Technology Stack Proposal

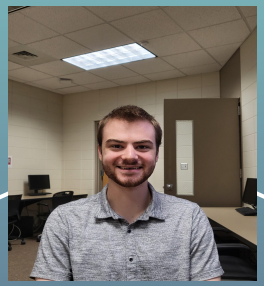
EVIZ

05/24/24

Presentation to Energy Economics Team

Kenny Howes and Edom Maru

Meet the 2024 Summer Research Team



Kenneth Howes

Calvin 3rd year Computer Science

Edom Maru

Calvin 3rd year Computer Science

Requirements Document

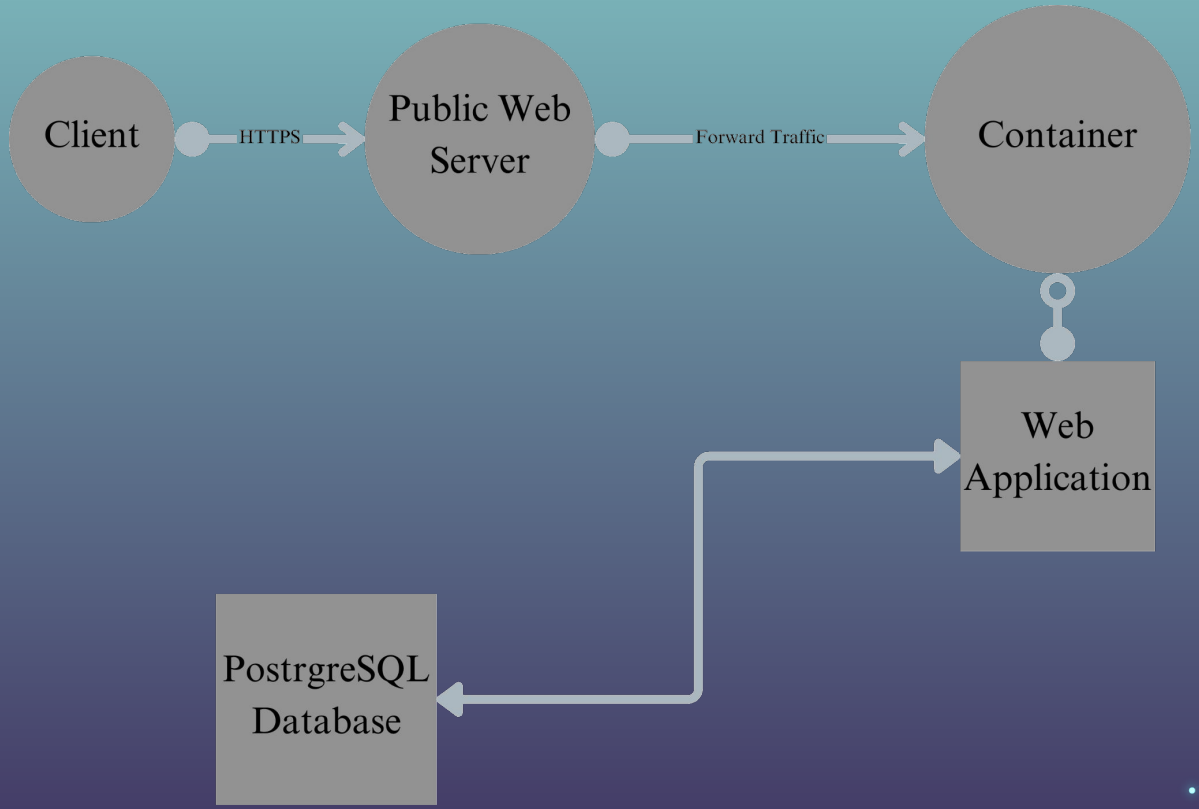
Functional Requirements

1. The site will facilitate sandbox-like, free exploration of data.
    i. The site will allow users to form queries based on the various columns present in the PSUT data table which will be used to retrieve data from the database.
    ii. The site will allow users to graphically visualize data that they recieve.
        a. A user will be able to visualize the flow of energy from the primary stage to the last stage of an analysis in a Sankey diagram.
        b. A user will be able to visualize the RUVY matrices corresponding to their desired data.
        c. A user will be able to create graphs of the following:
            - machine efficiencies
            - ECC stage efficiencies
            - product stage efficiencies
            - industry stage efficiencies
    iii. The site will allow users to download data in both Excel and CSV formats.
2. The design will be structured in such a way that the project is initially deployable on EVIZ and can easily be migrated to a cloud provider at a later date.
3. The site will allow for users to authenticate and be authorized to interact with the IEA proprietary data.
    i. The site and client will communicate within a secure connection.
4. The site will be able to interact with multiple data sets, each possibly within separate databases on the same database management system.
5. The site will relatively agnostic to any machines, energy carriers, or row & column names including if those attributes are changed or new entries are added to the data set.
6. The web interface will provide an mechanism easier than writing PostgreSQL scripts to make queries.
7. All functionality on the site will be easy to find and use, needing no more than familiarity with websites.
8. Any software dependencies used will provide enough external support that the project can be maintained and updated easily even between being handled by multiple, different teams.

Performance Requirements

1. Performance:
    i. The site should load within **1.5 seconds** of a user accessing the website from the Calvin University campus.
    ii. The process from receiving a query to displaying data and visualizations should take no more than **3 seconds**.
    iii. The system should handle the concurrent accesses of up to **100 users** without going above the aforementioned time requirements.
2. Provide **1 page** of documentation on the website designed to help users of the site understand the website and what it provides.
    i. Provide popups as needed across the website.
3. Any function present in the code base should be associated with at least **3 sentences** of comments that outline the function's:
    - parameter(s)
    - return value(s)
    - overall description

The Initial Design



Design Attributes

- Cloud Migratability
  - Public Web Server
    -> Load Balancer
  - Containerized Web Application
    -> Container App or Virtual Machine

Components

- We need a public web server
- And the internal web application
  - This needs a primary framework

Primary Framework

- Whatever primary framework is chosen, we will have to have more supporting libraries
- Possible examples:
  - Linear Algebra library
  - Visualization libraries
  - Database interaction libraries
- This is common in software engineering–software dependencies are plentiful to make a fully fledged application

With that said…

. . . our recommendation:

Why

- It is in Python which has very fast development speeds.
- Can utilize with Python's industry level data visualization and linear algebra libraries (like Plotly and NumPy)
- It gives control over how computed data appear in the HTML through its templating system
    - Displaying the RUVY matrices easier
- Authorization is accomplished by just wrapping functions (views) that need to be hidden
- Dedicated integration with PostgreSQL
- Lots of support

Limitations

- Python implementations are typically slower than others. Django tends to rank lower on sites like TechEmpower

Public Web Server

- A secure connection point at which the clients can access the website
- Actual web application is kept private and communications are kept secured to and from the public web server

Our Recommendation. . .

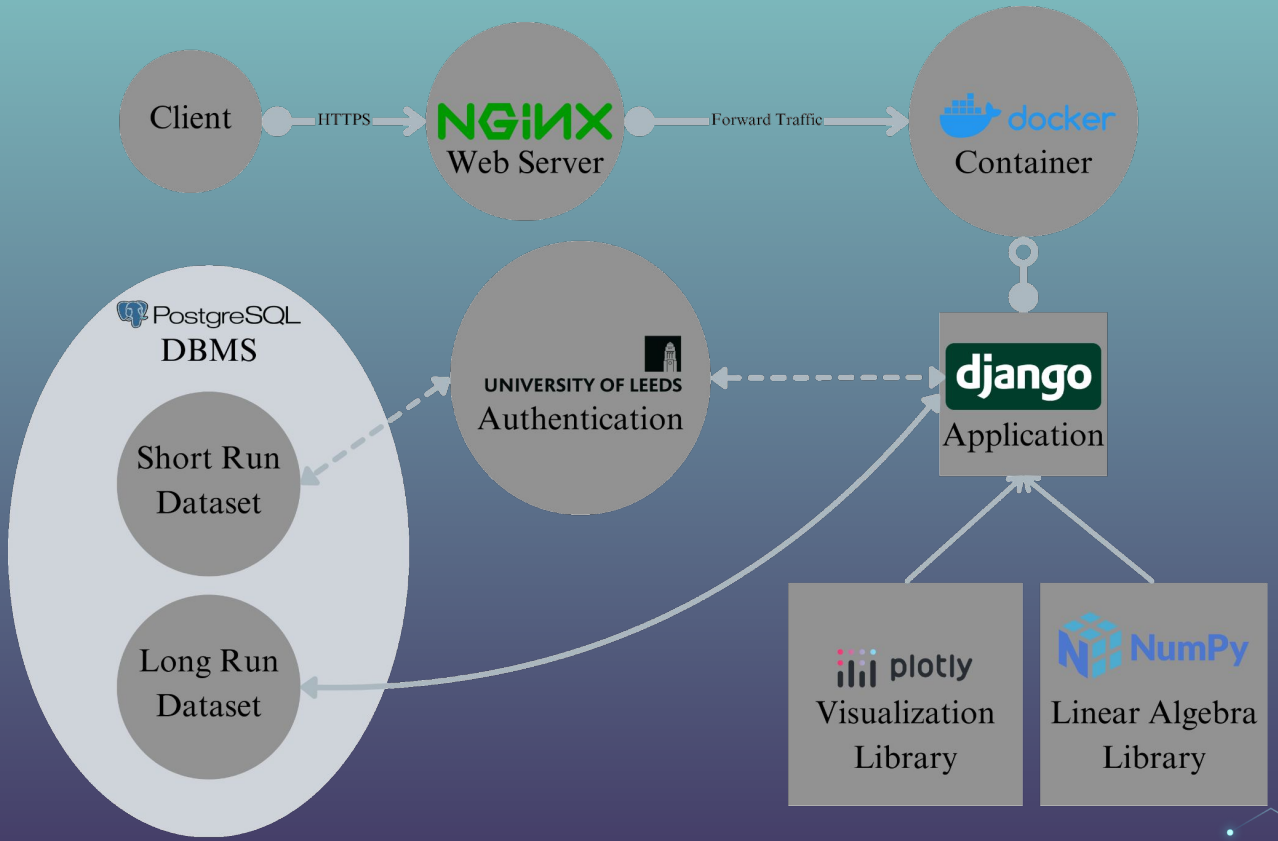
Why

- Offers security and native infiltration protection
- Large community and support
- Specialized in being a reverse proxy (technical term for what we need)
    - Good integration with the Docker, the container software to be used

Limitations

- Little to none for our project

Final Design



Decision Matrix: Primary Framework

Weights	Criteria	Django	Flask	React	Shiny	Actix	Panel	Reflex
0	Cost (>$50, no go)	5	5	5	5	5	5	5
0.3	Community and long-term support (maintainability)	4	4	5	4	3	2	1
0.2	Flexibility of native tools (linear Algebra)	4	4	3	4	3	4	4
0.15	Provides Sankey Diagrams and x-y line plots	4	4	4	4	3	5	3
0.15	Performance (speed of software, concurrent users)	3	3	3	2	5	2	2
0.1	Ease of use and development speed	5	4	3	3	3	3	2
0.05	Provides integrations with PostgreSQL	5	4	3	4	4	4	4
0.05	Provides authentication and authorization integrations	5	4	4	3	4	3	4
1	**Totals**	4.05	3.85	3.8	3.55	3.4	3.1	2.45

Decision Matrix: Public Web Server

Weights	Criteria	Nginx	Apache
0.5	Security	5	5
0.3	Good Community support	5	5
0.2	Easy to proxy to the container	5	4
1	Totals	5	4.8

Primary Frameworks:
Less Desirable Characteristics

- Even though it is simple, as the project gets more complex so does working in Flask.
- Flask does not natively support linear algebra operations.
- It doesn't provide visualization tools directly.
- It is fast, but for high concurrency it might require additional set up.

- It is only a front-end software, so server performance will be more reliant on the internal web server software.
- Integration with database has to be done by the actual web server.
- Linear Algebra libraries are not as good as Python (Django) can provide.

- Although Shiny in R has good documentation and a large community, Shiny in Python was relatively recently released
    - So less documentation.
- Larger, more intensive apps have to be built “in the right way” for good performance, otherwise it will start to slow down.
    - Negative for not thoroughly knowing Shiny already.
- The querying mechanisms for the integration with a database seems to be “clunky”.

ACTIX

- Good, not great community support
- Both of us work better in Python (Django) than Rust
- It seems like it is a lot of work with less opportunity for integration when it comes to authentication

- It must be integrated with a web server, like Flask or Django, for backend processes
    - This can add complexity
- Because it is relatively new there is not a lot for community support
- It relies on Python libraries like NumPy for Linear Algebra
- Authentication support is limited

REFLEX

- Performance is probably not great
  - Python is already a slower option, and Reflex simply wraps React in Python
- With little good documentation, development speed may be significantly hindered
- It is too new with an initial release in 2022
  - There is barely any community support or documentation

Public Web Server:
Less Desirable Characteristics

For our purposes

- Apache and Nginx are almost identical
- Apache is simply less directly specialized in being a reverse proxy and has less integration with Docker

Conclusion

Primary Framework Recommendation:

Public Web Server Recommendation:

Next Steps

- Stress testing helps identify potential bottlenecks and ensures the system can handle expected loads.
- Key performance metrics that will be monitored during the stress testing, such as response time, data visualization capabilities, and seamless PostgreSQL integration.
- If Django does not meet the performance requirements, implement the alternative solutions such as Django with HTMX or React with Node.js

Thank you!

Questions for us?

Appendix A: Web Server Fallback

- While Django is the recommended choice, it's important to consider fallback options in case performance issues arise.
- HTMX can enhance Django's performance by allowing for more dynamic web applications without the need for a full page reload
- Seamless integration between Django and HTMX. HTMX is designed to work well with server-side frameworks like Django,
- If these Python frameworks are not fast enough then we will transition into a javascript framework, such as react over Node.js