**FUNCTIONS**

**FILE: VIEWS.PY**

Functions:

1. checkIn()

The function renders the 'check\_in.html' template, which contains a form for check-in. However, it requires users to log in first

1. checkOut()

The function renders the 'check\_out.html' template, which contains a form for check-out. However, it requires users to log in first

**FILE: AUTH.PY**

Functions:

1. starter\_page()

The function checks whether the user is logged in. If the user is not logged in, it displays a login form. However, if the user is already logged in, it displays the check-in form.

1. login()

The function checks whether the request method is POST or GET. If the method is POST, it assigns the login and password information passed by the user to variables. Subsequently, it searches for the login in the database. If the login is not found, it displays an error message 'Login does not exist.' However, if the user's login is found in the database, it proceeds to check the password. The password entered by the user in the form is hashed and compared to all passwords in the database. If a match is found, it returns 'Logged in successfully!' and logs in the user using the'login\_user' function from the Flask-Login library, redirecting them to 'views.checkIn'. If there is no matching password in the database, it returns an error message: 'Incorrect password, please try again.

1. logout()

The function requires the user to be logged in to access it. To log out the user, it utilizes the 'logout\_user()' function from Flask-Login.

1. sign\_up()

The function represents a Flask route for the '/sign-up' endpoint, accessible via both GET and POST methods. It checks whether the request method is POST or GET. If the method is POST, it assigns the login, password1, and password2 values from the submitted form to variables.

Next, it searches for an existing user in the database with the same login. If a user with the same login already exists, it flashes an error message indicating that the login already exists. Additionally, if the length of the login is less than 4 characters, it flashes an error message stating that the login must be greater than 3 characters.

If the passwords entered in password1 and password2 fields do not match, it flashes an error message indicating that the passwords don't match. Similarly, if the length of password1 is less than 7 characters, it flashes an error message stating that the password must be at least 7 characters.

If none of the above conditions are met, it creates a new user in the database with the provided login and a hashed password, using the SHA256 algorithm. The new user is added to the database, the changes are committed, and the user is logged in using the login\_user() function from the Flask-Login library.

Finally, a success message is flashed indicating that the account has been successfully created, and the user is redirected to the 'views.home' endpoint.

**FILE: \_\_init\_\_.PY**

Functions:

1. create\_app():

It initializes a Flask application instance named app. The app configuration is set with a SECRET\_KEY for session security and a SQLALCHEMY\_DATABASE\_URI pointing to a SQLite database using the DB\_NAME variable. The Flask-SQLAlchemy extension is initialized with the app using db.init\_app(app). The views and auth blueprints are imported, representing different parts of the application's routes and views. The blueprints are registered with the app, assigning them URL prefixes of '/' and '/', respectively. The User table is imported. Within the application context, the function creates all the necessary database tables using db.create\_all().

An instance of LoginManager is created and configured to set the login view as 'auth.login'. It is then initialized with the app using login\_manager.init\_app(app). The load\_user function is defined as a callback for the LoginManager to retrieve the user object based on the user ID.

Finally, the function returns the Flask app instance.