Coursework 2

1 Requirements

1.1 Explanation of the choice of techniques

The first step in generating requirements was to understand the context of the problem being solved. The product I will be creating is a data visualisation dashboard for Higher Education Institution (HEI) data for use by students, staff, and researchers. Based on the problem overview, I decided to create a context diagram to provide a high-level overview of the dashboard and its interactions with external entities to quickly get the scope and boundaries. The context diagram created can be seen in Figure 1.

With the context diagram created, I considered which one of the BABOK techniques would be most suitable to elicit the requirements, given the restraints set for this coursework. With this I decided that brainstorming and document analysis would be the most suitable techniques for requirement elicitation. Brainstorming allowed me to be creative to generate multiple ideas in a short amount of time. Brainstorming was also chosen as it means I will not get consumed by detailed analysis at such an early stage. I also decided to analysis existing and comparable solutions to further elicit requirements. By having a visual of a working environmental data dashboard, I will be able to confirm the necessity of the requirements already elicited whilst developing new ones I may not have already thought of.

To document the elicited requirements, I will use User stories. The key reason for using user stories is the fact that they foster further thinking about the functionality of the product, particularly from the perspective of the user. This allows me to focus on features that will provide the most value to the user. The simplicity of use stories also makes them easy to write, understand and later prioritise. The final reason I decided to use user stories to document my requirements is the flexibility they allow. Because user stories are brief and simple, they can be easily changed and reprioritised as a response to changes, adaptations, feedback, and evolving priorities. Because my users will all have a similar set of goals whether they are staff, students or researchers, the user in my user stories will always be users. As the creation of my web app continues, they may be some new actors that are considered or there may some features that I realise are more useful for, e.g., staff but for now ‘user’ will be my only actors.

To prioritise, the requirements, I will use a combination of MoSCoW and forced pair ranking. I decided to use MoSCoW as it allows me to identify the critical features of the dashboard that must be included. The ‘Should have’ and ‘Could have’ categories also allow me to approach the prioritisation of non-critical requirements in a balanced way. Forced pair ranking will then be used to confirm that I have placed the requirements in the correct categories but also prioritise the requirements within each category. I will relatively compare the requirements and will end up with a list of prioritised requirements in order of criticality to the user. Both techniques are straightforward and easily adaptable, making them a suitable choice.

1.2 Prioritised requirements

User story	Acceptance criteria	MoSCoW category	Priority within MoSCoW category
As a user, I want to able to rank HEIs in tabular format based on environmental metrics so that I can understand their relative performance.	a. A table should be provided, showing institutions ranked based on chosen metrics. b. The ability to switch between ascending and descending order of rankings should be available. c. Ties in the rankings should be handled and displayed appropriately. d. The ability to only show HEIs in one region in the ranking table should be given	Must Have	M1
As a user, I want to be given an overview of each HEI's environmental metrics	a. A menu should be available for choosing a specific HEI of interest. b. The overview page should contain some key statistics for that HEI c. Charts of the selected HEI's environmental data over time should be shown d. The ability to filter chart categories based on preferences should be provided.	Must Have	M2
As a user, I want to focus on one environmental metric so that I can compare HEIs in that specific area.	a. A page allowing me to choose metrics should be available. b. Choosing a metric should display a bar chart showing data for all HEIs for a specific year. c. Data filtering options by region and year should be accessible.	Must Have	M3
As a user, I want to be able to customise the data visualisation graphs created to compare environmental metrics specific to my needs	a. The dashboard should allow the creation of a graph with a selected group of HEIs and metrics of my choice.	Must Have	M4
As a user, I want to visualize Higher Education Institutions (HEIs) on a map to gain a geographical overview and analyse institutions within the same region	a. Upon entering the map page, a map of England with different regions should be displayed. b. Clicking on a region should provide a zoomed-in view with points representing HEIs within that region. c. Hovering over each point should reveal a dialog box with key information about the respective HEI.	Should Have	S1
As a user, I want to dashboard to respond quickly to my interactions so that I	a. Clicking on a metric for data visualization should render the updated chart within 5 seconds.	Should Have	S2

can efficiently analyse environmental data.	b. The initial page load time for accessing the dashboard should be under 10 seconds. c. The system should maintain acceptable response times during simultaneous access by multiple users are accessing the dashboard simultaneously.		
As a user, I want the graphs on the dashboard to be interactive so that I can quickly view more information than what is initially provided	a. Hovering over sections or points within graphs should provide additional information. b. Clicking on a graph should enlarge it on the screen.	Should Have	S3
As a user, I want to be able to use the dashboard on a desktop or laptop with different web browsers so that I am not constrained to only specific requirements to use the dashboard	a. The dashboard should be compatible with major web browsers, including Chrome, Safari, Edge, and Opera. b. The dashboard should have a maximum width of 1440px. c. The dashboard should be suitable for landscape viewing.	Could Have	C1
As a user, I want to be able to export and download selected charts and data so that I can carry out further analysis and use them in my reports/presentations	a. An option should be available to select charts for download. b. Charts should be exportable as PNGs within a zip file or a PDF file. c. An option should be available to select tables for export. d. Tables should be exportable as CSV or XLS. e. I should be able to login and view my selected charts from my previous session	Could Have	C2
As a user, I want the dashboard and graphs created to be highly customisable so that I can generate graphs that meet my specifications for presentations and/or reports	a. Font size and font colour within the charts should be changeable. b. The colour scheme/gradients used within the charts, or a group of charts should be changeable.	Could Have	C3

Requirement C2 is quite a complex one so I will analyse it further using use case modelling. The following use cases have been created using requirement C2:

Use case	Login
ID	UC1
Brief description	User logs in to their account

Primary Actor(s)	Registered user
Secondary Actor(s)	None
Preconditions	1. The user has an account stored in the database 2. The user has a valid email and password
Main Flow	1. User navigates to login page 2. Page presents login form with fields for email and password 3. User enters credentials 4. System and database validate credentials 5. If credentials are valid, system logs user in 6. Export list page is displayed, showing previously saved charts for export, if any
Postconditions	1. User is successfully logged in 2. Export list page is displayed
Alternative Flows	If user enters invalid credentials: 1. Error message is displayed 2. User can retry entering correct credentials or reset password

Use case	Create custom graph
ID	UC2
Brief description	User can create custom visualisation graphs and tables
Primary Actor(s)	New user Registered user Logged-in user
Secondary Actor(s)	RESTAPI
Preconditions	None
Main Flow	1. User navigates to custom graph page 2. User selects some filter variables for data and type of graph to be created 3. System requests data from RESTAPI 4. RESTAPI responds with data 5. System allows the creation of a customised graph
Postconditions	Custom graph is displayed on the screen
Alternative Flows	None

Use case	Add graph to export list (extends UC2)
ID	UC3
Brief description	Allow a registered user who is logged in to add a custom graph to their list of graphs for export
Primary Actor(s)	Logged-In User
Secondary Actor(s)	None
Preconditions	1. The user is logged into their account 2. The user has created a custom graph (UC2)
Main Flow	1. User creates a custom graph 2. User decides to add the custom graph to their export list 3. User selects the option to add the graph to the export list 4. System adds the custom graph to the user's export list 5. The graph added to the custom list as saved to the database and associated with the user's account
Postconditions	The custom graph is successfully added to the user's list of graphs for export.
Alternative Flows	If the user is not logged in:

	1. System takes user to the login page 2. After successful login, the user can proceed with adding the graph to the export list If there are technical issues or errors during the adding or saving process: 1. Error message is displayed 2. User is informed about the issue and can retry

Use case	View graphs in export list (extends UC3)
ID	UC4
Brief description	Allow a logged in user to view the graphs in their export list
Primary Actor(s)	Logged-In User
Secondary Actor(s)	None
Preconditions	1. The user is logged into their account 2. The user has added at least one custom graph to their export list (UC3)
Main Flow	1. User selects the option to view their export list 2. System retrieves and displays a list of graphs the user has added for export 3. User can select a specific graph from the list to view details or remove from the export list
Postconditions	The user successfully views the list of graphs in export list
Alternative Flows	If user's export list is empty 1. System informs user that their export list is currently empty

Use case	Export graphs in export list (extends UC4)
ID	UC5
Brief description	Allow a logged-in user to export the charts in their export list for download, specifying the desired file type
Primary Actor(s)	Logged-In User
Secondary Actor(s)	None
Preconditions	1. The user is logged into their account 2. The user has added at least one custom graph to their export list (UC3) 3. The user has viewed the graphs in their export list (UC4)
Main Flow	1. User selects the option to export charts 2. System presents a form allowing the user to specify the file type for download 3. User selects the desired file type for download 4. System processes the request and generates the export file in the specified format 5. Export file is downloaded to user's local system
Postconditions	The user successfully exports the charts from their export list in the specified file type
Alternative Flows	None

Use case	Register account
ID	UC6
Brief description	Allow a user to register an account
Primary Actor(s)	New User
Secondary Actor(s)	None

Preconditions	1. The user has accessed the webapp and wishes to register 2. The user has a valid email address
Main Flow	1. User clicks the button to register an account 2. System presents a registration form with fields for email and password 3. User enters the required information 4. System validates the entered information, ensuring the email is unique 5. User submits the registration form 6. System creates a new account for the user and associated it with the entered information
Postconditions	The user's account is successfully registered
Alternative Flows	If the email is not unique: 1. System prompts user to enter a different email 2. User enters a new, unique email 3. System continues with registration process

The use case diagram for the use cases shown above can be seen below:

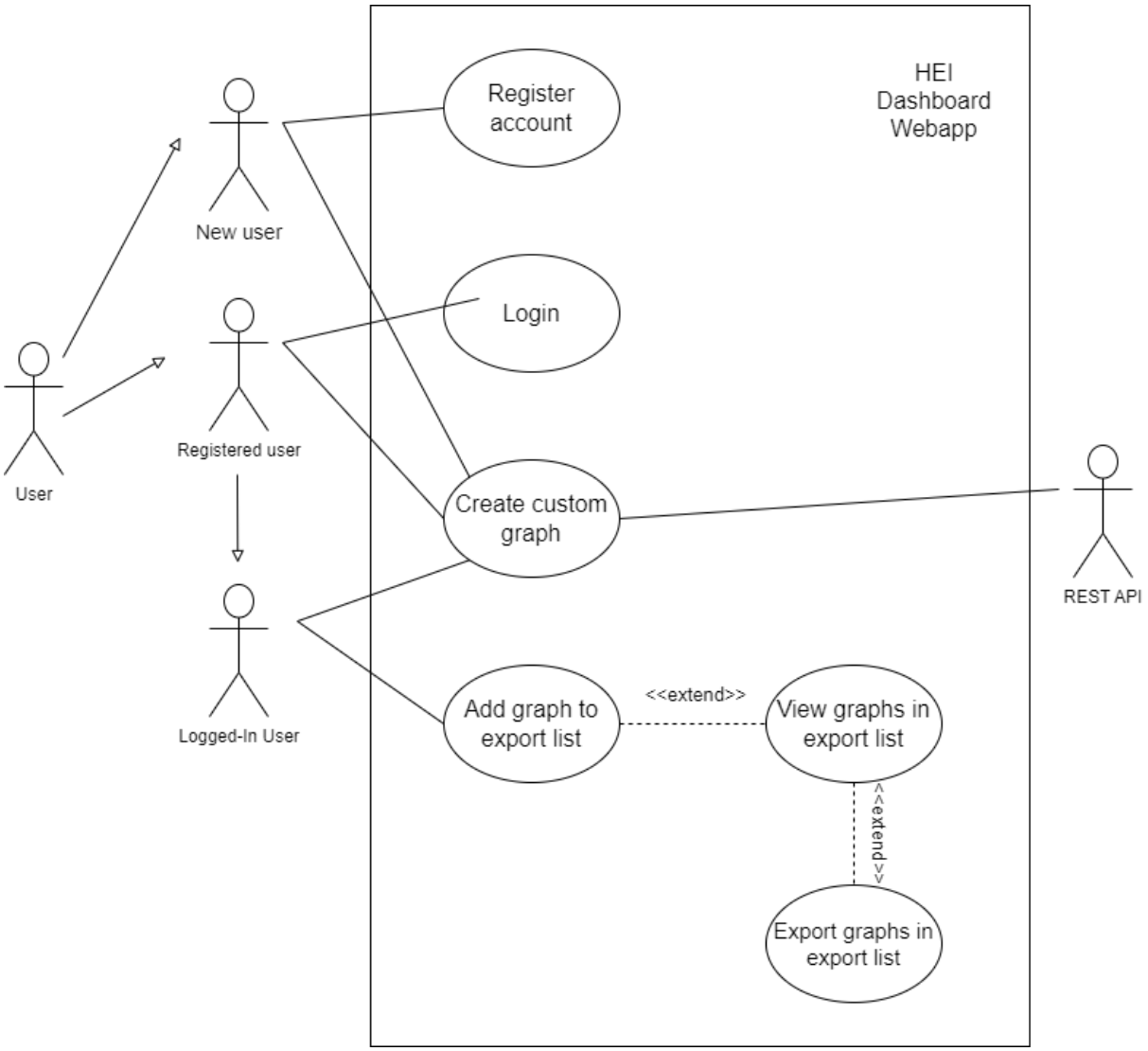
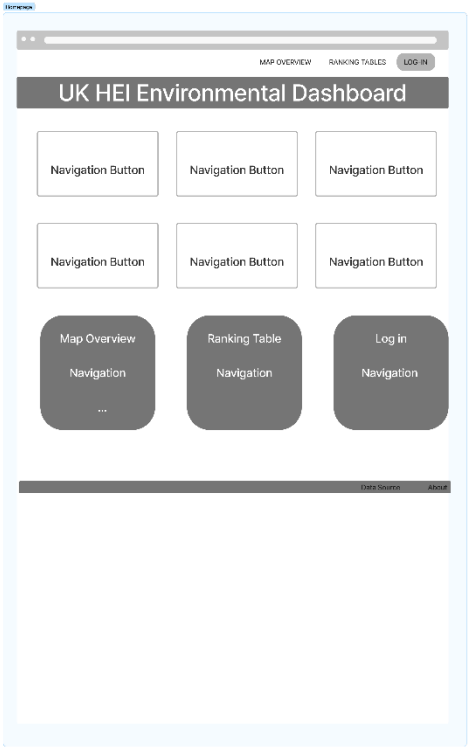
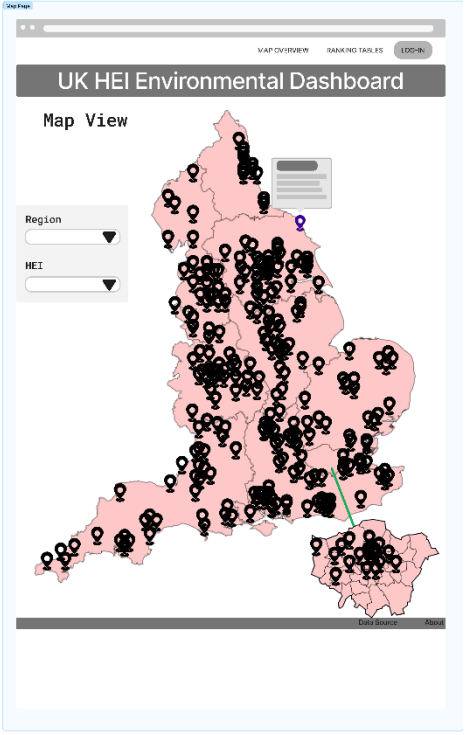


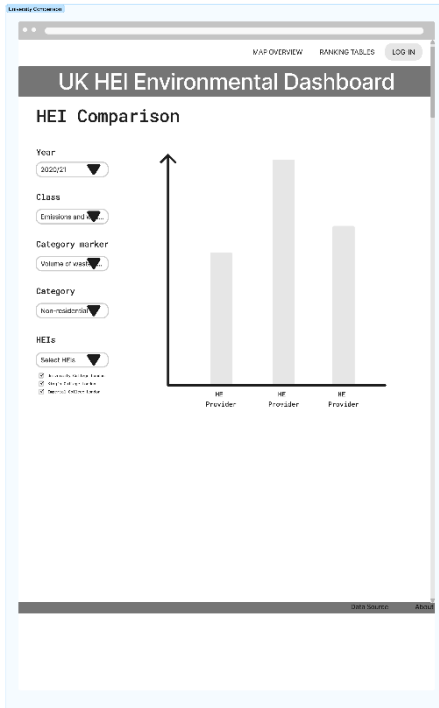
Figure 1 Use case UML diagram exploring requirement C2

2 Design

2.1 Interface design







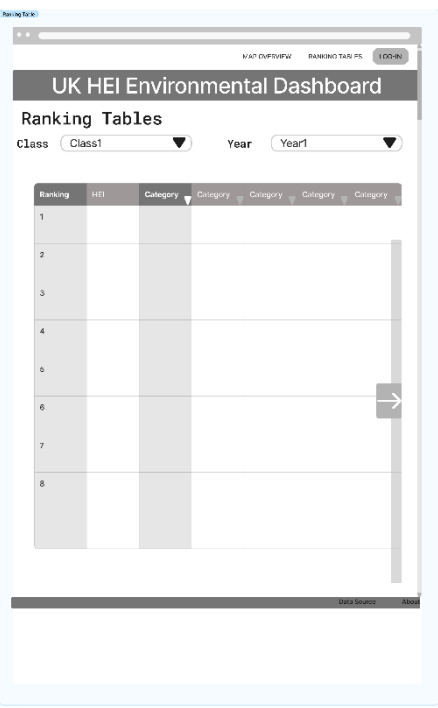


Figure 2 Wireframes showing layout for each page of the HEI Dashboard web app.

2.2 Application design

Using the requirements and wireframes, Figure 3 was created to represent the application design.

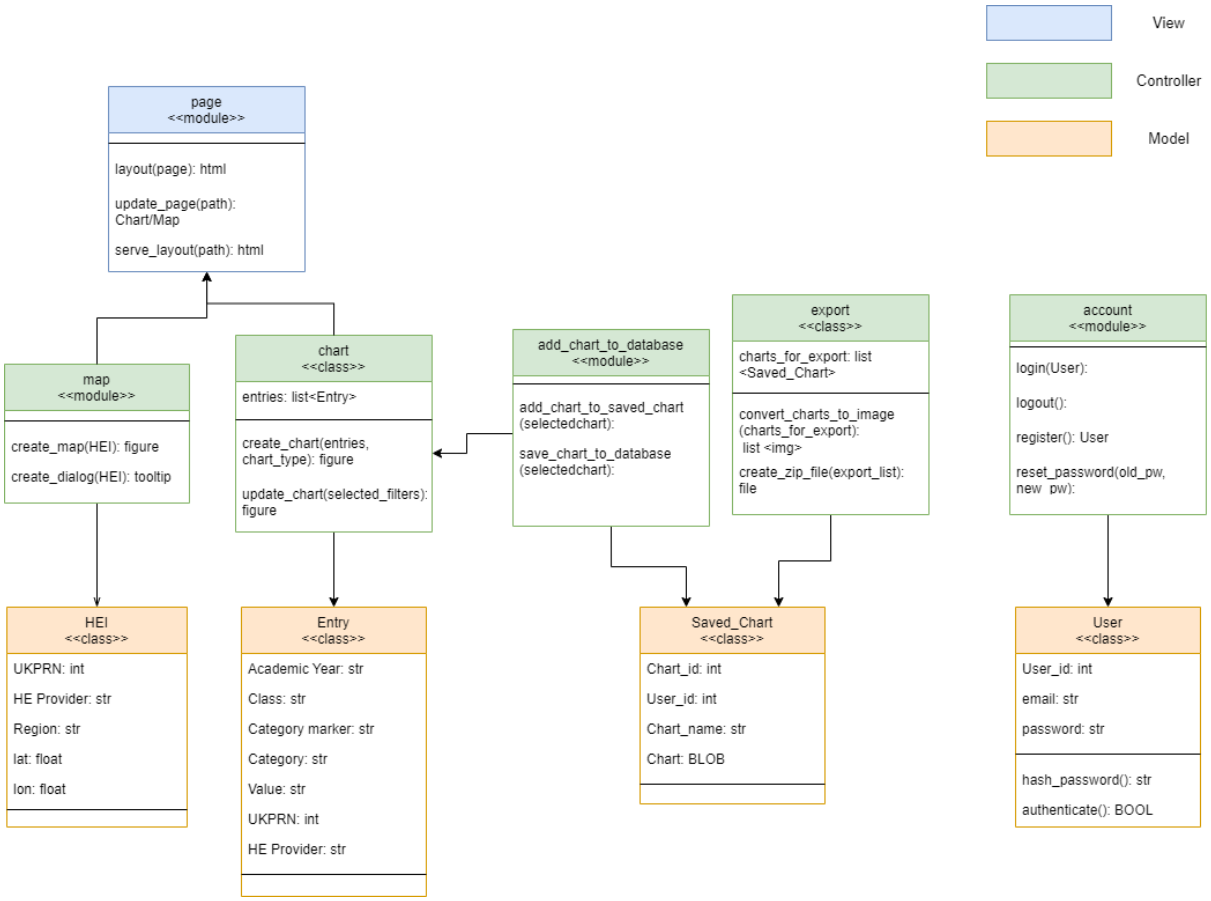


Figure 3 Class diagram implementing Model-View-Controller design pattern for the design of the HEI dashboard.

The application design incorporates key principles such as modularization, high cohesion, and loose coupling to enhance code maintainability and reusability. These principles were prioritized to streamline the application's structure, making it easier to maintain, update, and adapt for various elements and pages of the web app.

The initial step involved dividing the prepared dataset into separate classes, namely HEI and Entry.

The next step taken in the application design was to brainstorm a list of the functions required as part of the web app (Table 1).

Table 1 The functions created as part of application design and a brief description of each

Function	Description
create_map(HEI)	Generates geographical map displaying HEI and provides a visual representation of their location
create_dialog(HEI)	Creates a tooltip for each HEI, giving more information on each of them
create_chart(entries, chart_type)	Given a list of data entries, a chart of specific type is generated and displayed
update_chart (selected_filters)	Updates chart based on filters selected by user, refining displayed data according to preferences

add_chart_to_saved_chart (selectedchart)	Adds a chart to a user's list of saved charts, providing a personal repository
save_chart_to_database (selectedchart)	Adds a selected chart to the database, so that users access saved chart across sessions
convert_charts_to_image (charts_for_export)	Convers a list of charts to image format
create_zip_file (export_list)	Creates ZIP files containing charts from export list
login(User)	Authenticates a user and logs them into the system
logout()	Logs current user of the system
register()	Initiates the user registration process enabling new user to create accounts
reset_password(old_pw, new_pw)	Facilitates the process of resetting a user's password from an old one to a new one

A pivotal consideration in the application design was accommodating a feature allowing users to select and export generated charts. To facilitate this, the application creates and saves charts in the database, associating them with the respective user. As elements of the design took shape, the Model-View-Controller (MVC) design pattern was chosen, aligning with the focus on modularization, reusability, and maintainability. While acknowledging that MVC introduces complexity, its use was justified by anticipating the application's potential growth in complexity over subsequent releases.

The application design is visually represented as a class diagram, offering a clear depiction of the separation and organization of model, view, and controller classes. The diagram also aids in understanding the level of coupling through arrows indicating relations and dependencies. However, it is acknowledged that a class diagram may oversimplify the MVC pattern, potentially missing dynamic interactions between classes, especially in the absence of code implementation.

The View in the MVC diagram corresponds to each page seen in the interface design. The Models is the data stored in the database. The Controllers are the classes and modules that carry out the key functionalities within the web app including visualisation of charts, map, exporting, and account creation and usage.

Upon reflection, the class diagram is seen as a starting point for the web application, acknowledging that the actual implementation may reveal additional functions required for optimal functionality. Despite the likelihood of complexities not fully captured in the diagram, it serves as a valuable roadmap for creating a well-organized and scalable web application.

2.3 Database design

The logical design of the database to be used in both apps is found in the entity relationship diagram shown in Figure 4.

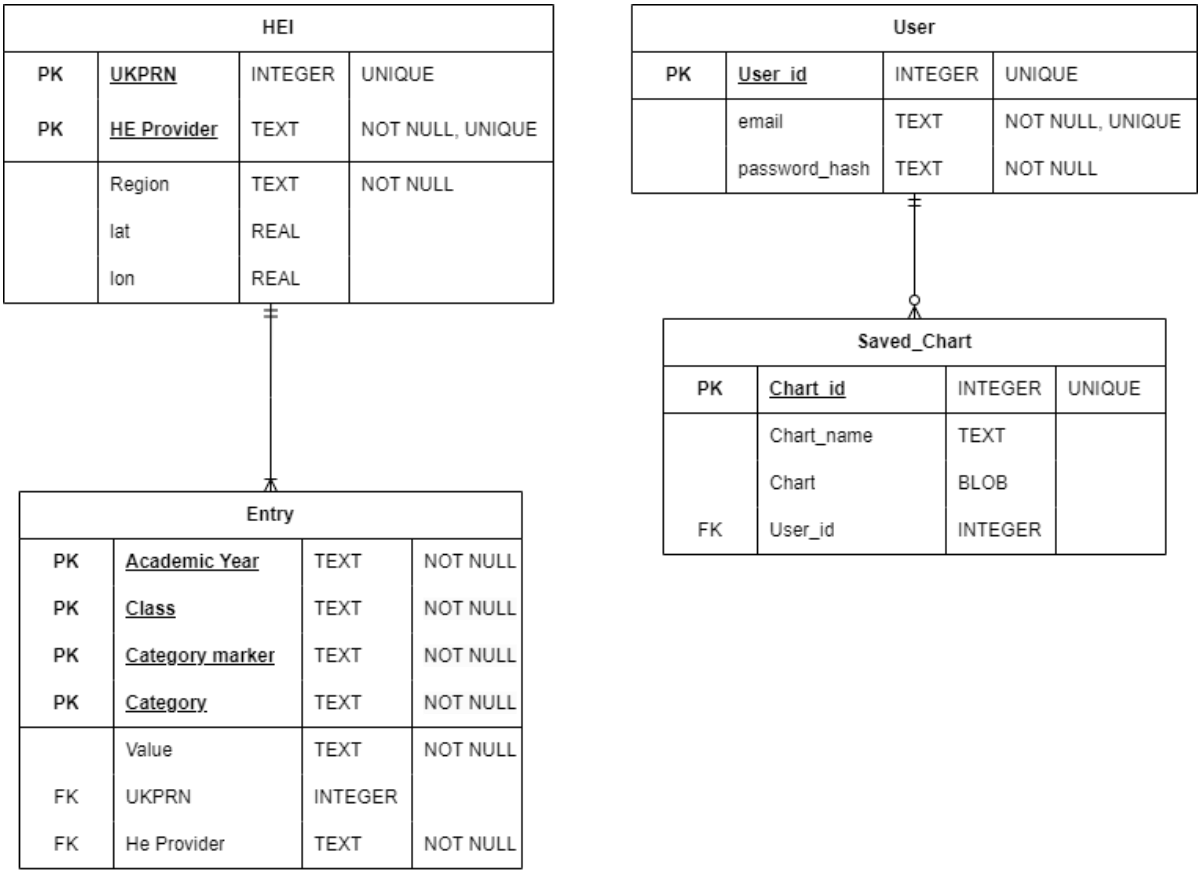


Figure 4 Entity relationship diagram for the database behind the HEI environmental dashboard app. Entities HEI and Entry are from the dataset whilst User and Saved_Chart are created by the webapp. The database design achieves 3NF.

In the database creation process, several crucial decisions were made, starting with the structuring of the dataset. The web app's requirements and wireframes indicated the necessity of an HEI (Higher Education Institution) entity. For this entity, the attributes identified were UKPRN, HE Provider (also the name of the HE), and the Region of the HE Provider. Recognizing UKPRN as a unique identifier for each HEI, it was chosen as the primary key (PK). However, to address transitive dependencies where Region is also dependent on HE Provider (a non-key attribute), a composite PK comprising both UKPRN and HE Provider was implemented. Although composite primary keys can be challenging in SQLite, the decision was made to avoid adding a new identifier column, considering the uniqueness of UKPRN.

During the initial database design, the Entry entity was initially divided into two entities: Metric (Class, Category marker, Category) and Entry (Academic Year, Value). However, this approach raised two issues. Firstly, the Metric entity had transitive dependencies, and secondly, it was later deemed redundant due to the absence of scenarios exclusively using Metric information. The Entry entity, with a composite PK of four attributes, presented potential problems. Creating a new Entry_id as a single PK would introduce transitive functional dependencies (Category -> Category marker and Category marker -> Class), necessitating the splitting of the Entry entity into additional tables. To minimize complexity, the decision was made to retain the composite PK.

Regarding the Saved_Chart entity, initially conceived as a chart entity for every user, a reconsideration prompted a shift in approach. The revised design involved saving only the charts explicitly saved by users, those added to the export list. This adjustment streamlined the database by avoiding unnecessary information and conserving storage space. The rationale was to store charts associated with users, facilitating easy retrieval when users log in for subsequent sessions, aligning with a more efficient and focused database structure.

Given that:

1. All attributes have a single value
2. Where there is a composite PK, each non-key attribute is full dependent on the entire PK, not a subset of the PK
3. No non-key attribute is transitively dependent on any key and all non-key attributes are dependent on the PK only,

the data is in 3NF form.