ResUI: Crafting an Intuitive Interface for ResilientDB

Gopal Nambiar
gnambiar@ucdavis.edu

Dieu Anh Le
ndale@ucdavis.edu

Rishika Garg
risgarg@ucdavis.edu

Sasha Pimento
spimento@ucdavis.edu

Anubhav Mishra
imishra@ucdavis.edu

Fall 2023

This project introduces ResUI, a comprehensive user interface for ResilientDB, a groundbreaking permissioned blockchain fabric. Motivated by ResilientDB's innovative consensus protocols and versatile interfaces, including the Key-Value store and Python SDK, ResUI aims to enhance the marketability of ResilientDB by providing an intuitive and user-friendly interface. The user interface seeks to avoid information clutter by breaking it into smaller, focused pages containing dedicated sections. The ReactJS framework is employed for cohesive and visually appealing designs. back-end functionality ensures secure and efficient user authentication processes for user management. ResUI also offers a comprehensive solution for managing ResilientDB instances, covering installation, configuration, and usage. The integration of Grafana and Prometheus enhances the monitoring capabilities, providing real-time insights into ResilientDB metrics, thus positioning ResUI as a valuable contribution to the ResilientDB ecosystem.

1 Introduction

ResUI draws inspiration from ResilientDB, an innovative blockchain technology originating from ExpoLab at UC Davis. It represents a significant advancement in the field of High-Throughput Permissioned Blockchain Fabrics.

While conventional blockchain protocols rely on traditional consensus protocols like Practical Byzantine Fault Tolerance (PBFT) and HotStuff, ResilientDB uses novel consensus protocols such as GeoBFT, Proof Of Execution, Resilient Concurrent Consensus (RCC), and Spotless. Additionally, it provides a wide range of interfaces like the Key-Value store and Python SDK, enabling users to use ResilientDB for various functionalities, showcasing its versatility. Furthermore, ResilientDB not only gives users the ability to perform transactions in a decentralized and democratized way but also supports Grafana for visualizing monitoring data. Its recent acceptance into the Apache Incubator opens doors for developers to contribute and potentially elevate projects to top-level Apache Software Foundation status.

With these accomplishments in mind, the next focus now pivots to establishing ResilientDB as a marketable product. Recognizing the importance of user interface in software products, this project introduces ResUI — a full-stack web application that provides the missing UX layer for ResilientDB. It provides a user-friendly interface crafted to streamline navigation for newcomers while systematically presenting ResilientDB's features to minimize on-screen clutter. Moreover, ResUI offers Sign Up and Login functionalities along with an Instances page, allowing authenticated users to view their running ResilientDB and SDK instances, complemented by a monitoring dashboard to analyze associated metrics.

In summary, the contributions of ResUI are as follows:

1. **User Interface Development**: Develop an intuitive and visually appealing user interface to enable users to interact seamlessly with ResilientDB.

2. **User Management**: Implement user registration and login functionalities with secure storage in a back-end database

3. **Monitoring Dashboard**: Create a monitoring dashboard that offers

real-time insights into the status and performance of ResilientDB instances.

4. **Containerization Support**: Provide support for containerization technology like Docker to enhance the scalability and portability of ResilientDB.

2 Technologies

ResUI utilizes a broad-range of cutting-edge front-end and back-end technologies. On the front-end, modern frameworks like React.js are used besides traditional HTML and CSS to deliver a sleek black-and-teal design system extending across desktop and mobile. Complementing this, the back-end architecture is powered by ResilientDB for data storage and management, Node.js and Python for functionalities, Prometheus and Grafana for monitoring graphic, and Docker for containerization, portability, and server hosting.

2.1 Front-End

React.js: React stands as the cornerstone of ResUI front-end development, offering a powerful and efficient approach in crafting a dynamic and engaging user experience on the front-end. Its flexibility and robust features contribute to the efficiency of our development process, delivering a responsive and feature-rich interface for the users.

React offers a component-based architecture, employing a modular structure with components that encapsulate functionality and rendering logic. This modularity significantly enhances code organization, maintainability, and reusability. In addition, React's built-in modules such as *styled-components*, *react-scroll*, and *react-router-dom* significantly streamline the process of tailoring websites to specific needs while maintaining a level of flexibility that allows the developer to customize the website.

HTML and CSS: HTML and CSS compose the backbone of the front-end, collaboratively shaping an intuitive and visually compelling user interface. HTML establishes the structural foundation for web pages, dictating content organization and presentation. Through semantic markup, it communicates not only visual information but also the meaning and purpose of different content elements. CSS takes the lead in visual styling, enabling the definition of colors, fonts, spacing, and layout. It guarantees a uniform design across the application by segregating style from structure. Additionally, through the use of *@media screen* CSS plays a pivotal role in responsive design, seamlessly adapting the interface to various screen sizes and devices.

2.2 Back-End

ResilientDB: ResilientDB stands as a robust and modular open-source system designed for deploying and operating permissioned blockchain applications with a focus on high-throughput consensus. In our project, ResilientDB serves a dual role as both the back-end technology for storing user data on our website and as the distributed database underpinning our blockchain fabric. This innovative fabric, constructed from the ground up, embodies state-of-the-art software engineering principles, showcasing a lean design that simplifies application deployment. ResilientDB empowers developers by providing flexibility in implementing and testing Byzantine fault-tolerant (BFT) protocols, extending its utility beyond traditional cryptocurrencies to Blockchain-as-a-Service applications.

ResilientDB's multi-threaded deep pipelines contribute to its exceptional throughput, allowing seamless consensus among replicas. Developers benefit from the configurability to adjust thread count and pipeline stages to match specific application requirements. The platform further supports testing through YCSB transactions and Smart Contracts, providing a versatile environment for application development. Furthermore, the inclusion of a Graphical User Interface (GUI) elevates the user experience, enabling developers to compile, deploy, and run the platform effortlessly. This GUI serves as a centralized hub for monitoring individual replica performance, offering a comprehensive and accessible toolset for analyzing results with just a click. In essence, ResilientDB plays a pivotal role in our project, embodying efficiency, flexibility, and user-friendly functionality in the storage of user data and the management of our distributed database.

Node.js and Python: Node.js and Python contribute to distinct aspects of the system architecture in ResUI. Node.js drives the user authentication functionalities, specifically managing the sign-up and log-in processes. Renowned for its non-blocking, event-driven architecture, Node.js ensures efficient and responsive handling of user interactions, well-suited for concurrent operations vital in user authentication. On the other hand, Python plays a pivotal role in the command-line interface of the project. Leveraging its simplicity and versatility, Python enables seamless interaction with the system through the command line. This proves particularly beneficial for users preferring a command-line environment for specific operations, providing flexibility and ease of use.

Prometheus and Grafana: Prometheus excels in efficient metric collection and storage, scraping real-time metrics from diverse sources, including ResilientDB instances. This contributes to real-time monitoring, providing immediate insights into system health and performance. Grafana, known for transforming raw data into actionable insights, promotes a proactive approach to system monitoring. Its intuitive interface facilitates the creation of dashboards, effectively tracking critical performance aspects. With real-time visualization, Grafana provides dynamic, interactive panels for a comprehensive view of database metrics. Integrated into ResUI, it centralizes monitoring insights, enhancing user experience and accessibility within the primary interface. Collaboratively, Grafana and Prometheus provide real-time metric visualization and proactive measures through alerting and notifications.

Docker: Docker supports deployment and scalability by encapsulating applications and dependencies into portable containers, ensuring consistency across environments. Its key contributions include simplifying deployment processes, offering cross-platform compatibility, and enhancing maintainability through isolated containerization.

3 Architecture

The project architecture comprises several interconnected components orchestrated to provide a seamless user experience and manage ResilientDB instances.

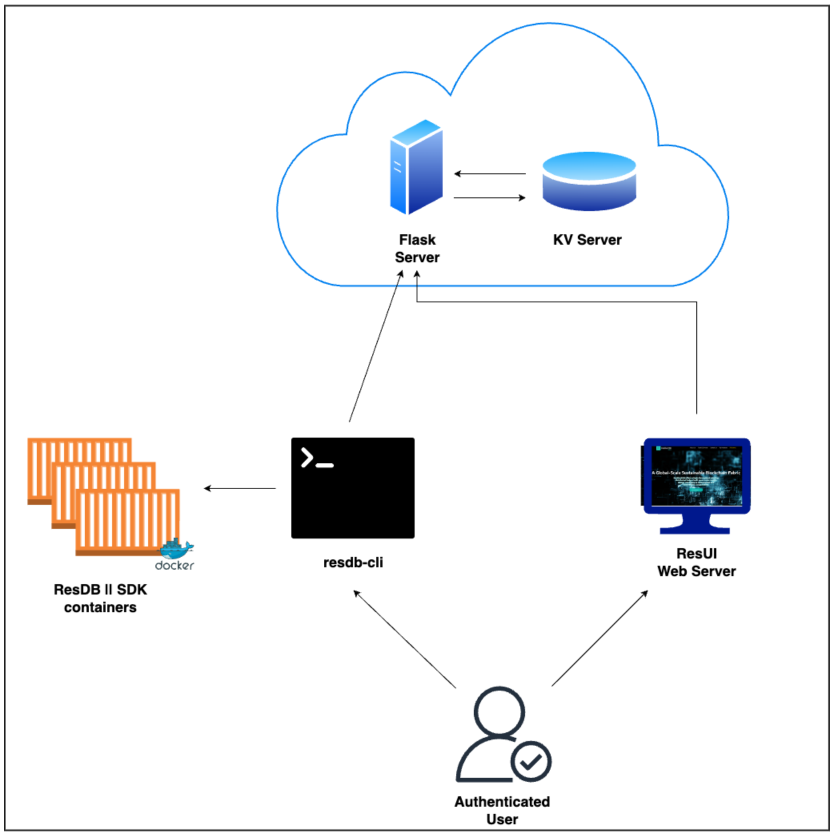


Figure 1. Diagrammatic Representation of Architecture of ResUI

1. **KV Server (ResilientDB)**: Hosted on a Linux (Ubuntu 20.04) VM on Google Cloud Platform (GCP), which acts as the primary key-value store for user data and ResilientDB instance data.

2. **Flask API Server**: Hosted on GCP, which serves as an intermediary between user requests and the KV Server. In addition, it accepts requests from authenticated users through ResUI website and ResilientDB CLI.

3. **User Interfaces**:

    - The ResUI website makes API calls to the Flask server to retrieve and display data from ResilientDB and performs POST processing based on the responses, enhancing the displayed data on the webpage.

- ResilientDB CLI queries the Flask server for user-specific instance information as well as calls the Flask API to update the count of ResilientDB and SDK instances when new instances are created using the CLI.

4. **Authentication Mechanism**: Users are authenticated through the ResUI website and the same credentials are used for authentication in ResilientDB CLI. Moreover, user credentials are stored in ResilientDB, allowing for seamless authentication checks during CLI usage.

5. **User Privileges**: Authenticated users have access to both ResUI and ResilientDB CLI.

6. **ResilientDB CLI Functionality**: The command line interface allows the creation of ResilientDB and SDK containers and calls Flask API to update instance counts and queries for user-specific instance information.

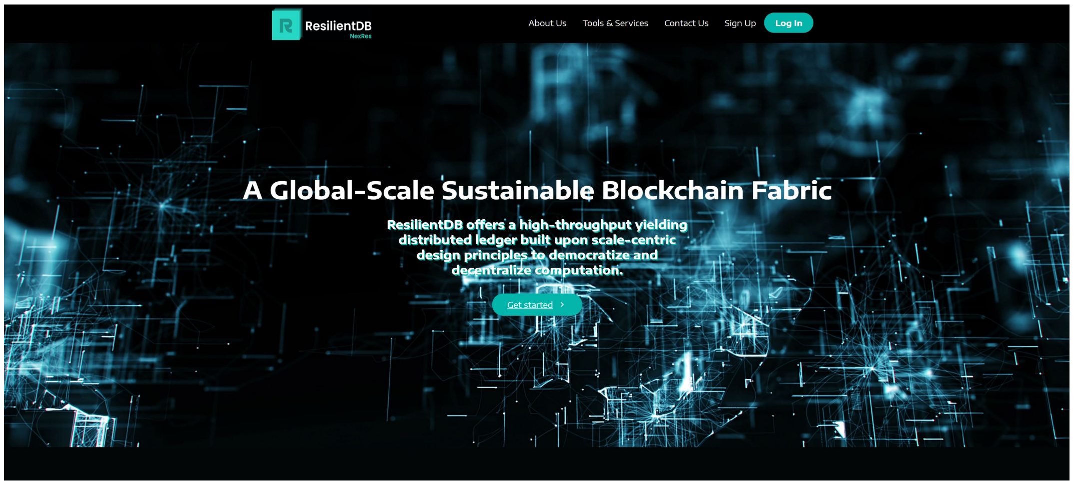
This architecture ensures a cohesive flow of data between user interfaces, the Flask API server, and the KV Server, creating a robust and user-friendly environment for managing ResilientDB instances.

4 UI Design

The user interface aspect of this project comprises multiple pages, each composed of various components. Adopting a modern black and teal design theme, the visualization strategically avoids information clustering by segmenting content into smaller, digestible pages, as demonstrated below.

4.1 Landing Page

The Home Page comprises several components, organized as follows:

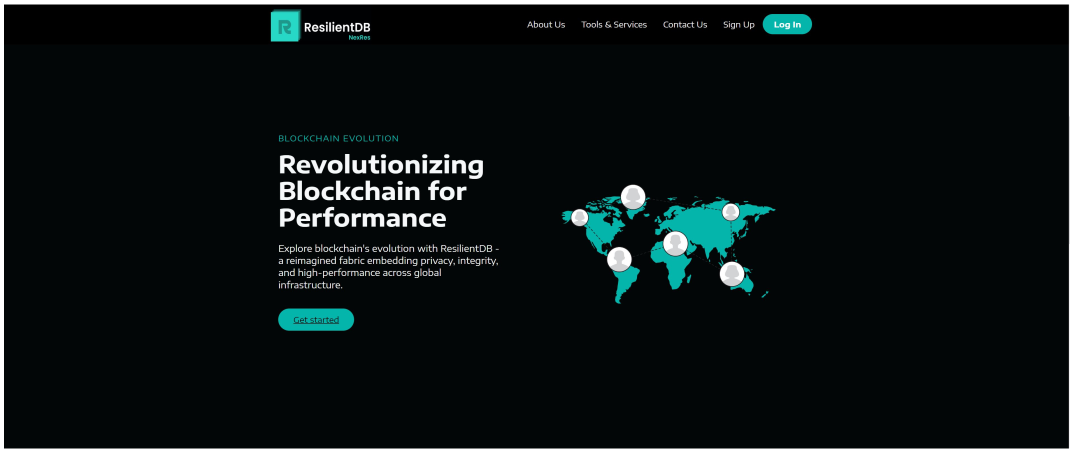


A Global-Scale Sustainable Blockchain Fabric

ResilientDB offers a high-throughput yielding distributed ledger built upon scale-centric design principles to democratize and decentralize computation.

Figure 2. Screenshot of Home Page

Navbar: The Navbar prototype consolidates elements from the existing ResilientDB landing page, providing users with a unified and intuitive navigation experience. It features an About Us Tab, expanding on hover to display sections like Mission & Vision, Roadmap, and Publications. Additionally, the Tools & Services Tab grants access to the ResilientDB GitHub Repository and tools such as Explorer, Monitoring, and Prometheus.

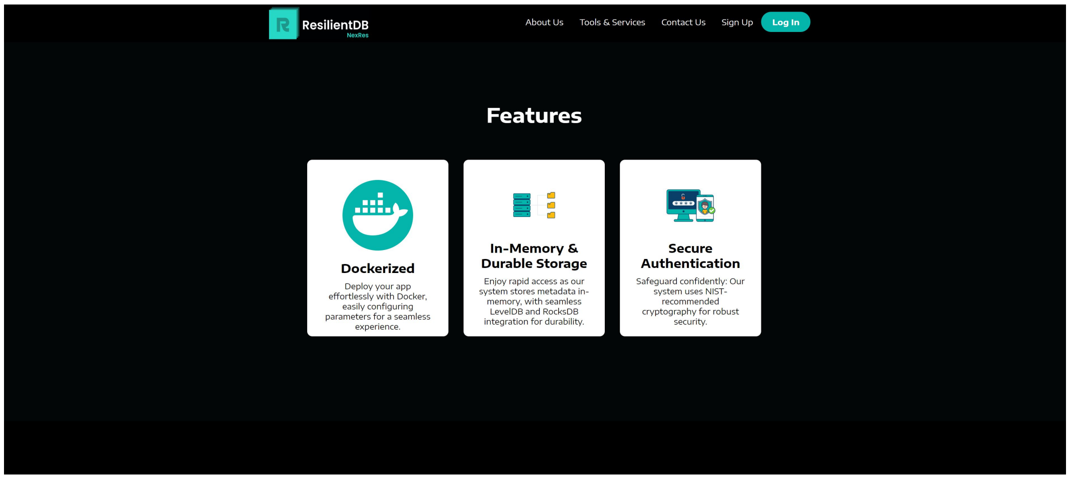


Revolutionizing Blockchain for Performance

Explore blockchain's evolution with ResilientDB - a reimagined fabric embedding privacy, integrity, and high-performance across global infrastructure.

Figure 3. Screenshot displaying the Get Started section

Landing Page Background: This section features a background video animation accompanied by a 'Get Started' button. Clicking this button directs users to the Sign Up Page, enabling account creation for ResilientDB.



Features

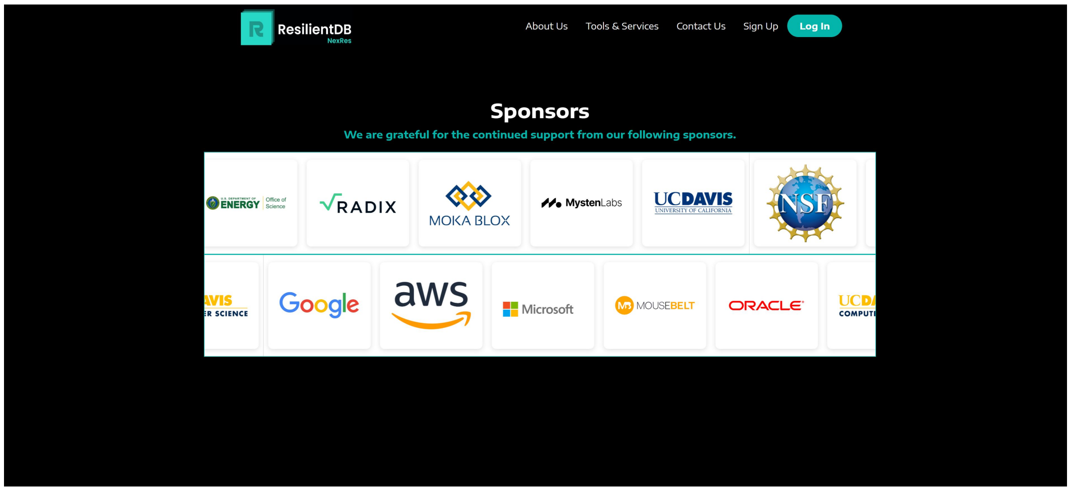
Dockerized
Deploy your app effortlessly with Docker, easily configuring parameters for a seamless experience.

In-Memory & Durable Storage
Enjoy rapid access as our system stores metadata in-memory, with seamless LevelDB and RocksDB integration for durability.

Secure Authentication
Safeguard confidently: Our system uses NIST-recommended cryptography for robust security.

Figure 4. Screenshot of Information and Services component

Information and Services Components: This segment offers concise information about ResilientDB, showcasing various services available. Engaging hover effects encourages user interaction on the website.



Sponsors

We are grateful for the continued support from our following sponsors.

Figure 5. Screenshot of Sponsors carousel component

Sponsors Component: Implemented as a carousel, the Sponsors section uses the Marquee effect in React, presenting two groups of sponsors for enhanced visual appeal.

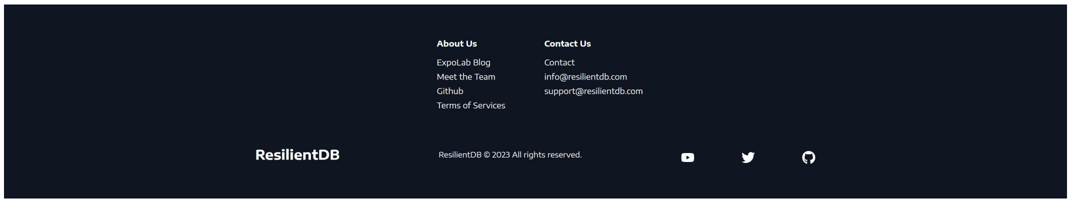


Figure 6. Screenshot of Footer component

Footer: ResUI features a footer with a variety of links for enhanced navigation. One link directs users to the About Us section, providing insights into the team. Additional links lead to the ResilientDB Blog for articles related to ResilientDB and the project's GitHub repository. The Contact Us section facilitates easy communication with ResilientDB members via clickable email IDs, opening a mail pop-up in the user's native mail application. The Social Media section presents ResilientDB's presence on platforms like YouTube, Twitter, and GitHub, keeping users updated on the project's latest technological innovations.

4.2 About Us Page

The About Us page opens with three embedded video elements introducing ResilientDB's vision, mission, and Next Generation project. These videos facilitate seamless user interaction, encouraging users to engage with the content by clicking on the videos to learn more about ResilientDB as it eliminates the necessity to navigate to external pages. To streamline information and prevent overwhelming content, the About Us section is divided into three smaller pages: Meet The Team, Roadmap, and Publications.

4.2.1 Meet The Team Page

The 'Meet The Team' component stands as the initial section of the About Us page, showcasing the individuals instrumental in shaping innovative concepts at ResilientDB. Organized in a hierarchical structure, it starts with categories like 'Our Advisors,' 'Our Visionaries,' 'Our NexRes,' and 'Our Alumni,' highlighting the diverse contributors steering ResilientDB's success.

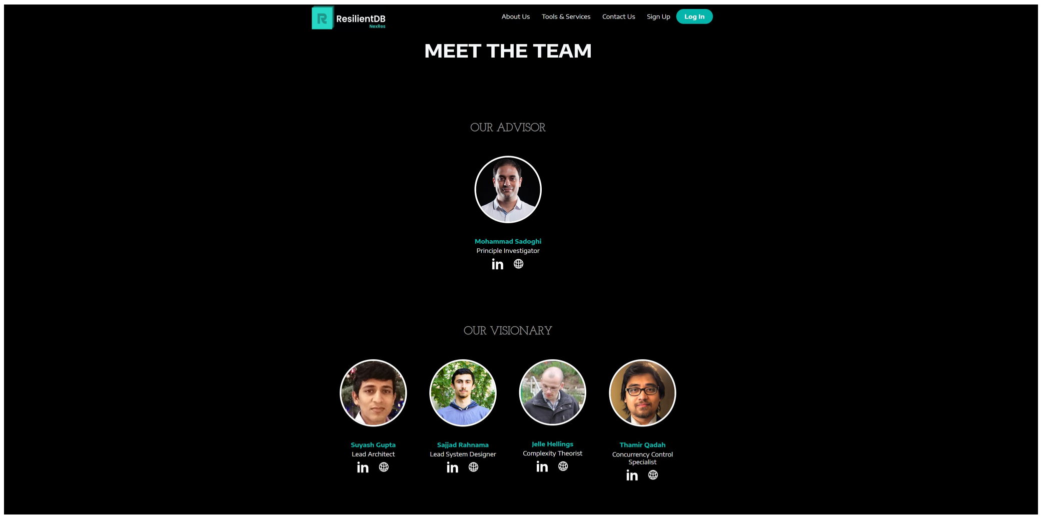


Figure 7. Screenshot of Meet the Team Page showing the Team Leaders

Each team member is presented through a card format displaying their picture, name, position, and an embedded LinkedIn link; the Advisor and the Visionaries are additionally presented with a web icon that redirects to their webpage. This construction is facilitated by the CreateCard function mapping entries in *TeamInfo.js*, ensuring the automated update process new team additions.

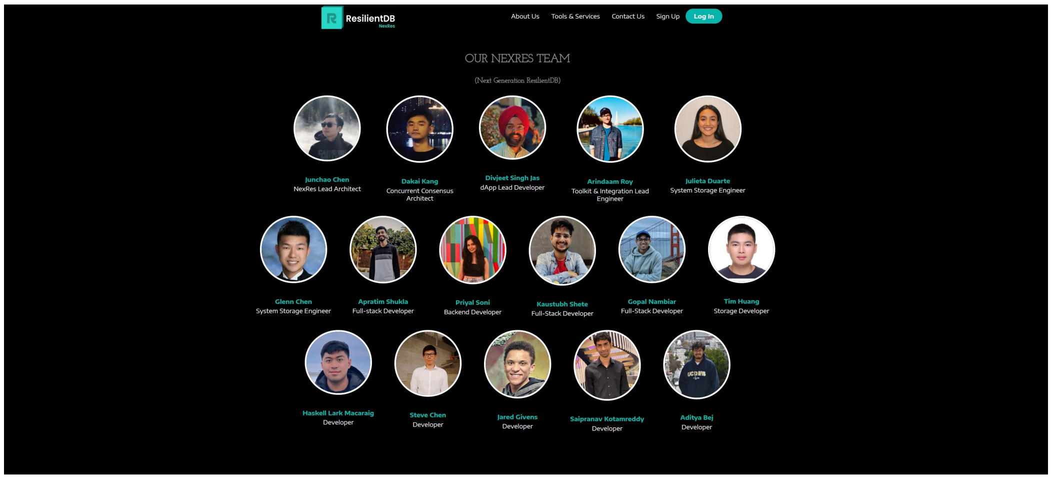


Figure 8. Screenshot of Meet the Team Page showing the Team Members

4.2.2 Roadmap Page

Roadmap stands as another major component of the About Us page, providing users with insight into ResilientDB's journey until the present day and its future plans. To implement the Roadmap, the vertical-timeline-element in React was utilized, showcasing each milestone alongside engaging animations that enhance the storytelling process. Additionally, buttons have been included for specific milestones; for instance, clicking on a button redirects users to associated papers or announcements in a new tab. Similar to the 'Meet The Team' component, each Roadmap element is clearly defined, enabling developers to easily add more milestones by updating the database *timelineElement.js* with the appropriate formatting.

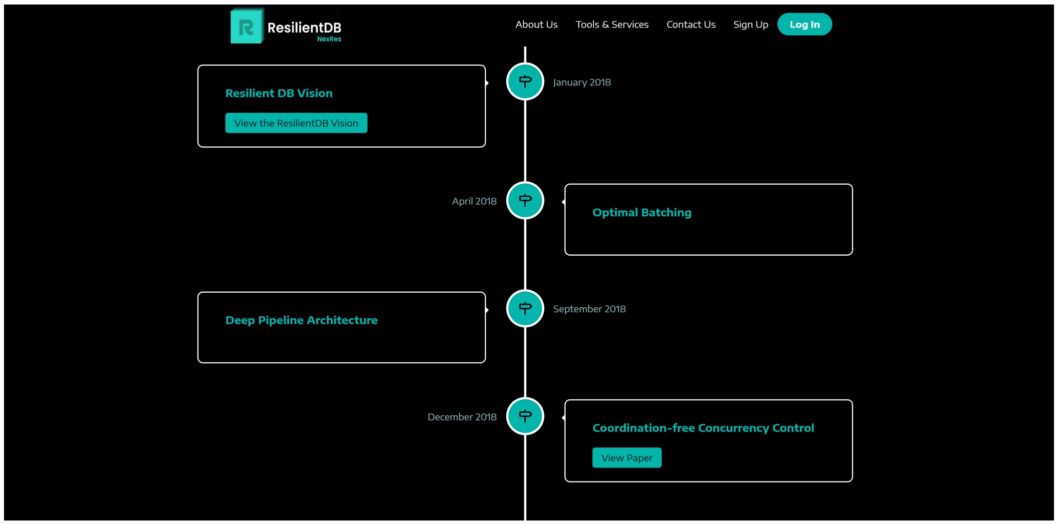


Figure 9. Screenshot of Roadmap page

4.2.3 Publications Page

Regarding the third component in the 'About Us' section—the publications section—inspiration was drawn from the original ResilientDB website to create a dedicated area for showcasing publication works. This section presents a substantial collection of research works published by various members of ExpoLab contributing to ResilientDB since its inception. Furthermore, highlighted within are award-winning research publications, specifying the conferences where awards were received. For instance, the 'Best Paper Award' was achieved for the paper titled "SGX Accelerated Consensus" at the EuroSys conference in 2023. This feature accentuates the most notable research works for visitors seeking to explore top-tier publications. Each research paper is presented with a specific title and an associated hyperlink, providing direct access to the respective research paper.

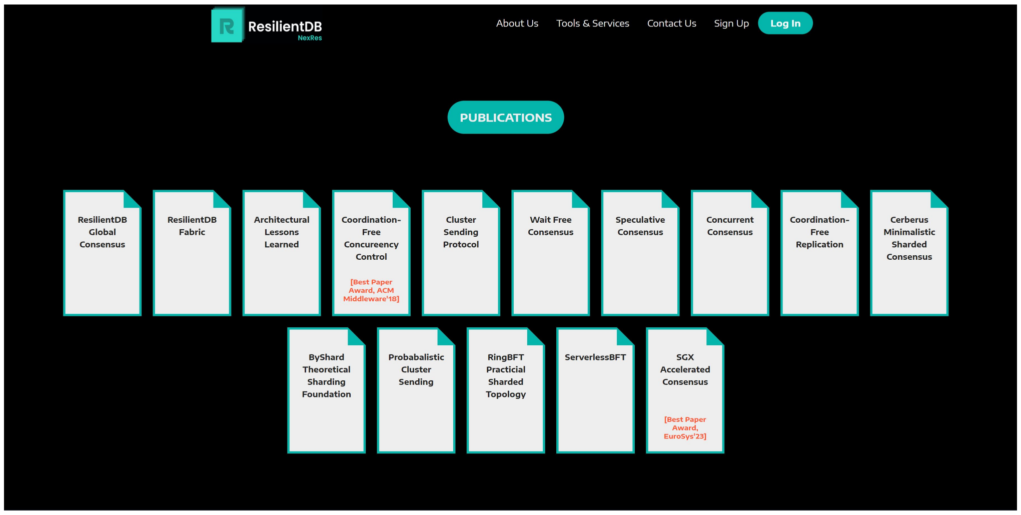

ResilientDB
NexRes
About Us
Tools & Services
Contact Us
Sign Up
Log In
PUBLICATIONS
ResilientDB Global Consensus
ResilientDB Fabric
Architectural Lessons Learned
Coordination-Free Concureency Control
[Best Paper Award, ACM Middleware'18]
Cluster Sending Protocol
Wait Free Consensus
Speculative Consensus
Concurrent Consensus
Coordination-Free Replication
Cerberus Minimalistic Sharded Consensus
ByShard Theoretical Sharding Foundation
Probabalistic Cluster Sending
RingBFT Practicial Sharded Topology
ServerlessBFT
SGX Accelerated Consensus
[Best Paper Award, EuroSys'23]

Figure 10. Screenshot of Publications Page

4.3 Contact Us Page

The third component of Navbar is the Contact Us Page, which serves as a platform for technical inquiries, clarifications, feature requests, and reporting website flaws. This page displays details of ResUI project developers, providing their email addresses and LinkedIn profiles for direct contact. Similar to the ResilientDB member card in 'Meet The Team', each ResUI team member card is generated by mapping data entries in *contact.js*. Furthermore, these cards feature a hover effect to enhance user interaction. Besides that, a button is included, linking to the ResUI GitHub repository to provide direct access to the project's code base.

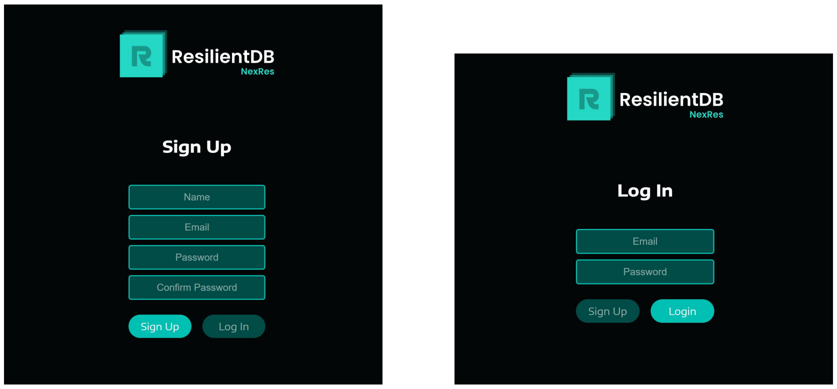


Figure 11. Screenshot of Sign Up and Log In pages

4.4 Sign-Up and Login Pages

The Sign-Up and Login pages follow a minimalist design. With a focus on improving user experience, we have integrated two buttons for seamless switching between the Log-In and Sign-Up pages. Additionally, the ResilientDB logo positioned at the top provides a convenient way to navigate back to the home page. Upon logging in, the Log-In button will be replaced with a welcoming message and display a "My Instance" page that is discussed further in the next section.

4.4.1 Server.js

Overview: The *server.js* file serves as the back-end for the user authentication processes, handling user sign-up and login functionalities. It is implemented using Express.js, a web application framework for Node.js. The server interacts with a Flask server through API requests to manage user registration and authentication.

Setup: The server is configured to run on port 5500, and it employs Cross-Origin Resource Sharing (CORS) middleware to handle cross-origin requests. Additionally, a base URL for the Flask server is defined as flaskBaseUrl, allowing communication between the Node.js server and the Flask server.

User Signup Endpoint (/api/signup): The server exposes a POST endpoint at /api/signup to handle user registration. Upon receiving a sign-up request, it extracts the user's name, email, and password from the request body. It then validates the provided information and sends a corresponding API request to the Flask server's setUser endpoint for user registration. If the registration is successful, the server responds with a success message; otherwise, it returns an error message.

User Login Endpoint (/api/login): For user login, the server provides a POST endpoint at /api/login. Similar to the sign-up process, it extracts the user's email and password from the request body and sends an API request to the Flask server's getUser endpoint for authentication. Upon successful authentication, the server responds with a success message and a token (assuming token-based authentication). In case of any issues during the login process, the server returns an error message.

Additional Endpoint (/api/instances): An additional GET endpoint at /api/instances is implemented to retrieve instances data for a specific user. The server extracts the user's email from the query parameters, makes an API request to the Flask server's getInstances endpoint, and responds with the fetched data. If the retrieval is successful, the server responds with the data; otherwise, it returns an error message.

Error Handling: The server incorporates error handling mechanisms to address potential issues during sign-up, login, or instance data retrieval. In case of errors, appropriate error messages are sent to the client.

4.4.2 Signup/index.js

Overview: The *Signup/index.js* file contains the front-end logic for user registration. It features a signup form that collects user details, performs client-side validation, and sends a request to the back-end server for registration.

Signup Form: The signup form includes fields for the user's name, email, password, and password confirmation. The form ensures that the password and confirmation match before sending a signup request to the back-end server. Upon form submission, a fetch request is made to the /api/signup endpoint on the server. The server's response is processed, and appropriate messages are displayed to the user.

4.4.3 Login/index.js

Overview: The *Login/index.js* file handles user login on the front-end. It provides a login form with fields for email and password, sending a request to the back-end for authentication.

Login Form: The login form includes fields for the user's email and password. Upon form submission, a fetch request is made to the /api/login endpoint on the server. If the login is successful, a token is stored in the local storage, and the user is redirected to a protected route or the home page.

Error Handling: Error handling is implemented to manage potential issues during the login process. If an error occurs, relevant messages are logged to the console for further investigation.

4.5 My Instances Page

After logging in, users can view and manage their running instances through 'My Instance' page, which can be broken down into multiple components.

4.5.1 Command Line Interface

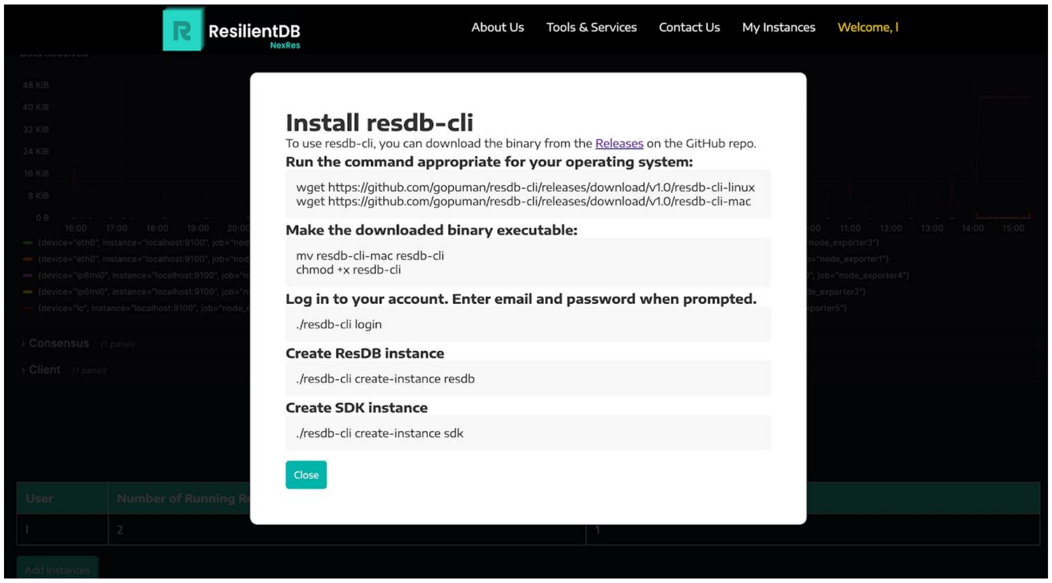
Overview: The resdb-cli project is a powerful command-line interface designed to facilitate the management of ResDB instances and Python SDK instances. By offering functionalities to create, delete, view, and manage instances, resdb-cli simplifies these operations, providing a seamless and efficient experience for users.

Installation: To utilize resdb-cli, users can easily download the binary from the Releases page on GitHub. The installation process is straightforward and can be found via the Add Instance button at the bottom left of My Instance page once logged in:

1. Visit the Releases page

2. Download the latest release suitable for your operating system (for example: *resdb-cli-linux* for Linux)

3. Make the downloaded binary executable using the following command: *chmod +x resdb-cli*



Install resdb-cli

To use resdb-cli, you can download the binary from the Releases on the GitHub repo.

Run the command appropriate for your operating system:

```
wget https://github.com/gopuman/resdb-cli/releases/download/v1.0/resdb-cli-linux
wget https://github.com/gopuman/resdb-cli/releases/download/v1.0/resdb-cli-mac
```

Make the downloaded binary executable:

```
mv resdb-cli-mac resdb-cli
chmod +x resdb-cli
```

Log in to your account. Enter email and password when prompted.

```
./resdb-cli login
```

Create ResDB instance

```
./resdb-cli create-instance resdb
```

Create SDK instance

```
./resdb-cli create-instance sdk
```

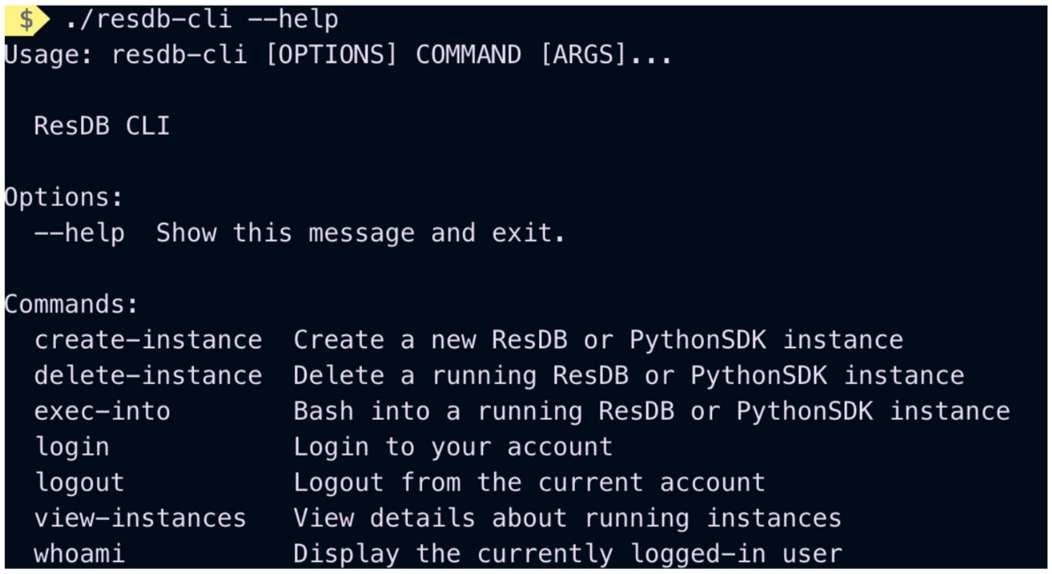
Figure 12. Instruction on how to install resdb-cli

Configuration: resdb-cli leverages a configuration file (*config.ini*) to store essential settings like the MongoDB URI. Users can easily configure the CLI by following these steps:

1. Create a configuration file named *config.ini*.

2. Replace the flask_base_url value with the appropriate endpoint connection string. An example configuration is provided below:

   - **[Server]**: flask_base_url = xyz:8080
   - **[User]**: current_user = bob@gmail.com



```
$ ./resdb-cli --help
Usage: resdb-cli [OPTIONS] COMMAND [ARGS]...

  ResDB CLI

Options:
  --help  Show this message and exit.

Commands:
  create-instance  Create a new ResDB or PythonSDK instance
  delete-instance  Delete a running ResDB or PythonSDK instance
  exec-into        Bash into a running ResDB or PythonSDK instance
  login            Login to your account
  logout           Logout from the current account
  view-instances   View details about running instances
  whoami           Display the currently logged-in user
```

Figure 13. Command options for resdb-cli

Usage: After installation and configuration, users can seamlessly perform various actions related to ResDB and Python SDK instances using resdb-cli. The CLI is executed with the following command:
./resdb-cli

Commands

1. *./resdb-cli login* : This command logs in to the specified user account, enter email and password when prompted.

2. *./resdb-cli logout*: This command logs out from the current user account.

3. *./resdb-cli view_instances*: This command displays details about running instances of the users.

4. *./resdb-cli delete_instance instance_id* : This command deletes a running ResDB or Python SDK instance.

5. *./resdb-cli whoami*: This command displays the currently logged-in user.

6. *./resdb-cli -help* or *./resdb-cli command –help*: This command provides more detailed information about each command.

4.5.2 Flask Server

/setUser Endpoint, accessed through a POST request, allows for the registration of new users by providing their email and password in the request body as a JSON object with keys "email" and "password." A successful registration triggers a response with a status code of 200, signifying the successful addition of the user to the system. The response format for success includes pertinent details about the registration process.

/getUser Endpoint operates through the HTTP POST method and is designed for retrieving user information necessary for authentication. To utilize this endpoint, clients need to send a JSON-formatted request body containing the user's email and password. The expected keys in the JSON body are "email" for the user's email address and "password" for the user's password. Upon a successful authentication attempt, the server responds with a success status code of 200. The response format for success includes the relevant details needed for authentication, ensuring a standardized and clear communication between the client and the server.

/updateInstances Endpoint operates under the HTTP POST method and serves the purpose of updating the count of ResDB or SDK instances for a specific user. Clients interact with this endpoint by sending a JSON-formatted request body containing key-value pairs. The expected keys include "user_email" for the user's email address, "instance" specifying the type of instance (either "resdb" or "sdk"), and "inc" indicating whether to increment (true) or decrement (false) the instance count. This endpoint provides a flexible and straightforward mechanism for users to manage and adjust their ResDB or SDK instances as needed.

/getInstances Endpoint, accessible through the HTTP POST method, is designed to retrieve the current count of ResDB and SDK instances associated with a specific user. Clients interact with this endpoint by submitting a JSON-formatted request body that includes the "userEmail" key for the user's email address. Upon a successful request, the server responds with a success status code of 200. The response format for success provides details about the current count of ResDB and SDK instances, ensuring a standardized and clear communication between the client and the server regarding the user's instance status.

4.5.3 Monitoring Dashboard

Overview: The Instances page of ResUI contains a monitoring dashboard to track the performance and health of ResilientDB. Running exclusively on Ubuntu 20.04, we deployed ResilientDB within a Virtual Machine (VM) environment.

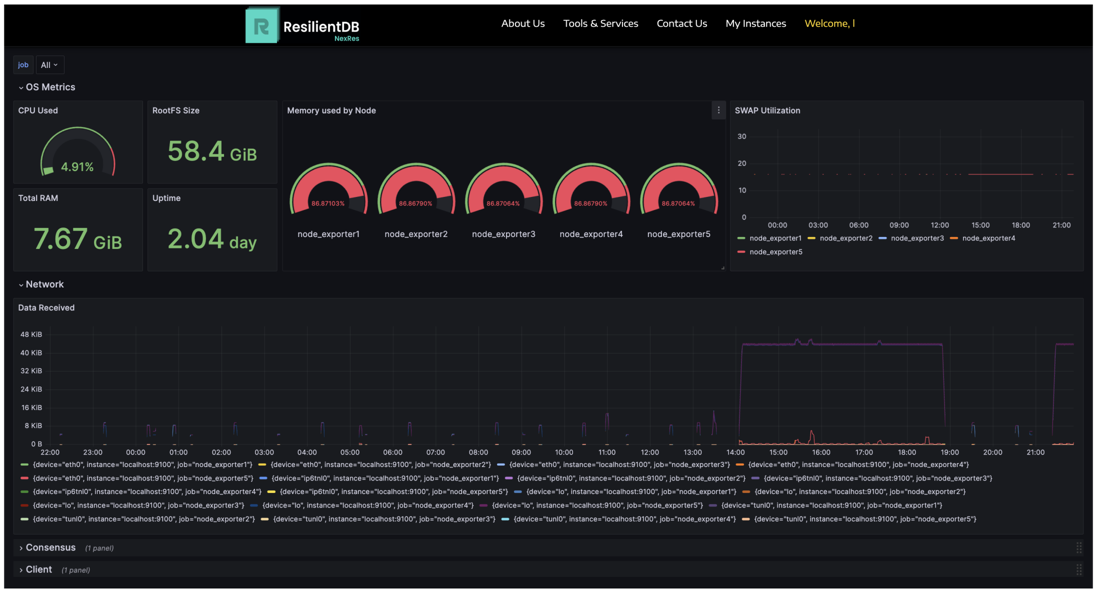

ResilientDB
NexRes
About Us
Tools & Services
Contact Us
My Instances
Welcome, I
job
All
OS Metrics
CPU Used
4.91%
RootFS Size
58.4 GiB
Memory used by Node
86.87103%
86.86790%
86.87064%
86.86790%
86.87064%
node_exporter1
node_exporter2
node_exporter3
node_exporter4
node_exporter5
SWAP Utilization
Total RAM
7.67 GiB
Uptime
2.04 day
Network
Data Received
48 KiB
40 KiB
32 KiB
24 KiB
16 KiB
8 KiB
0 B
{device="eth0", instance="localhost:9100", job="node_exporter1"}
{device="eth0", instance="localhost:9100", job="node_exporter2"}
{device="eth0", instance="localhost:9100", job="node_exporter3"}
{device="eth0", instance="localhost:9100", job="node_exporter4"}
{device="eth0", instance="localhost:9100", job="node_exporter5"}
{device="ip6tnl0", instance="localhost:9100", job="node_exporter1"}
{device="ip6tnl0", instance="localhost:9100", job="node_exporter2"}
{device="ip6tnl0", instance="localhost:9100", job="node_exporter3"}
{device="ip6tnl0", instance="localhost:9100", job="node_exporter4"}
{device="ip6tnl0", instance="localhost:9100", job="node_exporter5"}
{device="lo", instance="localhost:9100", job="node_exporter1"}
{device="lo", instance="localhost:9100", job="node_exporter2"}
{device="lo", instance="localhost:9100", job="node_exporter3"}
{device="lo", instance="localhost:9100", job="node_exporter4"}
{device="lo", instance="localhost:9100", job="node_exporter5"}
{device="tunl0", instance="localhost:9100", job="node_exporter1"}
{device="tunl0", instance="localhost:9100", job="node_exporter2"}
{device="tunl0", instance="localhost:9100", job="node_exporter3"}
{device="tunl0", instance="localhost:9100", job="node_exporter4"}
{device="tunl0", instance="localhost:9100", job="node_exporter5"}
Consensus (1 panel)
Client (1 panel)

Figure 14. Screenshot depicting installation steps on Dashboard

Database Configuration: To enable efficient monitoring, we instantiated a ResilientDB instance with 5 replicas, utilizing its Docker image. For services and monitoring, KV service monitoring was activated within the ResilientDB instance. This instance is communicated via port 8080 (localhost:8080), allowing us to capture and analyze the output.

Prometheus Integration: Prometheus is a powerful open-source monitoring and alerting toolkit specializing in collecting and storing time-series data. We used Prometheus to collect and store metrics from our ResilientDB instance. We configured the Prometheus YAML file before running it as a service to scrape metrics from the designated port 8080, ensuring a continuous flow of real-time data for analysis.

Grafana Dashboard Creation: Grafana, a leading open-source analytics and monitoring platform, was utilized to build a visually insightful dashboard for our ResilientDB instance. Leveraging Grafana's intuitive interface, we crafted panels to represent key metrics, enabling a comprehensive visualization of the database's performance and statistics. This dashboard served as a centralized hub for monitoring various aspects such as:

- node_cpu_seconds_total
- node_filesystem_size_bytes
- node_memory_MemTotal_bytes
- node_boot_time_seconds
- node_memory_MemFree_bytes
- node_memory_SwapFree_bytes
- node_memory_SwapTotal_bytes
- node_network_receive_bytes_total
- consensus
- client

Embedding the Dashboard into ResUI: The final step of integration ensures that users interacting with ResUI can access real-time insights into the health and performance of the underlying ResilientDB instance. The embedded dashboard provided a user-friendly interface for monitoring without the need to navigate to a separate tool.

5 Project Timeline

The Roadmap for this project unfolds across four distinct phases. The setup phase initiates the journey, encompassing the groundwork, infrastructure establishment, and resource allocation necessary for the project's execution. Following this, the implementation phase takes center stage, where the envisioned plans are implemented via three distinct features: (1) User Interface, (2) User Management, and (3) Containerization. Here, programming the back-end functionalities, designing the front-end interfaces, and Dockerization drive the project forward. Subsequently, the testing phase commences, emphasizing rigorous evaluations, bug fixes, and performance assessments to ensure the robustness and functionality of the project. Finally, the project delivery phase focuses on communicating the project's outcomes, showcasing its functionalities, and sharing insights gathered throughout its development.

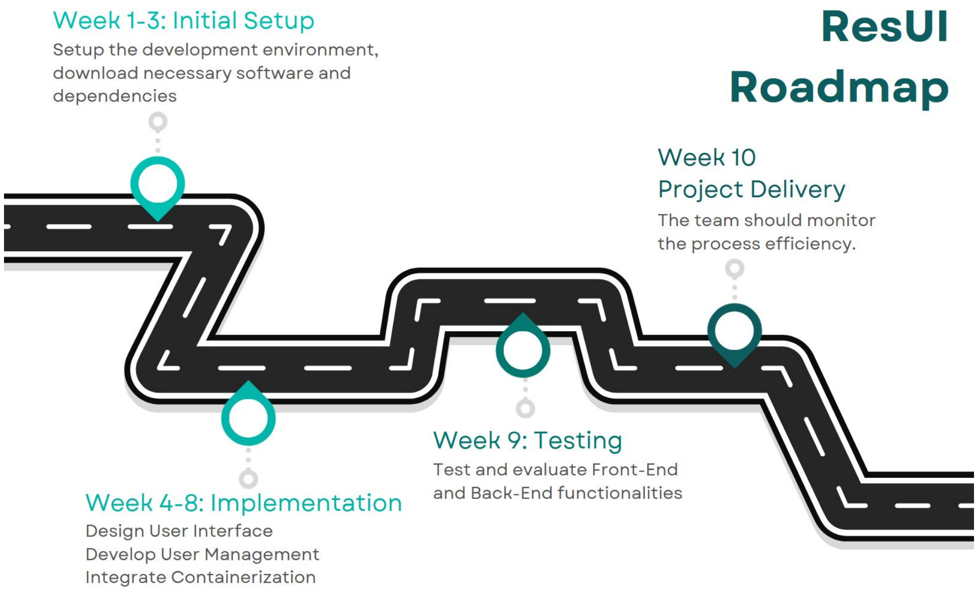

ResUI
Roadmap
Week 1-3: Initial Setup
Setup the development environment, download necessary software and dependencies
Week 4-8: Implementation
Design User Interface
Develop User Management
Integrate Containerization
Week 9: Testing
Test and evaluate Front-End and Back-End functionalities
Week 10
Project Delivery
The team should monitor the process efficiency.

Figure 15. Visual depiction of Project Timeline of ResUI

6 Future Works

While the current implementation of ResUI has successfully addressed key functionalities, potential avenues for future development and enhancement exist. In the Tools and Services tab, the Explorer section remains untapped, presenting an opportunity for redesign with a new user interface and advanced visualization techniques to provide a comprehensive view of the system's components. Similarly, the Monitoring and Prometheus sections within the Navbar could benefit from future development by implementing customizable alerts, trend analysis, and performance predictions, aligning with ResUI's goal of improving accessibility and appeal. Redesigning the Prometheus section specifically can offer a standardized and robust monitoring solution, enhancing reliability and performance observability for ResilientDB instances.

Future enhancements for ResUI include expanding monitoring capabilities in Grafana with advanced metrics and visualization options for a more comprehensive view of ResilientDB's performance. Optimizing the Docker setup and exploring container orchestration tools like Kubernetes could further enhance scalability and deployment management. User management improvements, such as implementing multi-factor authentication and additional account features, will enhance security and user experience. Integration with external tools, cross-browser compatibility testing, and thorough documentation creation contribute to a more robust and user-friendly ResUI. Fostering community engagement and conducting regular security audits ensure ongoing development and the application's long-term sustainability. Continuous performance optimization will guarantee ResUI's efficiency as user demand and the project's complexity evolve.

The outlined future work presents exciting opportunities for the continued evolution of ResUI. Addressing these aspects will not only enhance ResUI's capabilities but also contribute to its adaptability in diverse database management scenarios, ensuring a more versatile solution for users.

7 Conclusion

The development of ResUI represents a significant step in enhancing the user experience and accessibility of ResilientDB, a pioneering blockchain technology. By seamlessly integrating user-friendly interfaces with powerful backend technologies, ResUI offers a streamlined platform for users to interact with ResilientDB instances. The project successfully realized key functionalities, including user authentication, a monitoring dashboard, and instance management, while also identifying potential avenues for future development. Through meticulous design choices, continuous performance optimization, and a strong focus on security, ResUI provides a robust and versatile solution for users to navigate and interact with the capabilities of ResilientDB. As ResUI evolves with future enhancements and user feedback, it stands as a testament to the synergy between cutting-edge technologies and user-centric design, contributing to the broader ecosystem of high-throughput permissioned blockchain fabrics.

8 References

1. GitHub Link to the Project - `https://github.com/gopuman/ResUI`

2. Link to the Live Website - `http://104.154.189.181:3000/`

3. Blog Post - `https://shorturl.at/nBQT9`

4. The distributed database, ResilientDB - `https://resilientdb.com/`

5. Grafana information and installation - `https://grafana.com/`

6. Prometheus information and installation - `https://prometheus.io/`

7. Docker information and installation - `https://www.docker.com/`