**Task 1A: Scenario and Scope**

The load shedding epidemic in South Africa has flung the country into a full-scale energy crisis, and the average person is being greatly affected by it. The government has been encouraging people to conserve electricity, which is not an easy task since almost all of our appliances and devices require electricity to operate properly. This crisis has highlighted the need for individuals to be more mindful of their energy usage to reduce their impact on the environment and save money on their electricity bills. However, many people lack the tools and knowledge to accurately track their energy consumption and identify areas where they can make changes to reduce their energy usage.

The solution to this problem is ***Powerhouse***. Powerhouse is an electricity calculator application that allows users to input the type of appliance they are using and the amount of time they are using it to accurately calculate their energy consumption and cost and provide them with a deeper understanding of how their electricity costs work. The application will also provide users with tips and tricks for conserving energy and reducing their carbon footprint. By helping users to track their energy usage and providing them with the knowledge and resources to make informed decisions about their energy consumption, Powerhouse aims to empower individuals to reduce their impact on the environment, save money on their electricity bills, and minimize the potential damage caused to their appliances as a result of load shedding.

**Task 1B: User Requirements**

The Powerhouse application will primarily target homeowners. The average homeowner in South Africa cannot afford to install solar panels or purchase inverters. Therefore, Powerhouse will cater to these individuals by helping them to save money on electricity and prolong the lifespan of their appliances as much as possible. Homeowners should know how much power their electrical appliances use, the implications of using these appliances for prolonged periods, and how to reduce their electrical expenses.

During the development of Powerhouse, I interviewed several individuals to determine what features they would like in an application of this nature. The responses I received included:

* "I want to be able to set a budget for my energy usage so that I can stay within my financial means."
* "As a user, I want to receive tips and tricks for conserving energy so that I can reduce my carbon footprint and save money on my electricity bill."
* "I would like to learn more about how my electrical appliances work and the dos and don'ts of
* operating these appliances during load shedding.”

For security reasons, user passwords will not be stored as plain text in the backend database. Passwords will therefore be salted with a dynamic salt, and then hashed when being stored in the database (signup), and when being compared to the database (logins).

Example:   
Password: 12345678  
Salted Hash: a2240212f6e8d7db337b46aab21433ad==

This makes it impossible for passwords to be leaked in the event that the database is compromised.

**Task 2: Design the Database**

**tblAppliances:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Example** | **Constraints** |
| ApplianceID | AutoNumber | 21 | Primary Key |
| ApplianceName | Short Text | Samsung 303L Fridge | Unique, Not Null |
| Wattage | Number | 400 | Not Null |
| InputCurrent | Number | 200 | Not Null |
| InputVoltage | Number | 50 | Not Null |
| CostPerHour | Currency | R0.50 | Not Null |

**tblTips:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Example** | **Constraints** |
| TipID | AutoNumber | 5 | Primary Key |
| ApplianceID | Number | 21 | Foreign Key |
| TipDescription | Long Text | Turning off appliances before power outages can save them from incurring long-term damage | Not Null |

**tblUsers:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Example** | **Constraints** |
| UserGUID | Short Text | 0FF54C0B80D24CB6B443C1C9BE95EC29 | Primary Key |
| Username | Short Text | IntRicate | Unique, Not Null |
| EmailAddress | Short Text | adam.foflonker@gmail.com | Unique, Not Null |
| Forenames | Short Text | Adam | Not Null |
| Surname | Short Text | Foflonker | Not Null |
| PasswordHash | Short Text | a2240212f6e8d7db337b46aab21433ad== | Not Null |

**// TODO: Screenshot relationships from Access**

// tblAppliances will be the application’s central database. I.E, all appliances ever used by the users of the application will be stored here. A JSON file will then store the ApplianceID’s of all appliances that the current user account uses. These ID’s can be loaded into RAM as an array of integers.

// This means that the application will then feature a login page and that multiple users will be able to login to the app. User login credentials will be stored in the database as tblUsers.

// A user GUID exists to internally identify users and their activities by their GUID instead of username. This means that a username can be changed so long as the username remains unique. (add this info to user reqs)

// Because user GUID’s are unknown to users, it also helps in deterring users from tampering with the JSON file, because they’ll be modifying things but they won’t know who it’s for

**Task 3: Data Dictionary**

The application will make use of a JSON file “PowerhouseSave.json” to store the current state of the application including its’ active users and all the appliances associated with their account. Upon startup, the application will load all the save data from the JSON file into memory and then wait for the user to login to their account. Once the user logs in, the application will then get all of the ApplianceID’s that that user has attached to their name and retrieve the information for each of those appliances from the database “PowerhouseDb.mdb.” Multiple arrays will be required to efficiently carry out this process.

Sample text from PowerhouseSave.json:

{

    "type": "PhUsers\_u.PhUsers",

    "id": 1,

    "fields": {

        "Users": [

            {

                "type": "PhUsers\_u.PhUser",

                "id": 2,

                "fields": {

                    "m\_GUID": "0FF54C0B80D24CB6B443C1C9BE95EC29",

                    "m\_Appliances": [

                        1,

                        5,

                        21,

                        23,

                        30

                    ]

                }

            }

        ]

    }

}

Because a single user can have a large number of appliances and that there can be a potentially large number of users as well; certain getter functions in the ‘**PhUser**’ class will return variables by reference instead of by value using the ‘**out**’ keyword.

This architectural decision was made because these variables could potentially get way too large to efficiently pass them by value (creating copies of them when reassigning to new variables) which will bloat the memory used by the program. Additionally, this also assists in ensuring that different parts of the program will be able to properly modify the original instances of these variables instead of copies of them.

|  |  |  |
| --- | --- | --- |
| **Class Name** | **Attributes/Fields** | **Methods** |
| **PhAppliance** | **private** m\_ID: uint32;  m\_Name: string;  m\_Wattage: real;  m\_InputCurrent: real;  m\_InputVoltage: real;  m\_CostPerHour: real; | **public**  GetID(): integer;  GetName(): string;  GetWattage(): real;  GetInputCurrent(): real;  GetInputVoltage(): real;  CalculateCostPerHour(): real;  **private**  UpdateInDatabase(); |
| **PhUser** | **private**  m\_GUID: string;  m\_Username: string;  m\_PasswordHash: string;  m\_Forenames: string;  m\_Surname: string;  m\_Appliances: TArray<TAppliance>; | **public**  GetGUID(): string;  GetUsername(): string;  SetUsername(usrName: string);  GetForenames(): string;  SetForenames(names: string);  GetSurname(): string;  SetSurname(surname: string);  GetAppliances(out result: TArray<TAppliance>);  GetApplianceByName(name: string; out result: TAppliance);  GetApplianceByID(id: integer; out result: TAppliance);  **private**  UpdateInDatabase(); |
| **PhUsers** | **public**  Users: TArray<PhUser>; | **public**  constructor Create(); |
| **TJsonSerializer** | **private**  m\_Marshal: TJSONMarshal;  m\_Unmarshal: TJSONUnMarshal; | **public**  constructor Create();  SerializeJson(obj: TObject): string;  DeserializeJson(const jsonStr: string): TObject; |

Logical flow: When the program starts up, ALL users currently in JSON will be loaded into memory. After a user is logged in: it will ONLY load appliances into memory from the database that are currently in use by the current user who’s been logged in.

**//Prefix all powerhouse classes and units with ‘Ph’**

**// Add ctors and dtors**

**// Add an OTP email if forgot password?**

**What can be added to my app?**

1. Gamification: Consider adding a gamification element to your application to incentivize users to reduce their energy usage. For example, you could award points or badges to users who consistently reduce their energy usage over time.
2. Personalized tips: Rather than providing generic tips and tricks for reducing energy consumption, consider using the data you collect from each user to provide personalized recommendations based on their energy usage patterns. For example, if a user consistently leaves their TV on standby, your application could suggest ways to reduce standby power usage.
3. Integration with smart home devices: If possible, consider integrating your application with smart home devices such as smart plugs or smart thermostats. This would allow users to control their energy usage directly from your application and receive real-time feedback on their energy consumption.
4. Social sharing: Allow users to share their energy usage data and progress on social media platforms. This could help to raise awareness of energy conservation and encourage others to adopt similar practices.
5. Carbon footprint calculator: In addition to tracking energy usage, consider adding a feature that calculates a user's carbon footprint based on their energy consumption. This could help users to better understand the environmental impact of their energy usage and motivate them to reduce their energy consumption.