Clinic App Report

Phase 1 of the Project - Advanced Programming Course 21060

Keivan Jamali 99104468 | Nassrin Sharifi 99104779
Erfan Bidmeshki 401103933

GitHub Repository

22/1/2024

Contents

1 Introduction

In the first chapters, our heroes defined the central players - the User, Clinic, Appointment and more that would embark on adventures together. Sketches and schematics in hand, they began mapping relations between characters. Meanwhile, others prepared the stage, designing sturdy tables to safely hold details as the plot thickened.

Now we approach pivotal scenes where limitations are tested and functions formed. Primary users log in to begin their journeys, while assistants encounter tailored menus for their roles. Through it all, an API provides vital support, ensuring times remain true to real clinics.

Of course, such an epic undertaking requires the steeliest of managers. Tasks are portioned according to the sacred Fibonacci pattern, and codes are firmly tied to their duties on our quest board. Each new build promises richer experiences for all involved.

Stay tuned, curiouscrew, for more installments from this lively production! Our troupe works with dedication to deliver quality care - and a ripping good yarn - for their audiences. This writer looks forward to sharing their further adventures!
All the project is available on *GitHub* repository. you can follow us there.

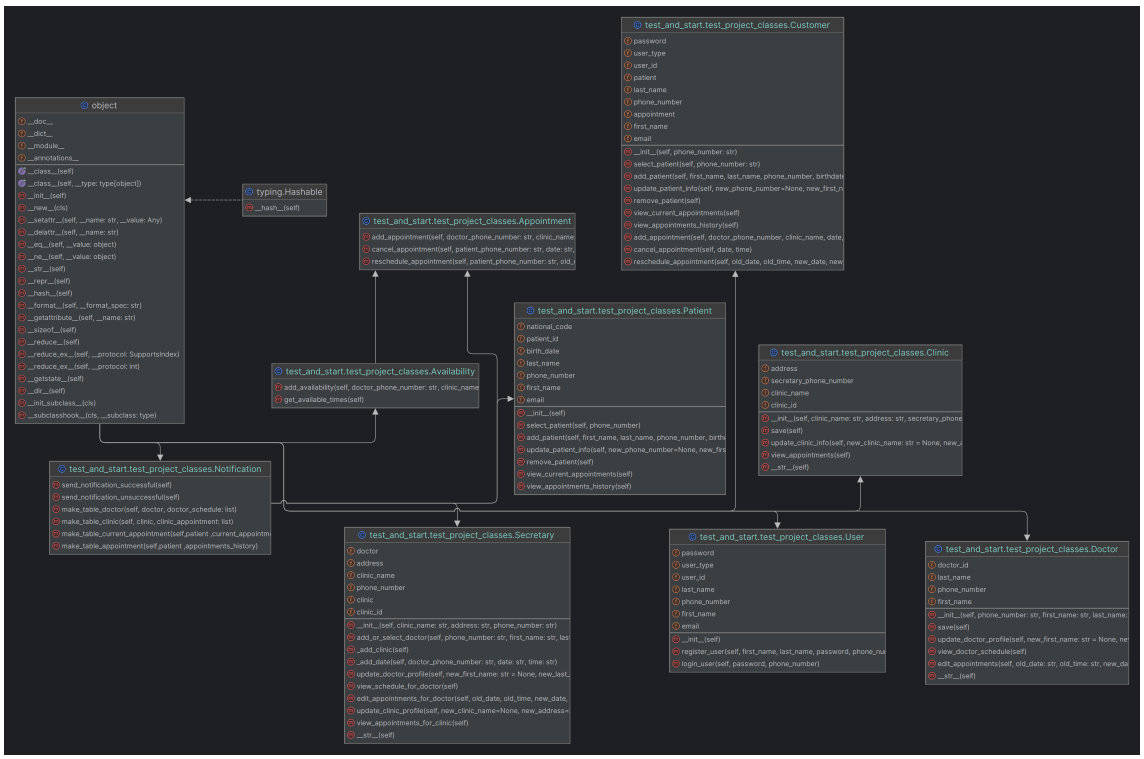

Figure 1: Class Diagram

2 DataBase

1. `user_table`:
    - `user_id` (primary key): A unique identifier for each user.
    - `first_name`: The first name of the user.
    - `last_name`: The last name of the user.
    - `email`: The email address of the user, which must be in a valid format.
    - `phone_number`: The phone number of the user, which must consist of numeric digits and be unique.
    - `password`: The password of the user, which must not be empty.
    - `user_type`: The type of user, which must not be empty.
2. `clinic_table`:
    - `clinic_id` (primary key): A unique identifier for each clinic.
    - `clinic_name`: The name of the clinic, which must be unique.
    - `address`: The address of the clinic.
    - `secretary_phone_number`: The phone number of the clinic's secretary, which must consist of numeric digits.
3. `doctor_table`:
    - `doctor_id` (primary key): A unique identifier for each doctor.
    - `phone_number`: The phone number of the doctor, which must consist of numeric digits and be unique.
    - `first_name`: The first name of the doctor.
    - `last_name`: The last name of the doctor.
4. `patient_table`:
    - `patient_id` (primary key): A unique identifier for each patient.
    - `phone_number`: The phone number of the patient, which must consist of numeric digits and be unique.
    - `first_name`: The first name of the patient.
    - `last_name`: The last name of the patient.
    - `birthdate`: The birthdate of the patient.
    - `national_code`: The national code of the patient, which must consist of numeric digits and be unique.

- email: The email address of the patient, which must be in a valid format.

5. calendar_table:
   - calendar_id (primary key): A unique identifier for each appointment.
   - doctor_id (foreign key): References the doctor_table to identify the doctor associated with the appointment.
   - clinic_id (foreign key): References the clinic_table to identify the clinic where the appointment is scheduled.
   - patient_id (foreign key): References the patient_table to identify the patient who booked the appointment.
   - appointment_date: The date of the appointment.
   - appointment_time: The time of the appointment.
   - canceled: A boolean value indicating whether the appointment has been canceled (default is FALSE).
6. availability_table:
   - availability_id (primary key): A unique identifier for each availability entry.
   - doctor_id (foreign key): References the doctor_table to identify the doctor's availability.
   - clinic_id (foreign key): References the clinic_table to identify the clinic's availability.
   - available_date: The date on which the doctor is available.
   - available_time: The time at which theHere's the continuation of the LaTeX code:
   - available_time: The time at which the doctor is available.
7. doctor_clinic:
   - doctor_id (foreign key): References the doctor_table to identify the doctor.
   - clinic_id (foreign key): References the clinic_table to identify the clinic.
8. customer_patient:
   - patient_id (foreign key): References the patient_table to identify the patient.
   - user_id (foreign key): References the user_table to identify the user associated with the patient.

3 app.py

In this file, we make the interface of the project for user and secretary interactions in a clinic system. The code consists of several functions and classes that handle various actions such as user registration, login, and secretary operations.

1. **Function:** `get_phone_number()`
    - This function prompts the user to enter a phone number and validates the input using a specific pattern. It returns the phone number if it is valid.
2. **Function:** `get_first_name()`
    - The function asks the user to enter their first name and returns the input as a string.
3. **Function:** `get_last_name()`
    - This function prompts the user to enter their last name and returns the input as the last name.
4. **Function:** `get_password()`
    - The function takes user input for a password and checks if it meets specific criteria, including the presence of lowercase and uppercase letters, digits, and special characters. It returns the valid password entered by the user.
5. **Function:** `get_email()`
    - This function prompts the user to enter an email address and validates it using a regular expression pattern. It returns the email address if valid or `None` if the user chooses not to enter an email.
6. **Function:** `choose_user_type()`
    - The function prompts the user to enter their user type, specifically "Secretary" or "Customer". It returns the user type.
7. **Function:** `register_or_login(user)`
    - This function allows a user to either register or login. It takes a user object as a parameter and returns the result of the registration or login attempt.
8. **Function:** `main()`

- The main function handles the user and secretary interactions within the clinic system. It starts by creating a `User` object and then calls the `register_or_login` function to register or log in the user.
- If the user type is "Secretary," it enters a loop where the secretary can perform various actions related to managing a clinic, such as adding/selecting doctors, updating the clinic profile, viewing appointments, and managing doctor profiles.
- If the user type is "Customer," it enters a loop where the customer can perform actions such as adding patients, selecting patients, updating patient information, managing appointments, and viewing appointment history.

Overall, this code provides an interface for users and secretaries to interact with the clinic system. It enables user registration, login, and various operations related to managing clinics, doctors, patients, and appointments.

Please note that the provided code is incomplete, and some variables (such as `exit` in the `main` function) are not initialized or defined. To ensure the code runs properly, these variables need to be appropriately handled and initialized.

4 user.py

To continue our work, we make an user class, which is responsible to login and register in the website.

1. **Function:** `register_user`
    - This function will check first if the user is already exist or not. If not, the user will insert into the table.
2. **Function:** `login_user`
    - In this function if the provided phone number is not in the table it will print an error and otherwise the password will be checked and if correct the user will be login and the object will return. When the app close the user will logout.

5 customer.py | patient.py

Patient Class | Attributes:

1. `__init__(self)`: The constructor function that initializes the attributes of a patient object.

Patient Class | Methods:

1. `select_patient(self, phone_number)`: Selects a patient from the database based on their phone number and sets the patient's attributes accordingly.
2. `add_patient(self, first_name, last_name, phone_number, birthdate, national_code, email, user_phone)`: Adds a new patient to the database with the provided details. It checks if the patient already exists before inserting the new record.
3. `update_patient_info(self, new_phone_number=None, new_first_name=None, new_last_name=None, new_birthdate=None, new_email=None, new_national_code=None`
   Updates the patient's information in the database with the provided new values. It checks for the existence of a patient with the new phone number before performing the update.
4. `remove_patient(self)`: Removes the patient from the database.
5. `view_current_appointments(self)`: Retrieves and displays the current appointments for the patient from the database.
6. `view_appointments_history(self)`: Retrieves and displays the appointment history for the patient from the database.

Customer Class | Attributes:

1. `__init__(self, phone_number: str)`: The constructor function that initializes instances of the Patient, Secretary, and Appointment classes. It establishes a database connection and retrieves the user's information based on their phone number.

Customer Class | Methods:

1. `select_patient(self, phone_number: str)`: Selects a patient from the database based on their phone number. It queries the database and sets the patient's information accordingly.
2. `add_patient(self, first_name, last_name, phone_number, birthdate, national_code, email)`: Adds a new patient to the database. It delegates the functionality to the Patient class.
3. `update_patient_info(self, new_phone_number=None, new_first_name=None, new_last_name=None, new_email=None, new_birthday=None, new_national_code=None)`
   Updates the patient's information in the database. It delegates the functionality to the Patient class.
4. `remove_patient(self)`: Removes the patient from the database.

5. `view_current_appointments(self)`: Retrieves and displays the current appointments for the patient from the database. It delegates the functionality to the Patient class.

6. `view_appointments_history(self)`: Retrieves and displays the appointment history for the patient from the database. It delegates the functionality to the Patient class.

7. `add_appointment(self, doctor_phone_number, clinic_name, date, time)`: Adds a new appointment for the patient. It delegates the functionality to the Appointment class.

8. `cancel_appointment(self, date, time)`: Cancels an existing appointment for the patient. It delegates the functionality to the Appointment class.

9. `reschedule_appointment(self, old_date, old_time, new_date, new_time)`: Reschedules an existing appointment for the patient. It delegates the functionality to the Appointment class.

6 secretary.py | doctor.py | clinic.py

Same as the last part, here we made 3 classes to work for us to perform secretary. To do this, first we make clinic and doctor. then connect the Secretary to them. Each secretary connects to only one clinic but each clinic can have multiple doctors. The secretary have the authority to add doctors and set available time for them in this class.

1. **Function:** `__init__`

   - By making an object, if the secretary is new, or there is no clinic connected to it, it will make the object and add it to proper tables and then select the Secretary and set it to the object. Otherwise if the secretary exist, it will only set to the object.

2. **Function:** `add_doctor`

   - In this function we add a doctor and select it. If the doctor already exist, we only select it.

3. **Function:** `update_doctor_profile`

   - Updates the doctor's profile with the given information (new first name, new last name, new phone number).

4. **Function:** `view_schedule_for_doctor`

- Retrieves the schedule for the doctor and displays it in a table format.

5. **Function:** `edit_appointments_for_doctor`
    - Edits appointments for a doctor by updating the appointment's date and time.
6. **Function:** `update_clinic_profile`
    - Updates the clinic's profile with new information (clinic name, address, secretary's phone number).
7. **Function:** `view_appointments_for_clinic`
    - Retrieves and displays the appointments for the clinic in a table format.
8. **Function:** `__str__`
    - Returns a string representation of the object, including the clinic name and phone number.

7 appointment.py | availability.py

We want to make two classes which add appointments for us and also check or us that if the date is available into the calendar. At the last we can use this classes to get data about appointments. Both the customer and secretary will use it.

1. **Function:** `add_appointment`
    - Here it will take the doctor phone number and clinic name and patient phone number to add the date and time to them and insert the data into calendar.
2. **Function:** `cancel_appointment`
    - By giving the needed information to it, if the appointment exist it will canceled and the cancel column will turn to True. It also change the reserved column in the availability table to False too.
3. **Function:** `reschedule_appointment`
    - This will change the appointment date and its related fileds will fix in both availability table and calendar table.
4. **Function:** `add_availability`
    - It will add some availability to the table.

5. **Function:** `get_available_times`
    - This function will return all the available date and times.