Design

Platform Choice

Based on the most likely users of the program there are several factors that could suggest a desktop application would be more suited to this tool. Academics and energy company employees using the program would be interested in repeat use so would be more inclined to download the program if it was deployed as a desktop application, however this would likely dissuade use by students who would use the tool fewer times, or who do not have access to their own computer, thus excluding one of the key target users of the program. While the main aim is to provide real-time forecasting of data for actual energy production, educating students about renewable energy is an important aim in of itself. A desktop application may also benefit academics and energy workers as they may wish to run simulations or add planned solar plants/wind farms, and store them own their own device, doing so may require the use of proprietary data which should not be stored on a public database of simulations. However, again from an educational perspective, the open access to future projects may be useful to students, or to researchers, so a shared repository of information would be to the benefit of all users, provided it does not impede confidentiality of information. This issue could be resolved in a web service by introducing an authentication service allowing users to make the choice of sharing run simulations and added plants with other users, or keeping them private. While the simulation aspect of the project could be run offline, the forecasting will always require an internet connection to run, so there is no trade off in hosting the application as a website against being able to run the program if deployed as a desktop application offline. Future updates would also be much easier to implement with a website than a desktop application and are less likely to affect usability based on user’s system specifications.

Design Constraints for Initial Development

Access to up-to-date data on renewable plants across the UK and access to weather APIs both have significant costs. For the running of this application for demonstrations during the course of the project there were several constraints arising from these costs. Due to the limited access to up-to-date energy production data the, free, dataset used may not be entirely up to date and therefore would not meet the requirements for business use, the cost of accessing this data is in the range of £100-£200 a year, which is not a massive cost. The API usage is slightly more expensive, AerisWeather was used for this project, as it has a free service allowing for 100 free API calls per day, this limited the portion of the full data set that could be used to display the functionality of the software. AerisWeather provides an $80 a month service which allows for 50,000 API calls a day, which would allow for the calling of the API for ~700 full datasets, more than enough to cover all the variables required to map any forecast inputs users have, provided the calls are made once and the data was saved to a database and this data was used for displaying forecasts to users. Alternatively, the system could easily be adapted to use a different forecasting API with better pricing and unlimited calls, requiring changes to just a few lines of code withing the program.

Future Developments to address constraints

* Introduce authentication for public guest users and private users who have access to private storage of simulations and can add future planned projects away from public view
* Introduce an optional paid service for industry and academic users, with greater functionality and customisation, to offset the costs associated with unlimited use of weather API, data storage services, website hosting and up to date records of wind and solar plants

UI Structure and Considerations

The two main functionalities shall be shown on a home screen, allowing for user to select the forecast or simulation home page. Within each of these there will be displays of energy output, either shown by radial graphs as in template visualization and with icons and displays on the map, as well as numerically.

The UI must clearly display the desired data taken from the backend and Weather Data API, both graphically and numerically, as this combination of display will allow for more generalised conclusions to be drawn(suitable for students, or for professionals presenting to untrained colleagues) and for specific precise data to be analysed (for academics and professionals). The UI must, whilst providing detailed statistics, not be so technical that it excludes use by non-technical users. To meet this aim, in addition to the dual methods of data representation, a basic set of variables will be presented to the users on the forecasting and simulation pages, consisting of factors that are understandable by all users, such as wind speed and daylight hours. There will also be a subsequent page for each service that allows for much higher detail to be input into the forecasting and simulations, such as factors such as wind gust and dewpoint as these will allow for more detailed analysis by technical users. Both the Forecast and Simulation home pages shall, respectively, also have a link to a page to load saved Forecasts or Simulations. Both of these home pages shall also be able to access a page to add future or theoretical powerplants to further modify the projections.

Visualisation Principles

Several basic Information Visualisation principles were followed when designing the user interface of the product. Sans serif fonts were used throughout the website, it has been shown that sans-serif fonts can increase readability if the right font is chosen, especially on a monitor, this will aid for accessibility for general users with eyesight conditions, as well as being easier for children and teenagers to use, which will benefit students using the product. For deciding the colour palette used for icons and graphs in the product a mix muted multicoloured tones were used as well as occasional use of bright colour. Bright colours are known to be better at drawing attention to important information, they can cause eye strain when used for long term, therefore brighter colours will be used only to highlight key data points that will not be used for detailed analysis, these bright colours will be useful at representing key information to students just looking to glean the most important point. More muted tones will be used for pieces of information and data representations that require more thorough and repeated reading[[1]](#endnote-1), which will primarily be used by academics or professionals, the wider use of less bright tones will make the overall appearance of the software more professional and be suited to the professional users of the software.

UI Navigation

After several initial drafts, it was decided that the following page hierarchy would be used, by having most pages able to be accessed directly from eachother navigability of the website shall be easy for any level of user.

Diagram

Description automatically generated

Figure :UI Navigation

Below are shown template UI visualisations for the home and forecasting pages, the simulation home page mirrors the forecasting page with only slight changes in settings. Other pages will have a similar structure to these, with a navbar and graphics where applicable. On each of the home screens for the forecasting and simulation a link will be used to access the advanced settings allowing for more detailed specifications.

The UI for the addition of plants and farms will be done in a simple form with text boxes and menus to enter specifications about the theoretical plant, the data input shall then be formatted and passed to the back end to be saved in the appropriate format. Likewise, the advanced settings forms will be simple, using a similar combination of text buttons and check boxes as the home pages to provide specifications for forecast and simulation.

A picture containing graphical user interface

Description automatically generated

Figure : Home Page

Diagram

Description automatically generated

Figure : Forecasting Page

System Architecture

It is common in web development to separate the server and the client when developing the full stack of a web application, as the server will not heavily influence the front end this is a useful structure as the main connections between client and server will be for passing data to the front end from the back end or via the back end from the database. The separation of the server shall be done by creating a RESTful API. One reason for this design choice is that it allows for future development to make use of cloud storage services, vastly increasing the database capacity used for storing simulations if user base becomes large, or if the application were to be expanded beyond forecasting just for the UK into a global service[[2]](#endnote-2). In addition to this, as renewable sources become more prevalent a much larger data storage capacity will be required for tracking all of these power production sites

The database is used for storage of information on renewable energy productions sites, which are fed into the program for calculating, in tangent with data from passed in from a weather API, energy production statistics. The database is also used to store solar plants and wind farms added by users, as well as saving simulations created by users.

System Technologies

A collection of web design technologies were used for the development of the product. A hybrid of commonly used stacks was used for the development, taking the most widely used elements from stacks to create one well suited to coding the back end in Java. Taking influence from MEAN and the typical python Django stack, the final stack used consisted of using a combination of Angular and node.js in Microsoft Visual Studio for the front end, and using Spring Boot in IntelliJ for the back end. MySQL was chosen for the database management; this will be discussed further in the database section of the design chapter.

As opposed to the typical use of JavaScript for the front end, TypeScript was used, this is because Typescript has several properties that make it easier to use. TypeScript sets out code in a clearer way than JavaScript, allowing for easier debugging (which can be useful when producing a product on a short time scale) and allows for static typing which is complementary when using Java for the back end as Java is a static typed language. The lack of static typing in JavaScript can actually speed up the process of writing code, which can be a benefit for small projects, however looking forward at the possibility of expanding this project TypeScript has more potential for scalability. TypeScript is also open source and rapidly increasing in popularity meaning that there are plenty of easily accessible resources to help with development[[3]](#endnote-3),[[4]](#endnote-4).

Several other tools were used to aid in implementation and testing of the program, such as Postman for testing the server API and Bootstrap for CSS styling. Version control was managed using GitHub, links to the complete repositories for the frontend and backend can be found in appendix III.

System Structure

*Diagram

Description automatically generated*

Database Considerations and Finalised Structure

Considerations

The initial decision faced was whether to go down the route of using a non-relational database, such as MongoDB, CouchDB or Cassandra, as opposed to using a relational database. While NoSQL databases have the benefits of simpler design and faster query speed for simple queries, there can be issues with inconsistent data retrieval, this is an incredibly significant point for this application, if there are issues retrieving accurate data, forecasts and simulations could be massively inaccurate, reducing the validity of using the application and potentially affecting user numbers. However, it has been shown that in certain circumstances SQL databases can be comparably fast to NoSQL databases, especially for more complex queries[[5]](#endnote-5),[[6]](#endnote-6). As NoSQL has a much smaller user base, it would mean that if the product were to be brought to market and team created to manage the application it would be harder to find team-members with sufficient knowledge in NoSQL database management as well as all of the other skills needed to develop the application[[7]](#endnote-7).

It was therefore decided that an SQL relational database would be used for the database of the program. Three choices were considered, PostgreSQL, MySQL and SQLite. SQLite has one major benefit for its use in small scale projects over other SQL models, it is embedded in the program which makes access to data far simpler. However, while this would be suitable for the current development cycle, it would massively impede scalability of the application, therefore SQLite was disregarded as a valid option. Differences between MySQL and PostgreSQL are minimal in terms of performance, and while PostgreSQL has additional functions and arguably better scalability, there is again a much smaller community using it than MySQL, meaning documentation and third-party services and tools are less ubiquitous[[8]](#endnote-8). Either MySQL or PostgreSQL would be a suitable option for this application, but ultimately MySQL was chosen.

Database Structure

Below is shown the overview of the finalised database structure, the full database ER diagram is shown in Appendix II. Below is a simplified version of the diagram.

Diagram

Description automatically generated

Security

The primary security feature used is the encryption of passwords for users of the system. For the initial development SHA-512 the passwords. SHA-512 was chosen as it is a very secure encryption method with regards to brute force attacks. For the initial release using SHA-512 is certainly sufficient as in the initial stages the login and registration doesn’t affect user access to the service so is not a required feature, but by registering, users could be informed of updates and new releases regarding the program if they did register at this stage, as well as allowing for user interest to be measured[[9]](#endnote-9). Potentially this could be changed to a more secure algorithm if required.

In future development stages further security measures would need to be implemented. The most significant change would be to separate the storage of user information from the main database and store it in its own database. This would be a relatively easy task as the once logged in a simple the checking of privately stored simulations etc. would only require checking the users name and user type against the simulation author name for example for access, which could be done with a temporary user id token, meaning that the primary database never needs to interact with the user database. Separating the databases would therefore prevent the possibility of injection attacks.

Further measures would have to be discussed if a paying service for premium access was implemented in future development cycles.

1. Byron, L. and Wattenberg, M. (2008). Stacked Graphs – Geometry & Aesthetics. *IEEE Transactions on Visualization and Computer Graphics*, 14(6), pp.1245–1252. [↑](#endnote-ref-1)
2. https://searchapparchitecture.techtarget.com/definition/RESTful-API [↑](#endnote-ref-2)
3. https://dzone.com/articles/what-is-typescript-and-why-use-it [↑](#endnote-ref-3)
4. https://serokell.io/blog/why-typescript [↑](#endnote-ref-4)
5. Győrödi, C.A.; Dumşe-Burescu, D.V.; Zmaranda, D.R.; Győrödi, R.Ş.; Gabor, G.A.; Pecherle, G.D. Performance Analysis of NoSQL and Relational Databases with CouchDB and MySQL for Application’s Data Storage. Appl. Sci. **2020**, 10, 8524. https://doi.org/10.3390/app10238524 [↑](#endnote-ref-5)
6. Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), 2013, pp. 15-19, doi: 10.1109/PACRIM.2013.6625441. [↑](#endnote-ref-6)
7. https://www.datastax.com/blog/sql-vs-nosql-pros-cons [↑](#endnote-ref-7)
8. https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres [↑](#endnote-ref-8)
9. https://www.researchgate.net/profile/Imam-Riadi-2/publication/327392778\_Analysis\_of\_Secure\_Hash\_Algorithm\_SHA\_512\_for\_Encryption\_Process\_on\_Web\_Based\_Application/links/5b8cbe5e4585151fd1447946/Analysis-of-Secure-Hash-Algorithm-SHA-512-for-Encryption-Process-on-Web-Based-Application.pdf [↑](#endnote-ref-9)