## **Indice**

1. Funzionalità richieste
2. Descrizione UML e diagrammi
3. Tecnologie usate e motivazioni
4. Organizzazione logica del codice
5. Scelte implementative e trade‑off
6. Test eseguiti
7. screenshot del progetto

Il progetto “Barbershop” è un sistema di prenotazione online per un salone di barbiere, sviluppato con Django 5.x e SQLite. L’applicazione è strutturata in quattro moduli principali:

* **users**: gestisce registrazione, login/logout e profili utente (con distinzione tra cliente e barbiere, avatar e descrizione);
* **services**: consente ai barbieri di creare, modificare ed eliminare i propri servizi (nome, descrizione, prezzo, durata e immagine) e agli utenti di visualizzare, filtrare per parola chiave, prezzo o durata e navigarli con paginazione;
* **appointments**: permette ai clienti registrati di prenotare un servizio scegliendo data, ora e (opzionalmente) barbiere; include controlli di conflitto orario, suggerimenti di slot liberi e funzioni di modifica/cancellazione;
* **notifications**: invia messaggi interni a clienti e barbieri per confermare creazione, modifica o cancellazione di prenotazioni, con etichettatura “non letto” e “letto” e paginazione nella inbox.

### **1. Funzionalità**

**1.1 Registrazione e autenticazione utenti**

* **Cliente/Barbiere**: form basato su django.contrib.auth.forms.UserCreationForm esteso in UserRegisterForm per raccogliere username, email, password e flag is\_barber.
* **Login/Logout**: utilizzo di view LoginView/LogoutView di Django, protezione CSRF attiva automaticamente.
* **Reset password**: supporto integrato con PasswordResetView e PasswordResetConfirmView forniti out‑of‑the‑box (pronti all'uso senza configurazioni aggiuntive).
* *Motivazione*: autenticazione integrata di Django garantisce hashing sicuro delle password, vista e URL preconfigurati, riducendo dispersione di codice e vulnerabilità.

**1.2 Gestione profilo**

* Modifica email, upload avatar e descrizione personale (breve introduzione dell'utente).
* Implementazione via ProfileForm (ModelForm su Profile) che sincronizza User.email e Profile.description, e view profile\_view con GET/POST per rendering e salvataggio.

**1.3 Servizi**

* **Model**: Service memorizza name, description, price, duration (DurationField), image.
* **ServiceForm**: ModelForm che rimuove duration dal form e aggiunge duration\_minutes (IntegerField); su save() converte minuti in timedelta.
* **Filtro**: ServiceFilterForm con campi q, min\_price, max\_price, duration\_minutes, usato in service\_list e homepage.
* **Paginazione**: Paginator con 5 elementi per pagina.
* *Motivazione*: utilizzo Django ORM per query filter(), union di QuerySet per ricerca full-text, trasformazioni in clean/save semplificano UX.

**1.4 Prenotazioni**

* **AppointmentForm**: gestisce date con DateInput(min=today), time con ChoiceField generato dinamicamente, barber opzionale.
  + Validazione in clean(): parsing orario ISO/via strptime, blocco slot passati nella giornata.
* **logica view**: appointment\_create e appointment\_edit assegnano barber scelto o random da Profile.objects.filter(is\_barber).
* **Controllo conflitti**: se esiste già Appointment per barber, date e time, chiama get\_next\_slots() che cicla 7 giorni e slot 20-min per suggerimenti.
* **Notifiche**: creazione istanze Notification per cliente e barbiere immediatamente dopo salvataggio.

**1.5 Notifiche interne**

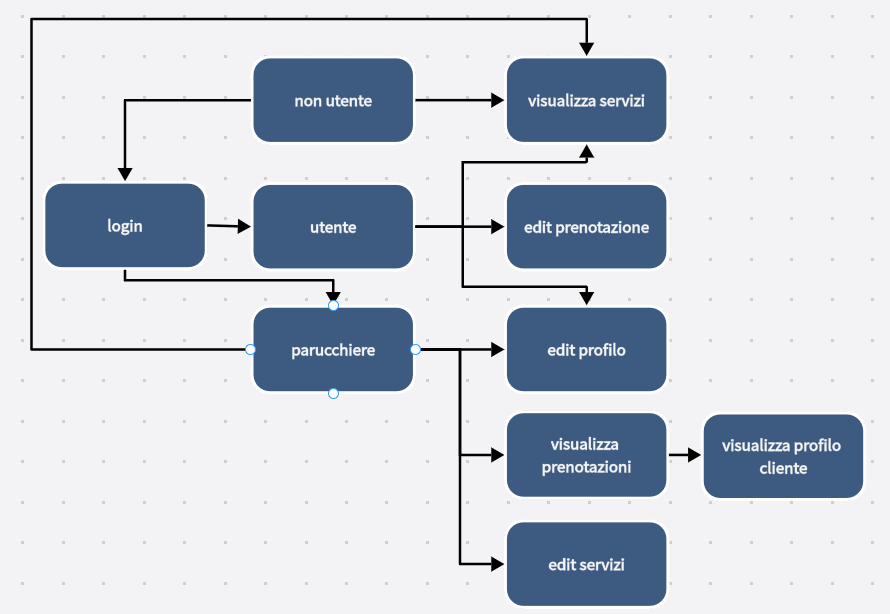
* **Model**: Notification con FK a User, message, created\_at, is\_read.
* **Context processor**: unread\_notification\_count espone count() di notifiche non lette.
* **Views**: notification\_list con paginazione, mark\_as\_read che imposta is\_read=True.

**1.6 UI responsiva**

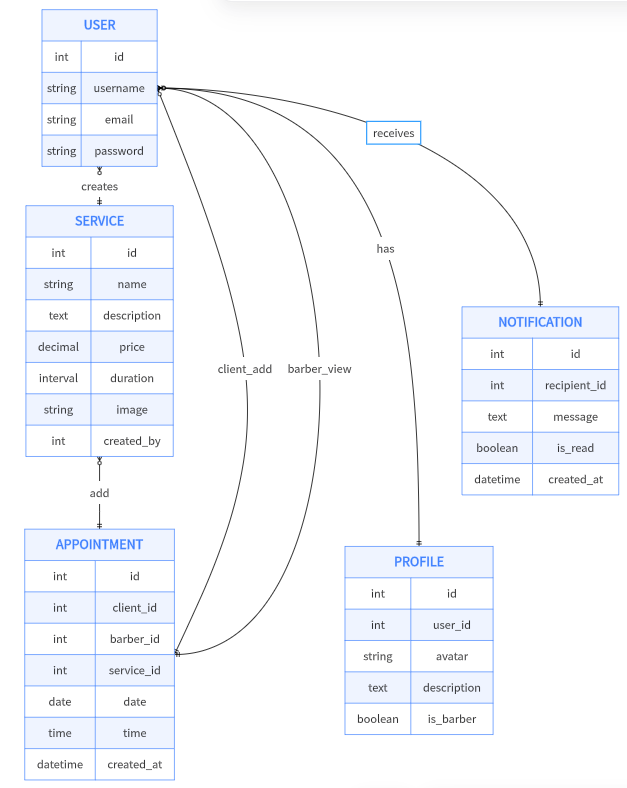
* **Template base**: base.html con navbar dinamica che utilizza tag {% if user.is\_authenticated %} per mostrare link diversi (My Services, Appointments, Profile, Notifications) e badge con numero di notifiche non lette tramite context processor unread\_notification\_count.
* **Navbar**: implementata con componenti Bootstrap (navbar-expand, navbar-toggler, collapse) per supporto mobile.
* **Forms**: uso di {% load widget\_tweaks %} per applicare in template la classe form-control ai campi Django Form, migliorando l’aspetto e la coerenza con Bootstrap.
* **Componenti**: cards per servizi e appuntamenti, alerts per messaggi di conferma e errori, list-group per notifiche e suggerimenti.
* **Responsive design**: grid system di Bootstrap (row, col-md-6, d-flex, flex-shrink-0) per adattare layout su desktop e device mobili.

### **2. Descrizione UML e diagrammi**

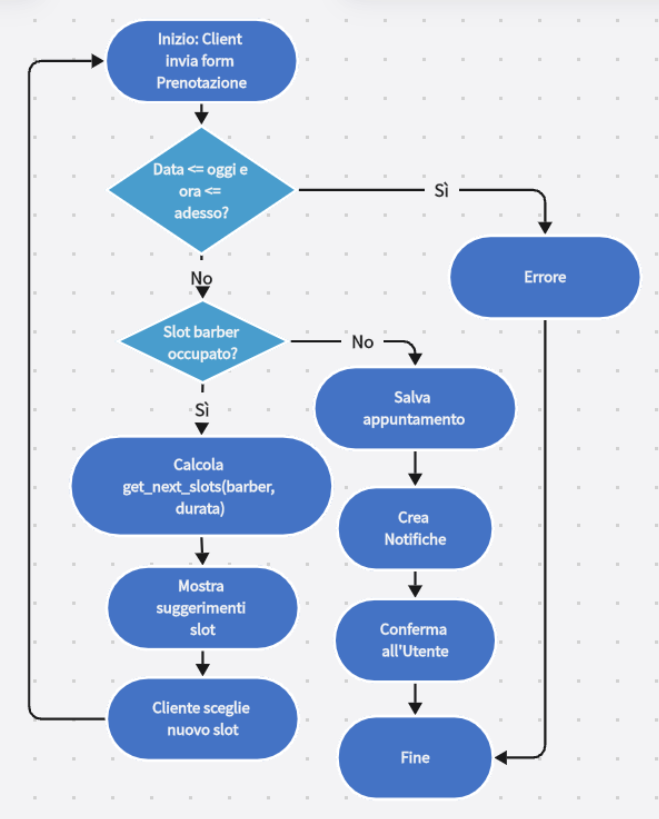
#### **2.1 Use Case Diagram**



#### **2.2 Class Diagram / Schema ER**



#### **2.3 Activity Diagram: prenotazione**



### **3. Tecnologie usate e motivazioni**

#### **3.1 Django 5.2 & Auth integrato**

* **Framework MVC**: separa Model, View, Controller → facilità manutenzione, test.
* **Django ORM**: mapping Python↔DB (QuerySet, filter, exclude, order\_by). Evita SQL diretto.
* **django.contrib.auth**: views, form, backend pronti per login, logout, reset password, CSRF, hashing password.
* **Scelta del login/auth**: utilizzo del sistema di autenticazione integrato di Django (django.contrib.auth) che include views, form e backend già pronti per login, logout, reset password, protezione CSRF e hashing delle password, garantendo rapidità di sviluppo e sicurezza elevata.

#### **3.2 Bootstrap 5.3**

* **Responsive**: mobile-first grid, navbar, card, form.
* **Confronto**: rispetto a Tailwind richiede minore setup, classi semanticamente riconoscibili.

#### **3.3 Pillow**

* Libreria ufficiale per elaborazione immagini, integrata in Django ImageField.
* *Alternativa*: ImageKit per processing avanzato, ma eccessivo per caricamenti base.

#### **3.4 Database**

* **SQLite**: database leggero, zero-config, integrato di default in Django; ideale in sviluppo per semplicità, tempi di setup nulli e facilità di backup (file singolo), con supporto completo per ACID a livello di piccole applicazioni.
* **Motivazione**: scelta di SQLite in tutte le fasi di sviluppo e test per rapidità, mentre PostgreSQL può essere adottato in produzione se necessario per scalabilità e funzionalità avanzate.

### **4. Organizzazione logica del codice**

* **Cartella progetto**: barbershop\_project con settings.py (DB, static/media, Auth), urls.py include app.
* **App**: users, services, appointments, notifications
* **Custom command**: services/management/commands/generate\_test\_data.py per popolamento rapido e testing.
* **Templates**: base.html, directory per app.
* **Static/Media**: static/css/style.css, media/ per upload.

### **5. Scelte implementative e trade‑off**

* **Gestione durata servizi**: il modello Service utilizza un campo DurationField per memorizzare timedelta; nel ServiceForm si nasconde duration e si aggiunge duration\_minutes (IntegerField), quindi in save() si converte il valore in timedelta con timedelta(minutes=mins). Questa scelta semplifica l'input utente mostrando direttamente minuti, mantenendo precisione sul DB.
* **Selezione slot e suggerimenti**: la funzione get\_next\_slots(barber, service\_duration, days\_ahead=7, max\_slots=3) genera i prossimi slot liberi controllando fino a 7 giorni a intervalli di 20 minuti (range(9–17, step 0, 20, 40)). Si tratta di un algoritmo greedy di complessità O(days \* slots\_per\_day), facilmente estendibile.
* **Notifiche in-app**: implementate tramite il modello Notification e context processor unread\_notification\_count. Dopo ogni operazione (creazione/modifica/cancellazione di Appointment), si crea istanza di Notification per cliente e barbiere. Il trade‑off è preferire notifiche interne anziché email, riducendo complessità SMTP e facilitando i test.
* **Validazioni form e sicurezza**:
  + ServiceForm.clean\_duration\_minutes(): impedisce minuti negativi lanciando ValidationError.
  + AppointmentForm.clean(): parsing orario via time.fromisoformat e fallback datetime.strptime, blocco slot già passati nella giornata.
  + Decoratori @login\_required su view sensibili e controllo request.user == instance.created\_by in service\_edit/delete e appointment\_edit/delete per autorizzazioni.
  + Protezione CSRF abilitata out‑of‑the‑box da Django su tutti i form.
* **UI e UX**:
  + Uso di Django template con {% load widget\_tweaks %} per aggiungere classi form-control e btn in modo dinamico, migliorando coerenza con Bootstrap.
  + Layout responsivo basato su Bootstrap grid (row, col-md-\*) e componenti (card, list-group, pagination).

### **6. Test eseguiti**

I test sono stati eseguiti manualmente e tramite custom management command generate\_test\_data.py:

1. **Registrazione utente**: verifica creazione di un utente cliente tramite UserRegisterForm e automatic login; conferma presenza di profile con is\_barber=False.
2. **Registrazione parrucchiere**: creazione utente parrucchiere con flag is\_barber=True e possibilità di accedere a CRUD servizi.
3. **Prenotazione slot libero**: il client effettua un appointment in uno slot libero (es. 13:00), verifica salvataggio di Appointment e invio Notification al parrucchiere .
4. **Prenotazione conflittuale**: tentativo di prenotare in uno slot già occupato, esecuzione della funzione get\_next\_slots() e visualizzazione dei suggerimenti in console (print di slot suggeriti).
5. **Creazione nuovo servizio**: tramite script di test, il parrucchiere crea un nuovo Service; verifica che Service.objects.get\_or\_create() funzioni correttamente.

### **7.Screenshot del progetto**

