**Python Password Safe: Database Design and Implementation**

**Introduction**

In the digital age, managing numerous passwords securely and efficiently is crucial. Python Password Safe offers a robust solution for storing and retrieving passwords with enhanced security measures. This document outlines the database design and implementation for a password safe application, featuring user authentication and encryption, built using Python.

**Application Relevance**

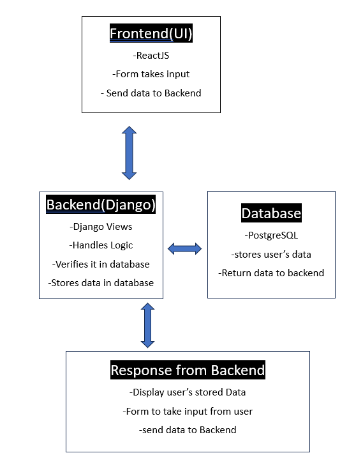
Python Password Safe serves several key purposes in managing digital security:

1. **Secure Storage**: Allows users to store their passwords safely in an encrypted format.
2. **Efficient Retrieval:** Enables users to quickly and securely retrieve their passwords when needed.
3. **User Management:** With user authentication, individuals can manage their own set of passwords, ensuring privacy and security.
4. **Customization:** Users can categorize and organize their passwords according to their preferences.
5. **Enhanced Security**: User authentication and encryption add multiple layers of security, ensuring that users’ sensitive information is well-protected.

Our application focuses on providing Security and improving comfort of the user with password management.

**Database Schema Design**

**Entity-Relationship (ER) Diagram**

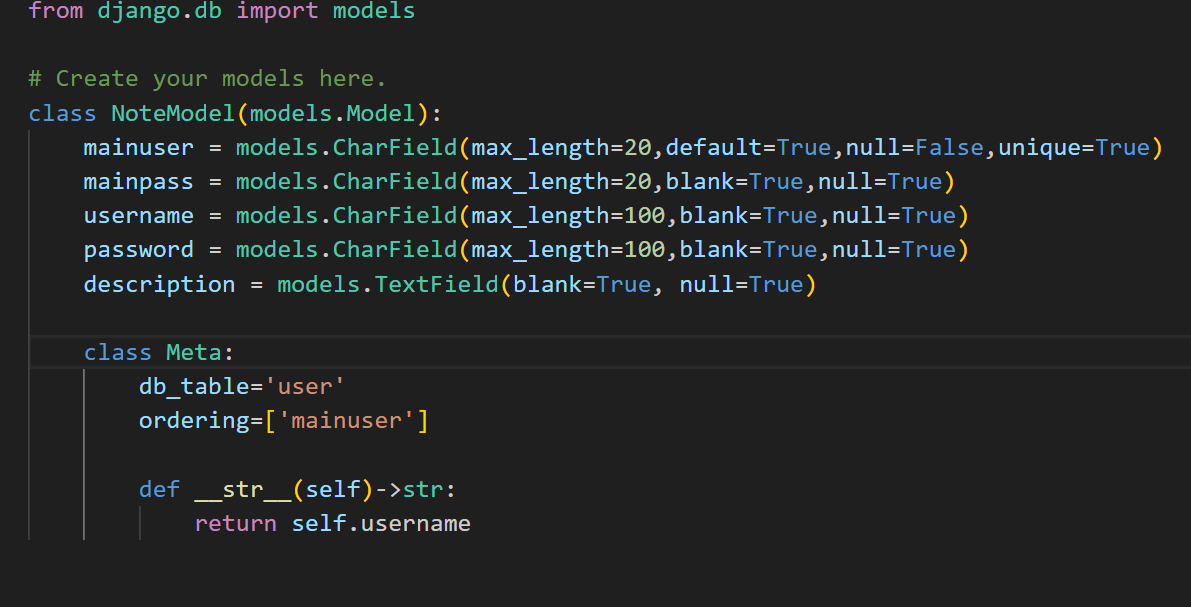


**Schema Explanation**

1. **User Table**: Stores credentials of users.
   * id: Unique identifier for each user (Primary Key)
   * mainuser: Unique username for login
   * mainpass: User's password
   * username: username to be stored by us
   * password: password to be stored by us
   * description: some description of text given by user.

This is very helpful to many users who feels difficult to remember credentials used in various websites. User can remember only username and password of this website and we will remember credentials of user used in other websites.

**Database Implementation**

In Django, the database schema is defined using models. Here's how the models would be implemented: 

Implementation Notes

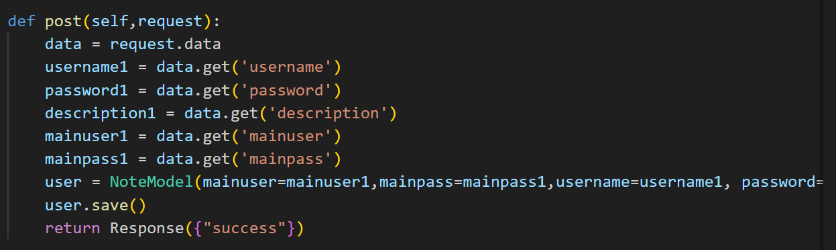
* **User Model**: Django's built-in AbstractUser is used for the User model, providing standard fields and functionality for user authentication.
* **Indexes**: Django will automatically create indexes for primary keys and foreign keys.

**Data Manipulation and Querying**

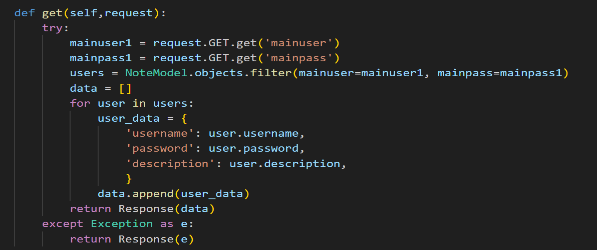
In Django, data manipulation is typically done in POSTGRESQL using SQL. Here are examples of common operations:

**Basic CRUD Operations**

* **Create a new entry**:

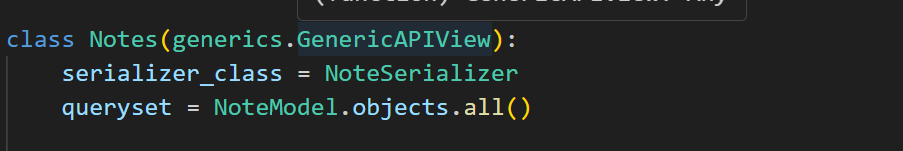


* **Read Data in database**:



**Advanced Queries**

* **Get saved credentials for a specific user**:



* **Enables user to enter new credentials to store**:

user = NoteModel(mainuser=mainuser1,mainpass=mainpass1,username=username1, password=password1, description=description1)

user.save()