subject to frequent changes, is handled through Server-Side Rendering (SSR) using databases or Firestore, balancing maintainability and performance.

- Solved hydration error:(Discussed in detail under architecture)
In the initial client-side run, the React app creates a mental model of the expected DOM, comparing it with existing nodes for integration. When rendering different content based on server-side rendering, it's a workaround. To avoid issues, introducing a state variable, `hasMounted`, initialized as `false`, helps. While `false`, it avoids rendering the "real" content. When `useEffect` triggers a re-render and sets `hasMounted` to `true`, the "real" content renders only after the component has mounted, aligning with React's expectations during hydration.
- Code is readable and anyone can make further additions

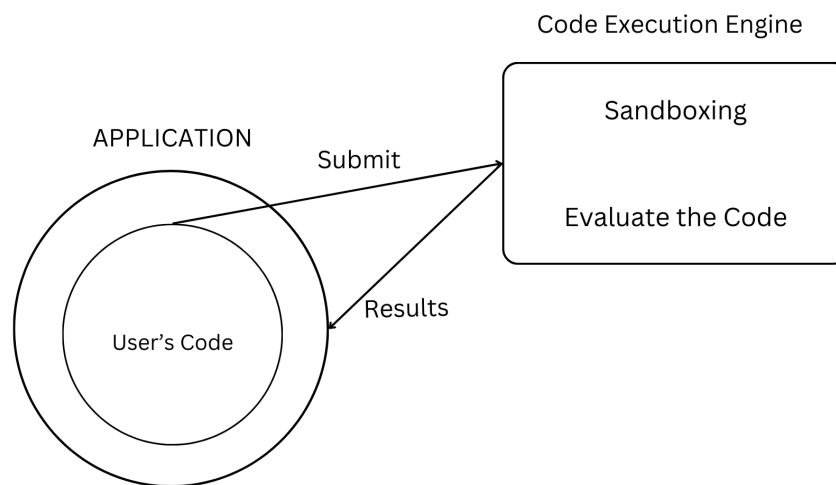
Conclusion

In the development journey of CodeBloom, strategic decisions were made to enhance performance and maintain data consistency. The utilization of Firebase transactions showcased a commitment to data integrity by grouping multiple operations into single transactions when used for like and dislike of problems. Thoughtful trade-offs were made, balancing dynamic and static storage options, while integrating client-side rendering and Static Site Generation (SSG) for optimal user experience.

The codebase has great readability and reusability, fostering a development environment that prioritizes maintainability. Incorporating various React packages contributed to crafting a user-friendly UI, where the challenges of CSS were gracefully addressed with the aid of Tailwind CSS, streamlining styling complexities.

Looking into the future, the CodeBloom project anticipates significant growth opportunities. The roadmap involves diversifying the problem set to enhance the platform's content, introducing new sections to expand its utility, and continuously improving the user interface. Additionally, there are plans to enhance inclusivity by adding language support for C++ and Python, creating a more comprehensive coding environment. Advanced testing methods will also be implemented to incorporate a greater number of test cases

Future Scope/ Guide to Implement for a Start-up



Sandboxing promises safe execution and security to the application and its server though its implementation is hard. When building a real application for a company, one would like to have a code execution engine, it is quite complicated and requires expertise and experience with tools like Docker. Our application is going to be safe, users can rely on our application without any doubts. What it will do is once the user submits the code, it is received and sent to the code execution engine for running so viruses or other malware cannot attack the server.

Looking ahead, the CodeBloom project holds promising avenues for expansion. The roadmap includes the addition of a diverse array of problems, further enriching the platform's content. The plan extends to incorporating new sections, broadening the scope and utility for users. The user interface is poised for continuous improvement, with plans to introduce language support for various other languages like C++ and Python, providing a more inclusive coding environment, and advanced testing methods to include more test cases.

Acknowledgment

The progress and this report would not have been possible without the support, guidance, and contributions of some individuals and institutions. I extend my heartfelt thanks to my Summer Intern professor and supervisor, Dr. Navjyoti Mazumdar, for his unwavering support and guidance throughout this project.

I am also grateful to the Indian Institute of Information Technology, Allahabad, for providing me with the opportunity, essential resources, and facilities. I appreciate the collective efforts of all who contributed to this research, directly or indirectly, as they played a crucial role in our progress.

Supplementary Material

Code Link: [Priyamvada-iiita/CodeBloom \(github.com\)](https://github.com/Priyamvada-iiita/CodeBloom)

References

1. [CodeBloom \(code-bloom.vercel.app\)](https://code-bloom.vercel.app)
2. [Priyamvada-iiita/CodeBloom \(github.com\)](https://github.com/Priyamvada-iiita/CodeBloom)
3. [Firebase console \(google.com\)](https://console.firebase.google.com)
4. [react-toastify - npm \(npmjs.com\)](https://www.npmjs.com/package/react-toastify)
5. [Getting Started | Recoil \(recoiljs.org\)](https://recoiljs.org)
6. [react-confetti-explosion - npm \(npmjs.com\)](https://www.npmjs.com/package/react-confetti-explosion)
7. [code-bloom – Overview - Vercel](https://vercel.com/code-bloom)
8. [React Icons \(react-icons.github.io\)](https://react-icons.github.io)
9. [The Perils of Hydration: Understanding how Gatsby/Next manage server-side rendering and hydration \(joshwcomeau.com\)](https://joshwcomeau.com)