General Documentation
Demand Response Portal

Prepared by the students of CSC 4615 – Fall 2023

Elijah C Monroe
Evyn Price
Mykola (Nick) Omelchenko
Serena L LaBelle
Shelby Smith
William Goodson
Won (Brian) Lee

Updated on November 28, 2023

Under the direction of

Eric L. Brown

Table of Contents

1. Introduction

1.1. Purpose

The purpose of this general documentation is to provide comprehensive and accessible information about the project. This is designed to serve as a valuable resource for users, administrators, and developers alike, and will assist future contributors in learning about the project.

1.2. Scope

The Demand Response Portal allows users to visualize the data from the meters that participate in the demand response program. It is only able to use and manipulate the data accessible in the associated database. It provides user and organization management. It does not automatically import real meter data or produce predictions from the data, so the power distributor must add the prediction data through the provided API calls.

1.3. Audience

The intended audience for this documentation is the developers and system administrators who plan on continuing work on the application, as well as any other stakeholders who wish to demo the Demand Response Portal.

1.4. Document Conventions

We were not invited or aware of any document conventions that took place in the fall of 2023, so we were unable to attend. However, we are looking forward to attending in the future.

2. System Overview

2.1. System Architecture

The Demand Response Portal system is comprised of four main components: Central Database, Cache Database, API Server, Reverse Proxy Server. The Reverse Proxy Server serves as the entry point for users. From there, the web client makes requests to the API Server. The API Server will then access the Central Database, and it will cache specified data in the Cache Database for quicker access.

2.2. Key Components

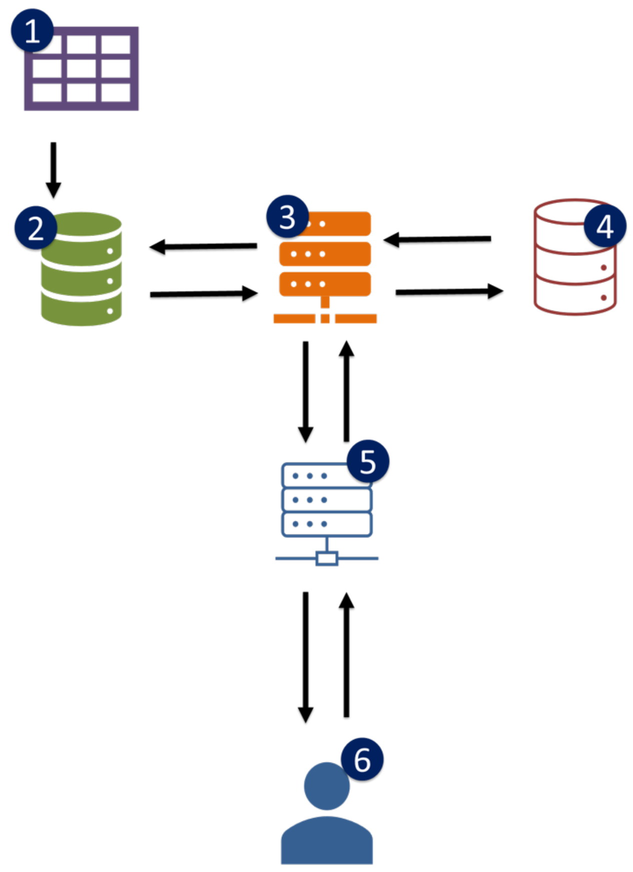
The Central Database is using MariaDB, which is fork of MySQL that provides improvements with a more open-source license, to store all of our user and meter, and it helps ensure data integrity with the use of foreign keys and triggers.

The Cache Database is using Redis, which is an in-memory key-value store that provides high-performance data handling, to cache our session data and meter data for quick access to improve the user experience.

The API Server is using Flask, which is a python-based web server, to serve the data from the databases to the user. We developed our API following RESTful principles, adhering to standard HTTP methods. The flask server also runs through a WSGI (Web Server Gateway Interface) server that uses worker processes to scale and optimize the performance of the API server.

The Reverse Proxy Server is using NGINX, which is an open-source web server and reverse proxy server, as both a web server to host our static web portal as well as a reverse proxy to pass requests from the client to the API Server.

2.3. System Diagram



1. DB Importer 2. Central Database 3. API Server 4. Cache Server 5. Reverse Proxy Server 6. User

2.4. Dependencies

The only dependency is a version of docker or docker desktop that supports docker compose. For users who wish to manage the database independent of the web application, a SQL management utility such as MySQL Workbench, TablePlus, or SQL Server Management Studio is required.

3. Installation and Setup

3.1. Prerequisites

This software requires Docker Desktop version 4.25.2 or newer, or docker engine version 24.0.7 or newer. Docker compose is required to manage the containers, which is bundled in the latest product versions.

3.2. Installation Steps

1. Clone the GitLab repository
2. Duplicate the .env.example file located in the src directory, and rename it to .env
3. Update the values in the .env file to match your needs
4. Run the command `docker compose up -d --build` from the src directory.

3.3. Configuration

There are three main configuration files that can be modified to adapt the portal to the users' specifications:

- drp-config.json: This config contains global values to manage the application, such as setting the credit factor.

- provider-config.json: This file provides the default power provider user credentials and organization.

- nginx-config: Configuration for webserver, such as location of HTTPS certificate.

- gunicorn-config.py: Configuration for the WSGI layer for the api server.

4. User Guide

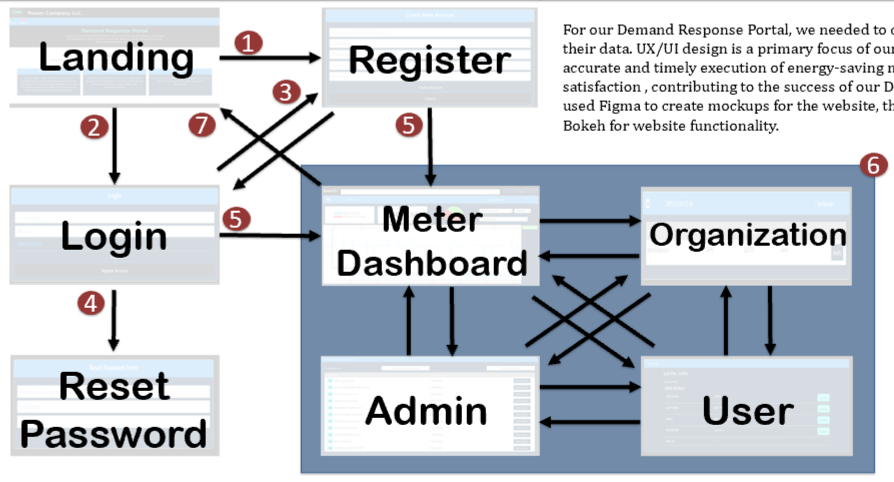
4.1. Getting Started

To start the application, run the command `docker compose up -d` from the project src directory. This will start all the required docker containers in "detached" mode. After a few moments, the database should be initialized, and data will begin importing from a provided CSV.

After the application has been initialized, it can be accessed through the web browser on the same system at https://localhost, or from another system at its provided hostname or IP address. Users in enterprise environments should expose port 443 to allow web traffic from the nginx proxy.

4.2. User Interface

UX/UI DESIGN



For our Demand Response Portal, we needed to design a front end for users to access their data. UX/UI design is a primary focus of our project, because it facilitates the accurate and timely execution of energy-saving measures and improves client satisfaction , contributing to the success of our Demand Response Portal project. We used Figma to create mockups for the website, then used HTML, JavaScript, CSS and Bokeh for website functionality.

PAGE DESCRIPTION

- **Landing Page** : Shows the basic information about the project.
- **Register Page** : Lets users register an account
- **Login Page** : Lets users with an account login
- **Password Reset Page** : Lets users with an account reset their passwords
- **Meter Dashboard** : Displays the bokeh plot of the meters the user has access too
- **Organization Page** : Allows Org Admins to alter users within their organization
- **Admin Page** : Lets Distributor Admins manage organization groups
- **User Page** : Lets users manage their own accounts.

TYPICAL USER JOURNEY

The typical user journey starts when users access the website where they are brought to the landing page.

1. If the users do not have an account, they can register.
2. If the users have an account, they can login.
3. The users can switch between the two if they accidently chose the wrong option.
4. If the user forgot their password, they can navigate to the reset password page to reset it.
5. After the user registers the account or logs in they are brought to the meter dashboard.
6. using the navigation bar at the top of the page they can move between the meter dashboard, the admin, the organization page, and the user page however they want.
7. When the user logs out, they are brought back to the landing page.

4.3. Functionality

Upon navigating to the web portal, users are brought to a landing page which provides information about the project and the features it offers.

A navigation bar at the allows users to navigate to each of the pages. Users will only be able to see pages to which they have access (ex: Only project administrators can navigate to the administrator's page). Users who have not logged in will be presented two options: Login or Create Account.

Creating an account requires a meter number or organization number, both of which can be added using the default provider organization account. If a user forgets their password, they can reset it with their meter or organization ID.

Upon logging In, users are brought to the meter's page. This page provides statistics on the meters in which the user has access. This includes the total savings rate, participation percentage, and real-time power usage compared to predictions. Using the drop downs menu on the right, user's can select which organization to view, meter ID, and timescale. Clicking "more details" includes additional data analysis metrics.

The organization page allows organization administrators to modify their members. Clicking "modify" allows the administrator to set the permissions level for the user (pending, rejected, approved, or administrator).

The user page allows a user to edit their own account information, such as their name, email address, and password.

The admin page allows a user to manage organizations and meters within the application. This includes creating new organizations and meters, as well as assigning meters to organizations. Clicking the green button to the left of the organization allows a user to modify the organization name.

4.4. Troubleshooting

No users can log in or create accounts – This commonly occurs because the web server fails to connect to the database. Check the .env file to ensure that the database host is set correctly. If docker networking is unchanged from the original source, the database host should be the container name ‘db’.

I don’t see any data in the models – Ensure that the db-importer container is running correctly. Verify that src/db/meter-data.csv exists and contains valid meter data.

5. Admin Guide

5.1. Administrator Roles

System Administrator – This is a user who is an admin of the default (provider) organization. They hold permission level 0 and have complete control of the application. Users with this permission level can see the administrator page, manage all organizations, and inherit all the permissions from roles below.

Power Provider – This is a user who is in the default (provider) organization. They currently hold no permissions greater than the organization administrator and serve as a placeholder.

Organization Administrator – This is a user who has the ‘administrator’ role of any organization other than the power provider organization. They can see the organization page and manage other members in their org.

Organization User – This is a member of an organization. They can select multiple meters within their organization.

Residential User – This is a user who only has a single meter attached to their account. They can view the visualizations page and manage their own account.

5.2. Configuration

Administrators can manage organizations, meters, and their assignments through the web application. Other functionality such as the database credentials can be managed through configuration files provided in the src directory. See the above setup guide for specific configuration guidance.

5.3. Maintenance

Project administrators should ensure that the docker containers remain up to date. The docker-compose.yml configuration file details the specific version for each docker container to ensure compatibility. Future teams should test latest versions of the containers to ensure that features remain functional.

5.4. Security

As stated in 5.3, administrators should ensure that docker containers remain up to date to patch new vulnerabilities and exploits. Organizations exposing the web application to the public should create a signed certificate to replace the self-signed version included in the Nginx configuration.

Future developers should also review the data flows between the web application and SQL server to ensure compliance with all security frameworks such as OWASP Top 10.

6. API Documentation

6.1. API Endpoints

- /api/passwordReset

    - Reset a user’s password
- /api/data/import
    - Import data to the database
- /api/data/export
    - Export data reports
- /api/entity/meters
    - Get accessible meters based on session id
- /api/entity/meters/add
    - Add meter to database
- /api/entity/meters/assign
    - Assign meters to user
- /api/entity/meters/unassigned
    - Get all meters that are not assigned
- /api/organization (get, put)
    - Get organization data
    - Update organization data
- /api/user/session/validate
    - Check session validity
- /api/user/session/create
    - Create a new session; sets cookie
- /api/user/session/delete
    - Delete session; used for logout
- /api/user
    - Get user data
- /api/user/update
    - Update user data
- /api/user/delete
    - Delete user data
- /api/user/register
    - Create a new user
- /api/user/permissions
    - Get user’s permission level
- /api/vis/
    - Fetches initial bokeh graph
- /api/vis/meter_data/<meter_id>
    - Data callback for the main graph
- /api/vis/pie_data/<meter_id>
    - Data callback for the participation pie chart
- /api/vis/total_data/<meter_id>
    - Data callback for total data
- /api/vis/acf_data/<meter_id>
    - Data callback for acf data in more info
- /api/vis/pacf_data/<meter_id>
    - Data callback for pacf data in more info
- /api/vis/sig_data/<meter_id>
    - Data callback for sig data in more info

- /api/vis/data/
    - Fetches real and prediction data from database

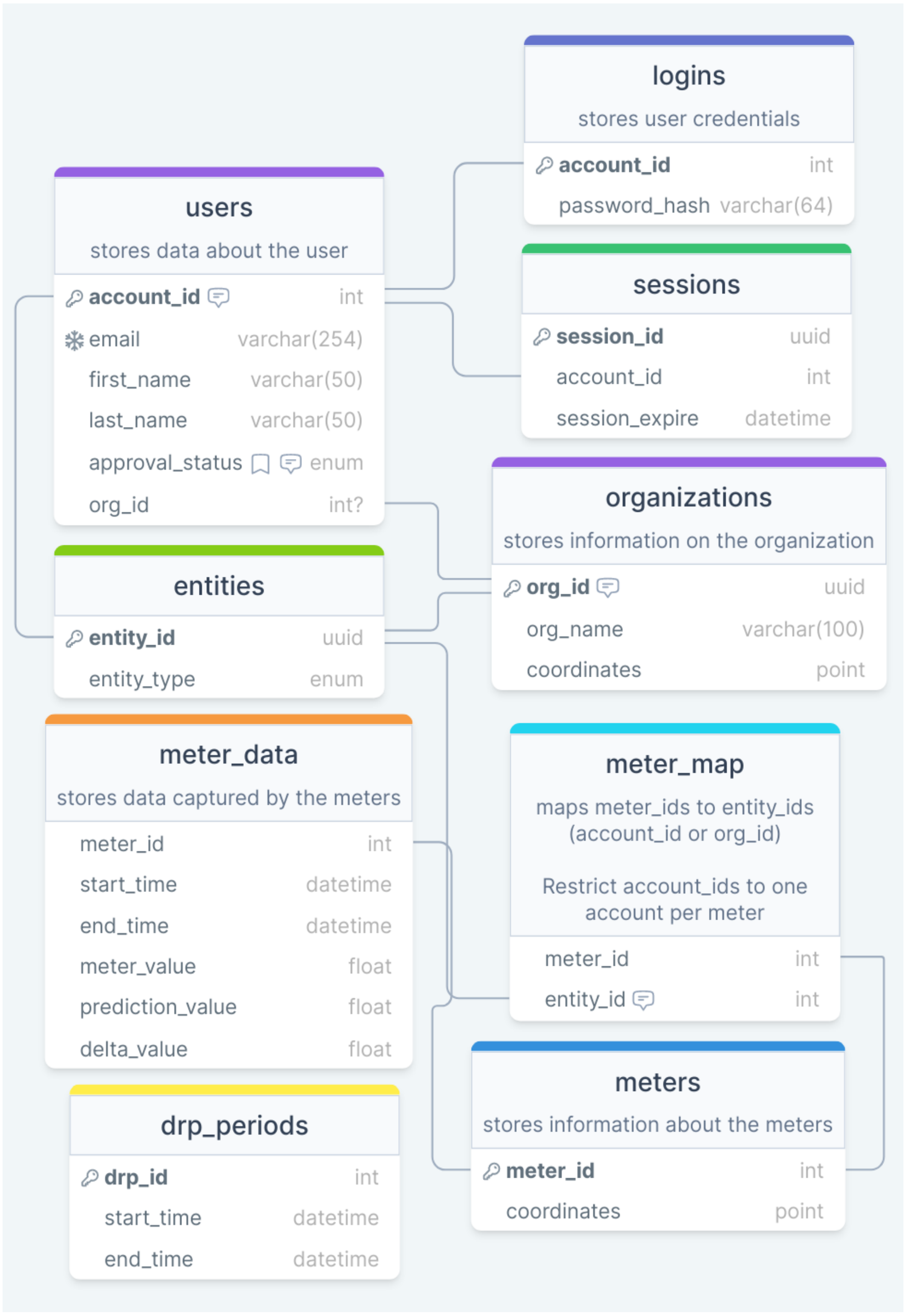
6.2. Authentication
The API is on a private docker network and inaccessible outside of localhost. It is intended to be used through the web portal.

6.3. Usage Examples
The intended usage of the API is through the web portal. The only requests not made through the web portal are with the data import calls.

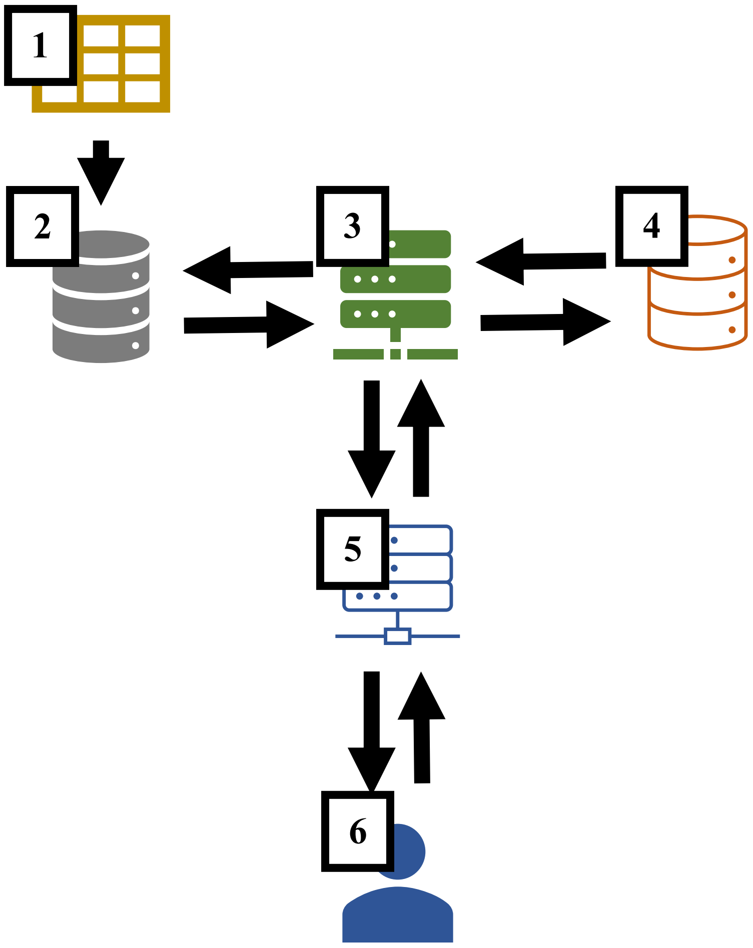
6.4. Rate Limits
No rate limits were implemented

7. Data Model

7.1. Database Schema



7.2. Data Flow



Here is a description of each of the servers:

1. **DB IMPORTER**: Fills the Database with the data received from client
2. **DB (MariaDB)**: Holds all the data.
3. **FLASK**: This is our API. Allows us to interact with the database using http methods (GET, POST, DELETE, etc.)
4. **DB CACHE (REDIS)**: Caches specific data in memory for fast delivery (sessions, meter data, meter-user connections)
5. **NGINX**: Pulling double duty as a web server for thing single page app and also a reverse proxy for the flask server
6. **USER**: Person accessing the website

7.3. Data Entities

To ensure data integrity, automating certain data management tasks, and controlling the associations and permissions within the database; we created the following triggers.

- **add_user_id:** Generates user IDs for new user records.
- **add_org_id**: Creates organization IDs for new organization records.
- **limit_residential_associations**: Ensures one user per meter association to maintain data integrity.
- **link_user_entity_delete**: Deletes user-related entries from the "entities" table upon user deletion.
- **link_org_entity_delete**: Removes organization-related entries in the "entities" table when organizations are deleted.
- **populate_deltas**: Calculates and updates meter data delta values.
- **org_prime_admin**: Automatically assigns admin status to the first user in an organization.
- **approve_residential**: Automatically approves users not associated with organizations.
- **disallow_org_user_map:** Prevents users in organizations from mapping to meters.
- **assign_distributor_meters:** Assigns newly added meters to a distributor entity for tracking.

8. Deployment

8.1. Deployment Architecture

The demand response portal system is designed to run anywhere that has docker installed.

8.2. Deployment Steps

1. Ensure docker (or docker desktop) is installed

2. Clone the repository

3. Copy the .env.example file and rename it .env

    - You may also want to configure it at this time

4. From the source folder, open a terminal and run `docker compose up -d –build`

    - Depending on your system you may need to use sudo

All of the docker containers should start. The db-importer container will start importing historical data. After it is finished, it will stop.

8.3. Scaling and Load Balancing

We do not implement any additional load balancing or scaling. We do run our flask api through gunicorn which adds worker processes.

9. Security

We have a much larger document detailing the security and risk assessment, see that for more details about this section.

9.1. Authentication and Authorization

We have multiple steps to the Authentication and Authorization Process.

1. We create a default admin user to allow the client to access the system and approve all other users.
2. When a new user accesses the website they can register the account and wait for approval
3. When the user is approved, they can then view their data.

9.2. Encryption

We encrypt the users passwords, by using bcrypt. We salt and hash all user passwords.

9.3. Security Best Practices

Keep Software Updated: Regularly update operating systems, applications, and software to patch vulnerabilities.

Use Strong Authentication: Implement strong, unique passwords and consider multi-factor authentication (MFA) for an additional layer of security.

Data Encryption: Encrypt sensitive data in transit and at rest to protect it from unauthorized access.

Firewall Protection: Employ firewalls to monitor and control incoming and outgoing network traffic, acting as a barrier between a trusted internal network and untrusted external networks.

Regular Backups: Perform regular backups of critical data, and test restoration procedures to ensure data recovery in case of a security incident.

Access Control: Implement the principle of least privilege, granting users the minimum level of access needed to perform their job functions.

Security Awareness Training: Educate employees about security threats, social engineering attacks, and best practices to recognize and avoid them.

Incident Response Plan: Develop and maintain an incident response plan to effectively respond to and mitigate security incidents.

Network Security: Use intrusion detection and prevention systems, and regularly scan and monitor network traffic for unusual activities.

Endpoint Security: Install and update antivirus software, and employ endpoint protection mechanisms to secure individual devices.

Secure Configurations: Ensure that systems and applications are configured securely, following best practices and removing unnecessary services and features.

Vendor Security Assessment: Assess and monitor the security practices of third-party vendors and service providers.

Physical Security: Secure physical access to servers, networking equipment, and other critical infrastructure.

Logging and Monitoring: Implement comprehensive logging and monitoring to detect and respond to security incidents in a timely manner.

Patch Management: Establish a patch management process to promptly apply security patches and updates to software and systems.

Mobile Device Management (MDM): Implement MDM solutions to manage and secure mobile devices used within the organization.

Security Audits and Assessments: Conduct regular security audits and assessments to identify vulnerabilities and weaknesses in the system.

Security Policies: Develop and enforce security policies that outline acceptable use, data handling, and other security-related guidelines.

Secure Development Practices: Integrate security into the software development lifecycle to identify and address vulnerabilities during the development process.

Continuous Monitoring and Improvement: Establish a continuous improvement cycle for security, adapting to emerging threats and evolving best practices.

Remember that security is an ongoing process, and organizations should stay vigilant, adapting their practices to address new threats and vulnerabilities.

For more information see Risk Assessment Report

10. Monitoring and Logging

This section is very limited, and is recommended to be worked on in the future.

10.1. Logging Mechanisms

Our Logging Mechanisms are very limited. We have some logging capabilities, but due to the nature of Docker it limits the amount of information we can know about the user. This causes the information to not be very useful.

10.2. Monitoring Tools

Docker does have logging capabilities, but they are very limited.

10.3. Alerts and Notifications

We do not have any Alerts and Notifications set up for the system.

11. Maintenance and Updates

We use Github to manage all Maintenance and Updates. We use standard Github practices. If more information on these practices is required there are many resources that can be found online.

11.1. Patching and Upgrades

We use Github to manage pull requests. This allows us to patch and upgrade the system as we continue with the project.

11.2. Backup and Recovery

For Backup and Recovery of the project we store the project on Github, and each member has clones on their machine. It would be very unlikely for us to lose the entire project.

11.3. Version Control

We use Github to manage Version Control. We created branches and we merged them into main.

12. Troubleshooting

12.1. Common Issues

If the dashboard is not updating as fast as you would want on change, it is not a network issue but a polling interval issue. You can change this in the vis file.

12.2. Debugging Techniques

For Debugging we read console log in the web browser, and the logging docker to see where the errors occur.

12.3. FAQs

1. Installation and Setup:

Q: What are the prerequisites for installing the Demand Response Portal?

A: The software requires Docker Desktop version 4.25.2 or newer, or Docker Engine version 24.0.7 or newer. Docker Compose is required to manage the containers, bundled in the latest product versions.

Q: How do I install the Demand Response Portal?

A: Follow these steps:

Clone the GitLab repository.

Duplicate the .env.example file in the src directory and rename it to .env.

Update the values in the .env file to match your needs.

Run the command docker compose up -d --build from the src directory.

Q: What configuration files can be modified to adapt the portal to user specifications?

A: There are three main configuration files:

drp-config.json: Contains global values to manage the application, such as setting the credit factor.

provider-config.json: Provides default power provider user credentials and organization.

nginx-config: Configuration for the web server, such as the location of the HTTPS certificate.

2. User Guide:

Q: How do I start the Demand Response Portal application?

A: Run the command docker compose up -d from the project's src directory. Access the application through the web browser at https://localhost or from another system using its provided hostname or IP address.

Q: What features are available in the user interface?

A: The user interface provides a landing page with information about the project and its features. A navigation bar allows users to access different pages based on their roles. Users can log in or create an account, and those without login credentials can choose to log in or create an account.

3. Admin Guide:

Q: What roles and responsibilities do administrators have in managing the Demand Response Portal?

A: Administrators have various roles, including defining administrator roles, managing configuration options, performing maintenance tasks, and addressing security-related measures. Detailed information is available in the Admin Guide.

Q: How can administrators manage the configuration of the system?

A: System configuration options are detailed in three main configuration files: drp-config.json, provider-config.json, and nginx-config. Administrators can modify these files to adapt the portal to user specifications.

4. API Documentation:

Q: What API endpoints are available in the Demand Response Portal?

A: The API documentation lists and describes available API endpoints, including their purpose and usage. It provides a guide on authentication mechanisms required to access the API and includes practical usage examples.

Q: How can I use the API to accomplish common tasks?

A: The API documentation includes practical examples demonstrating how to use the API to accomplish common tasks. It outlines authentication mechanisms and rate limits associated with the API.

5. Security:

Q: How is user authentication and authorization managed in the Demand Response Portal?

A: The authentication and authorization process involves creating a default admin user, allowing new users to register, awaiting approval, and then accessing their data upon approval. Detailed steps are outlined in the Security section.

Q: What encryption methods are used to secure user passwords?

A: User passwords are encrypted using bcrypt. The system employs salting and hashing for all user passwords to enhance security.

6. Monitoring and Logging:

Q: What logging mechanisms are in place for the Demand Response Portal?

A: Logging mechanisms are detailed in the documentation. Note that due to the nature of Docker, logging capabilities are limited, and information may be constrained.

Q: Are there any monitoring tools integrated into the system?

A: The documentation notes that monitoring tools are limited within Docker. Consider external monitoring solutions for more comprehensive monitoring.

7. Maintenance and Updates:

Q: How are patches and upgrades managed in the Demand Response Portal?

A: Maintenance and updates are managed through Github using standard practices, including creating branches and merging them into the main branch.

Q: What measures are in place for backup and recovery of the project?

A: The project is stored on Github, and team members have clones on their machines, minimizing the risk of data loss. Backup and recovery details are available in the Maintenance and Updates section.

8. Troubleshooting:

Q: How can I address common issues with the dashboard not updating promptly?

A: If the dashboard isn't updating as expected, check the polling interval in the vis file. Changing this interval should resolve the issue.

Q: What debugging techniques are available for the Demand Response Portal?

A: For debugging, read the console log in the web browser and the logging docker to identify and address errors.

9. Support and Contact:

Q: How can I contact the support team for assistance?

A: Support channels include email. Reach out to the team members listed in the Support and Contact section for assistance.

Q: Is there a specific response time for support requests?

A: The documentation doesn't specify a response time. Users are encouraged to contact the team by email for prompt assistance.

10. General Questions

Q: Why is the map no longer on the dashboard?

A: If the map does not appear you will probably need to switch its Google Api Key.

Q: Is there an interpretive dance representation of the project?

A: No. Around iteration 5 we realize the interpretive dance was not choreographed enough to be able to preform for the client at a professional standard. We do hope the next team who takes up this project can create one suitable for the client.

13. Support and Contact

13.1. Support Channels

You can contact the team by email if any problems arise.

13.2. Contact Information

Elijah Cain Monroe – ecmonroe42@tntech.edu
Enora Boscher - emboscher42@tntech.edu
Evyn Price – edprice42@tntech.edu
Mykola (Nick) Omelchenko – mmomelchen42@tntech.edu
Serena L LaBelle – sllabelle42@tntech.edu
Shelby Smith – sdsmith46@tntech.edu
William Goodson – wngoodson42@tntech.edu
Won (Brian) Lee – wlee44@tntech.edu .

14. Appendices

14.1. Glossary

API: Application Programming Interface. A set of rules and protocols for building and interacting with software applications.

CSV: Comma-Separated Values. A plain text file format used to represent tabular data, where each value is separated by a comma.

Docker: A platform for developing, shipping, and running applications in containers.

Flask: A lightweight Python web framework used for building web applications.

HTTP: Hypertext Transfer Protocol. The foundation of any data exchange on the Web.

HTTPS: Hypertext Transfer Protocol Secure. The secure version of HTTP, encrypted with SSL/TLS.

IP: Internet Protocol. A set of rules governing the format of data sent over the internet.

JSON: JavaScript Object Notation. A lightweight data interchange format.

MariaDB: An open-source relational database management system, and a MySQL fork.

MDM: Mobile Device Management. Software that allows IT administrators to control, secure, and enforce policies on smartphones, tablets, and other devices.

NGINX: An open-source web server and reverse proxy server

RESTful: Representational State Transfer. An architectural style for designing networked applications.

SQL: Structured Query Language. A domain-specific language used for managing and manipulating relational databases

SSL: Secure Sockets Layer. A protocol for establishing secure communication links between clients and servers in computer networks

TLS: Transport Layer Security. The successor to SSL, providing secure communication over a computer network.

URL: Uniform Resource Locator. A reference or address used to access resources on the internet.WSGI: Web Server Gateway Interface. A specification for a universal interface between web servers and Python web application frameworks.

14.2. Acknowledgments

Special Thanks for help on this project to:

Benjamin Burchfield

Travis Lee

Rajesh Manicavasagam

14.3. Version History

Ver 1.0: The original documentation from the first DRP Team