

TO BE MOVED TO OVERLEAF OR BLOG

ResUI: User Interface for ResilientDB

Team Members: Gopal Nambair, Dieu-Anh Le, Rishika Garg, Sasha Pimento, Anubhav Mishra

1.INTRODUCTION: Inspiration/Motivation behind ResUI:

Our inspiration for this project is ResilientDB. ResilientDB is a revolutionary blockchain technology that was born out of the ExpoLab at UC Davis in 2018 and also marks a significant advancement in the field of High-Throughput Permissioned Blockchain Fabrics.

While most blockchain protocols use traditional consensus protocols like PBFT and HotStuff, ResilientDB uses novel consensus protocols like GeoBFT, Proof Of Execution, RCC, Spotless etc. It also provides a wide range of interfaces like the Key-Value store and Python SDK, which enables users to use ResilientDB for various functionalities, thus showing the versatility of the system. Additionally, ResilientDB not only gives users the ability to perform transactions in a decentralized and democratized way but also supports Grafana for plotting monitoring data. With its latest achievement of acceptance into the Apache Incubator, it paves the way for developers to contribute to projects that can potentially become top-level Apache Software Foundation projects.

With all these achievements in mind, the next big step would be to establish ResilientDB as a marketable product. A crucial selling point of any software product is its UI and to that extent, we have developed ResUI, a user friendly interface designed to simplify navigation for first time users and systematically present ResilientDB's features, thus reducing clutter on the screen. We have also developed a SignUp and Login functionality and an Instances page, where authenticated users can view the number of ResDB and SDK instances they are currently running along with a monitoring dashboard that enables users to analyze the metrics associated with those instances.

Our contribution:

2. FrontEnd/BackEnd Technologies

React.js:

HTML/CSS:

ResilientDB:

Node.js and Python:

- Node for sign up log in
- Python for command line

Prometheus:

At the center of our project's monitoring system is Prometheus, a crucial backend tool specializing in streamlined data collection. Tailored for scalability, it consistently retrieves real-time metrics from our running ResilientDB instance. Its simplicity and integration capabilities facilitate proactive monitoring, alerting us to potential issues and ensuring the sustained performance of our system.

Key contributions of Prometheus:

- Efficient Metric Collection: Prometheus excels in collecting and storing time-series data. Its ability to scrape metrics from various targets, such as our ResilientDB instance, ensures a reliable source of real-time data.
- Real-Time Monitoring and Insights: Prometheus contributes to our project's real-time monitoring capabilities. By consistently collecting and storing metrics, it provides immediate insights into the health and performance of our ResilientDB instance and facilitates prompt identification of anomalies, enabling proactive measures to maintain system efficiency.
- Alerting and Notifications: Integrated with Grafana, Prometheus enhances proactive monitoring through alerting and notification functionalities. Configuring alert rules ensure timely notifications, enabling quick response to potential issues.
- Scalable Architecture: Designed for reliability and scalability, Prometheus' architecture ensures seamless scalability, making it an ideal solution for projects with expanding data requirements.

Grafana:

Grafana, employed as a pivotal backend technology in our project, plays a crucial role in enhancing the monitoring and visualization aspects of our system. As an open-source analytics and monitoring platform, Grafana excels in transforming raw data into meaningful insights, fostering a proactive approach to system monitoring.

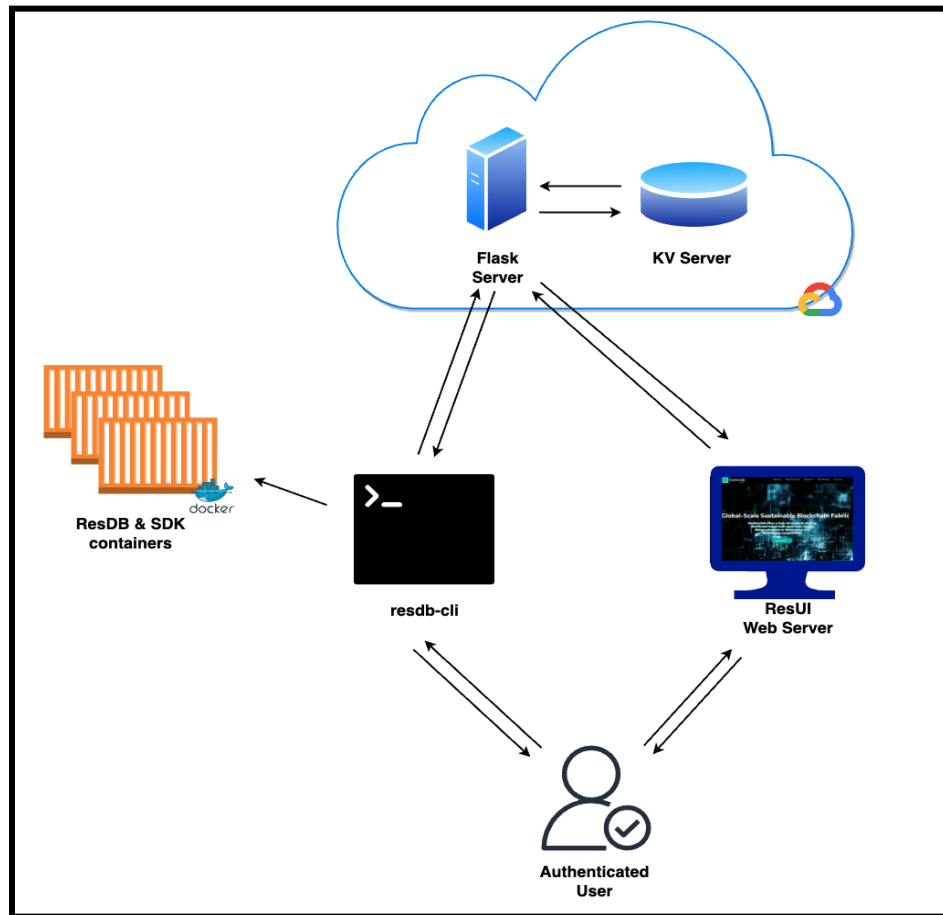
Key Contributions of Grafana:

- **Versatile Data Source Integration:** Grafana seamlessly integrates with various data sources, effortlessly connecting to Prometheus in our project to retrieve real-time metrics from ResilientDB.
- **Intuitive Dashboard Creation:** With a user-friendly interface, Grafana allows for the creation of visually appealing dashboards. Our tailored dashboard efficiently monitors critical aspects of RDB's performance.
- **Real-Time Visualization:** Grafana excels in providing real-time visualizations, offering dynamic and interactive panels. This capability ensures a comprehensive view of database metrics for prompt decision-making.
- **Embedding Capabilities:** Our integration of Grafana into RUI enhances the user experience by centralizing monitoring insights within the primary interface, streamlining accessibility.

Node Exporter:

Docker:

3. Architecture (include diagram)



4. Design Of UI (Each page should have a gif if its a blog or ample amount of screenshots if its a project report)

The user interface aspect of this project consists of multiple pages composed of multiple components. Besides a modern black and teal design theme, the visualization also avoids clustering of information by breaking down the information into smaller pages, as illustrated below.

1. Home/ Landing Page:

The Home Page consists of several components which are broken down as follows:

- **Navbar:** While designing the Navbar prototype, we realized that many elements of the existing ResilientDB landing page could be grouped together in the Navbar to give the users a more unified experience and an easy flow.
The Navbar currently consists of an About Us Tab, which on hover further expands to display other pages like the Mission & Vision, RoadMap and Publications page. Additionally, it also consists of a Tools & Services Tab, which gives users an ability to view the ResilientDB Github Repository and other tools like the Explorer, Monitoring and Prometheus pages.
- **Landing Page Background:** This section comprises a video animation running in the background along with a button for users to Get Started. Clicking on this button will direct users to the Sign Up Page that can enable users to create an account on ResilientDB.
- **Information and Services Components:** This section gives users some brief information about ResilientDB and also provides a services section where users can learn more about the different services that ResilientDB has to offer. Added hover effect offers engage user interaction with the website
- **Sponsors Component:** In order to develop the Sponsors section, we have implemented it as a carousel, using the Marquee effect in React for the 2 groups of sponsors.
- **Footer:**

2. About Us

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.

a. Meet the Team

The first component of the About Us page is 'Meet The Team,' highlighting the individuals contributing to the innovative concepts at ResilientDB. This section

follows a hierarchical structure, commencing with categories such as 'Our Advisors,' 'Our Visionaries,' 'Our NexRes,' and 'Our Alumni,' spotlighting the collective contributors driving ResilientDB's success.

Each team member is constructed using a card showing their picture, name, position, and an embedded LinkedIn link, created through the mapping of the CreateCard function to entries in TeamInfo.js. This automated process ensures effortless updates for new team additions.

b. RoadMap:

Roadmap is another major component of the About Us page, offers the user a better understanding of the path ResilientDB have taken until this day and the future plan.

In order to implement the Roadmap, we used the vertical-timeline-element in React to show each milestone along with an animation that engages the user and also helps in the storytelling process. Furthermore, we have also included buttons with certain milestone, for example if there is a paper or an announcement associated with the milestone, users can simply click on the button and will be redirected to the link in a new tab.

Similar to Meet The Team, each component of the roadmap is clearly defined, therefore developers can easily add more milestones to the roadmap by simply updating the database timelineElement.js using the appropriate format structure.

c. Publications

Now talking about the third component of our About us section, which is the publications section, for this section we took inspiration from the original ResilientDb's website and developed a section for the publication works. In this section, we display most of the research works, that have been published in our esteemed ExpoLab by various developers who have worked in the development of ResilientDb ever since the start. In addition to this, we have also highlighted the award-winning research works that have been published, with the name of the conference in which they were given the award such as the Best Paper Award for the paper titled "SGX Accelerated Consensus" in the EuroSys conference in the year 2023. This feature helps us in adding an extra emphasis to the top research works for visitors who want to read about some of the best research works.

All these research papers have a specific tile associated for them with a hyperlink added to them which redirects them to the respective research paper.

Talking about the development of all these pages as mentioned in the prior technologies section we have made use of the ReactJS front-end framework and made a separate component for each of the pages. Additionally, to be precise we made use of the Styled-Components library of ReactJS which helps us in the styling of our pages which helps in writing CSS inside JavaScript files using tagged template literals.

3. Contact Us

Another functionality that we have added to the navbar of our ResUI project is the Contact Us section. For any technical project where there could be some things in the project which are unclear and need some extra clarification or just in general maybe someone is interested in how the project is implemented both for the frontend and backend areas, or even if someone wants to just request an extra feature or some flaw of the website which needs to be reported to get an immediate fix. For all this purpose

4. Sign Up & Login (Backend functionality in detail)

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.

Server.js

Overview

The server.js file serves as the backend 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.

Signup > index.js

Overview

The `signup > index.js` file contains the frontend logic for user registration. It features a signup form that collects user details, performs client-side validation, and sends a request to the backend 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 backend 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.

Login > index.js

Overview

The `login > index.js` file handles user login on the frontend. It provides a login form with fields for email and password, sending a request to the backend for authentication.

Login Form

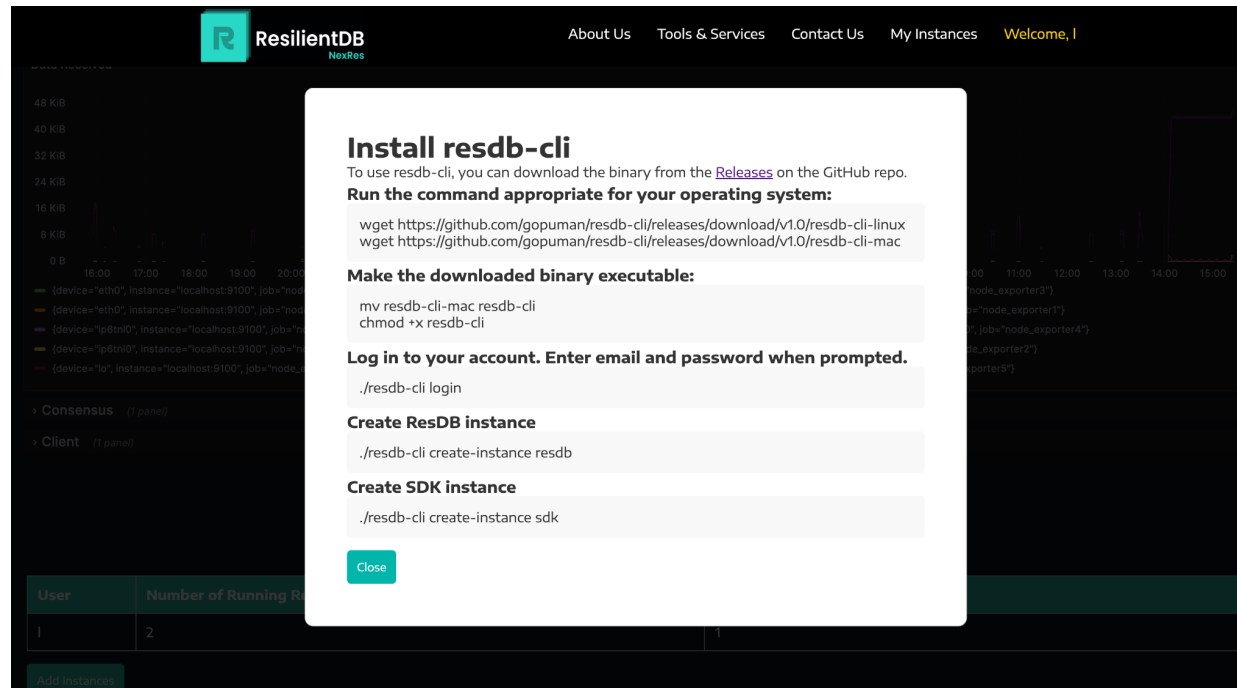
The login form includes fields for the user's email and password. Upon form submission, a fetch request is made to the `/api/login` endpoint on the server. If the login is successful, a token is stored in the local storage, and the user is redirected to a protected route or the home page.

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.

5. Instances Page

a. Mention about CLI / commands used?



The screenshot shows the ResilientDB web interface. A modal window titled "Install resdb-cli" is displayed in the center. The modal contains the following text:

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

A "Close" button is located at the bottom of the modal.

In the background, the ResilientDB dashboard is visible, showing a graph of memory usage (0 B to 48 KIB) and a table of running instances.

User	Number of Running Instances
1	2

```
$ ./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
```

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:

Visit the [Releases](#) page.

Download the latest release suitable for your operating system (e.g., resdb-cli-linux for Linux).

Make the downloaded binary executable using the following command:

```
chmod +x 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:

Create a configuration file named config.ini.

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
```

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

Login: Logs in to the specified user account.

`./resdb-cli login #` Enter email and password when prompted

Logout: Logs out from the current user account.

`./resdb-cli logout`

Create Instance: Creates a new ResDB or Python SDK instance.

`./resdb-cli create_instance <type>`

View Instances: Displays details about running instances.

`./resdb-cli view_instances`

Delete Instance: Deletes a running ResDB or Python SDK instance.

`./resdb-cli delete_instance <instance_id>`

Current User: Displays the currently logged-in user.

`./resdb-cli whoami`

For more detailed information about each command, users can run `./resdb-cli --help` or `./resdb-cli <command> --help`. The provided help functionality ensures clarity and ease of use.

b. Flask server / calls been made

1. `/setUser` Endpoint

Method: POST

Purpose: Registers a new user with the provided email and password.

Request Format:

JSON body with keys:

email: Email of the user.

password: Password for the user.

Response Format:

Success (Status Code 200):

2. `/getUser` Endpoint

Method: POST

Purpose: Retrieves user information for authentication.

Request Format:

JSON body with keys:

email: Email of the user.

password: Password for the user.

Response Format:

Success (Status Code 200):

3. /updateInstances Endpoint

Method: POST

Purpose: Updates the count of ResDB or SDK instances for a user.

Request Format:

JSON body with keys:

user_email: Email of the user.

instance: Type of instance (resdb or sdk).

inc: Increment (true) or Decrement (false) the instance count.

4. /getInstances Endpoint

Method: POST

Purpose: Retrieves the current count of ResDB and SDK instances for a user.

Request Format:

JSON body with keys:

userEmail: Email of the user.

Response Format:

Success (Status Code 200):

c. Calls to endpoint in code?

- `const { success, message } = await makeApiRequest("setUser", "POST", { email, password });`
- `const { success, message } = await makeApiRequest("getUser", "POST", { email, password });`
- `const { success, resdb, sdk } = await makeApiRequest("getInstances", "POST", { userEmail });`
- `response = make_flask_request("updateInstances", "POST", {"user_email": current_user, "instance": container_type, "inc": False})`

- `response = make_flask_request("getUser", "POST", {"email": username, "password": password})`
- `response = make_flask_request("updateInstances", "POST", {"user_email": current_user, "instance": type, "inc": True})`

d. Grafana/ Prometheus

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.

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 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 involved embedding the Grafana dashboard into our full-stack project, ResUI. This integration ensured that users interacting with ResUI could 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. How to run the application

6. Project Timeline (?)

- Make a diagram for this (Anh)
 1. Planning
 2. Software implementation
 3. Parallelisation of tasks
 - a. User Interface
 - b. User Management
 - c. Containerization
 4. Merging components and testing
 5. Launching

7. Future Discussion:

Adding pages

Docker implementation

8. Conclusion:

9. References: