Roles: Sohum (mainly responsible for coding the system) and Srikar (mainly responsible for designing the system and creating the diagrams)

The barbershop management system operates under several key assumptions that shape its design and functionality. First and foremost, the system operates entirely in-memory with no persistent storage implementation, as confirmed by the teaching assistant (Arsalan). This means all data is lost upon program termination, and no database or file system integration is required. Additionally, the system assumes all user inputs are valid and correctly formatted, implementing minimal error handling as these requirements were not explicitly stated in the initial project proposal. The system starts with no pre-existing data, requiring all accounts, appointments, and reviews to be created during runtime, with no initial data seeding necessary. The interface is graphical user interface implementation, and all interactions are text-based without special formatting requirements. The system assumes single-user access at a time, with no concurrent user handling, session management, or timeout mechanisms required.

The system's workflow follows a sequential process that begins with barber account creation and availability setup. A typical system walkthrough starts with a barber creating an account, signing in, and establishing their availability slots. This process can be repeated for multiple barbers, each creating their own account and setting up their individual availability schedules. Once the barbers are set up, a client can create their account, log in, and proceed to schedule availabilities. The client can log out after scheduling, and a barber can then log in to manage these appointments. Barbers have the capability to cancel appointments, modify their availability slots, and mark appointments as completed. When a client logs back in, they can view their past appointments and engage with the rating system. The rating system allows clients to rate the barbershop as a whole or rate individual barbers for specific appointments.

Through this project, we were able to learn about the utility of the SOLID principles when coding/designing the project, as they really helped us when we needed to repeatedly reconfigure aspects of our project. The system's architecture is built upon the SOLID principles of object-oriented design, ensuring a robust and maintainable codebase. The Single Responsibility Principle is implemented through the separation of distinct functionalities into different classes, such as `AccountData` handling user authentication, `BarberShop` managing business operations, and `AppointmentConfirmedHandler` dealing with appointment processing. The Open/Closed Principle is demonstrated through the system's extensibility, where new features can be added without modifying existing code, such as the ability to add new types of appointments or rating systems without changing the core appointment management logic. The Liskov Substitution Principle is maintained through the proper inheritance hierarchy, as none of the classes required inheritance that would make sense. The Interface Segregation Principle is applied by ensuring that classes only implement the methods they need. Finally, the Dependency Inversion Principle is followed by depending on abstractions rather than concrete implementations, allowing for flexible component interactions and easier testing and maintenance. This adherence to SOLID principles results in a system that is not only functional but also maintainable, extensible, and well-structured, making it easier to add new features or modify existing ones without compromising the system's integrity.