BARBER APP

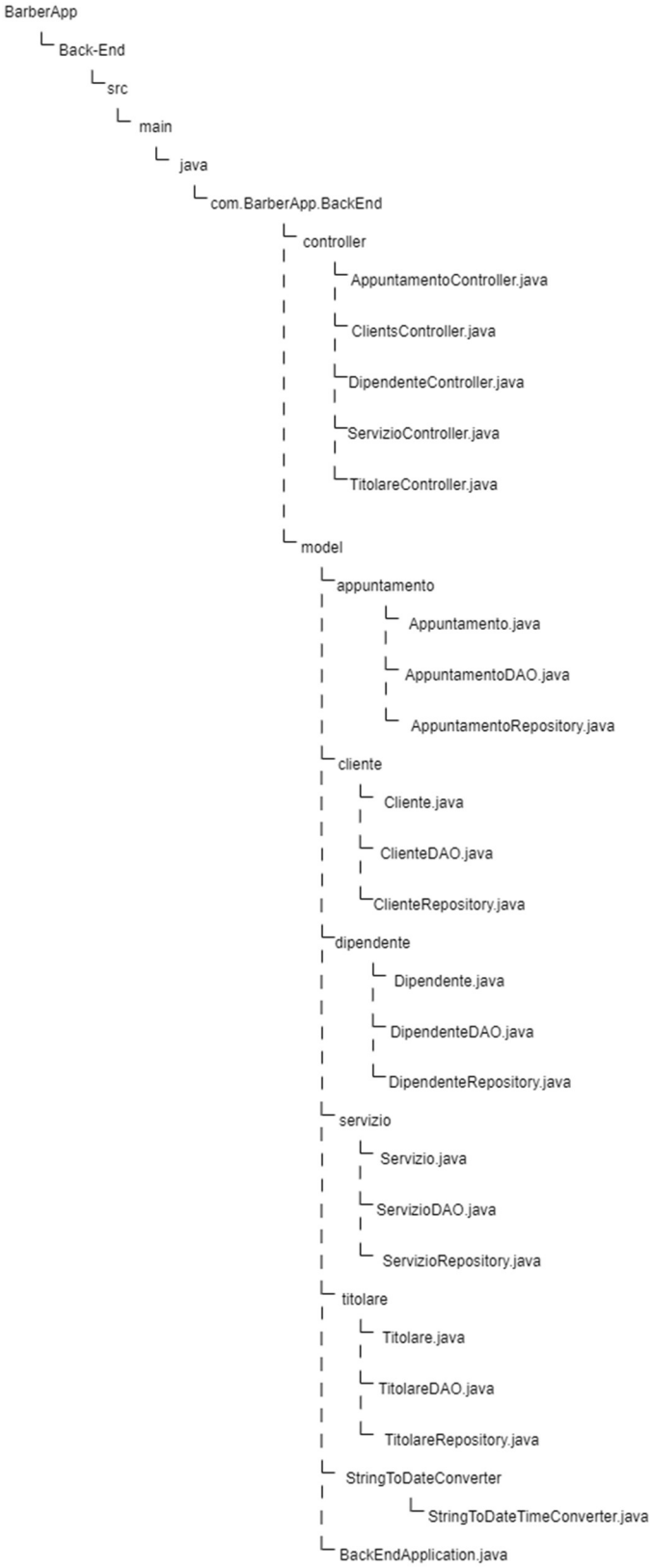
System Object Design (ODD)

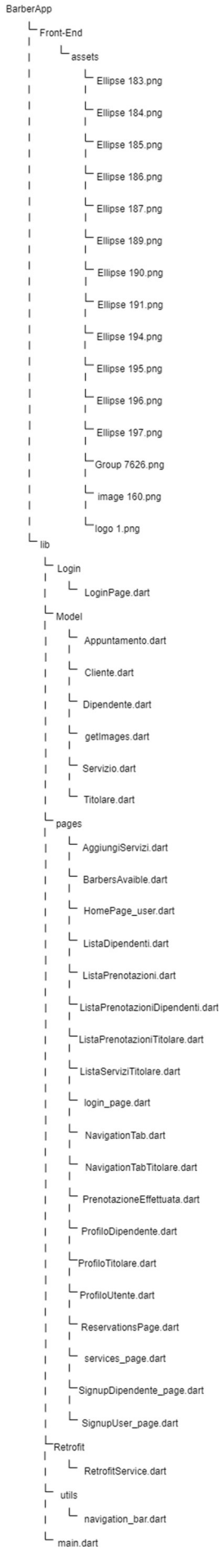
Autori
Francesco Pio Guerriero matr. 0512114790
Luca Vincenzo Lambiase Fontana matr. 0512113338
Ivan Prota matr. 0512106063

Sommario

Directories

Come già detto nel documento di System Design, il nostro sistema segue un modello architetturale MVVM, facendo eseguire il front-end sui dispositivi client mentre il back-end su un dispositivo remoto. Di seguito è riportato la struttura delle directory dell'intero sistema:




BarberApp
Front-End
assets
Ellipse 183.png
Ellipse 184.png
Ellipse 185.png
Ellipse 186.png
Ellipse 187.png
Ellipse 189.png
Ellipse 190.png
Ellipse 191.png
Ellipse 194.png
Ellipse 195.png
Ellipse 196.png
Ellipse 197.png
Group 7626.png
image 160.png
logo 1.png
lib
Login
LoginPage.dart
Model
Appuntamento.dart
Cliente.dart
Dipendente.dart
getImages.dart
Servizio.dart
Titolare.dart
pages
AggiungiServizi.dart
BarbersAvaible.dart
HomePage_user.dart
ListaDipendenti.dart
ListaPrenotazioni.dart
ListaPrenotazioniDipendenti.dart
ListaPrenotazioniTitolare.dart
ListaServiziTitolare.dart
login_page.dart
NavigationTab.dart
NavigationTabTitolare.dart
PrenotazioneEffettuata.dart
ProfiloDipendente.dart
ProfiloTitolare.dart
ProfiloUtente.dart
ReservationsPage.dart
services_page.dart
SignupDipendente_page.dart
SignupUser_page.dart
Retrofit
RetrofitService.dart
utils
navigation_bar.dart
main.dart

Specifica delle interfacce

Durante la definizione delle interfacce, abbiamo optato per non specificare tutti i metodi getter e setter ovvi presenti nelle classi entity.

Le classi che terminano con "DAO" rappresentano l'unico punto di accesso al database nell'applicazione. Inoltre, alcune condizioni particolarmente intricate, che non si prestavano facilmente all'espressione in OCL, sono state delineate attraverso il linguaggio naturale, garantendo una comprensione più chiara e immediata.

ClientsController	
Descrizione	Classe che gestisce le operazioni del Cliente
getAllClients	Metodo che ritorna una collezione di tutti i clienti salvati nel sistema
save	Metodo che registra un cliente all'interno del sistema **context** ClientsController::save(cliente : Cliente) **pre**: cliente <> null **context** ClientsController::save(cliente : Cliente) **post**: result = 200 || result = 500
delete	Metodo che elimina un cliente all'interno del sistema **context** ClientsController::delete(cliente : Cliente) **pre**: cliente <> null **context** ClientsController::delete(cliente : Cliente) **post**: result = 200 || result = 500
check	Metodo che verifica la presenza dell'indirizzo email del cliente all'interno del sistema **context** ClientsController::check(email : String) **pre**: email <> null & email <> "" **context** ClientsController::check(email : String) **post**: result = 200 || result = 500

update	Metodo che aggiorna le informazioni del cliente all'interno del sistema **context** ientsController::update(cliente : Cliente) **pre**: cliente <> null **context** ClientsController::update(cliente : Cliente) **post**: result = 200 || result = 500 || result = 501

AppuntamentoController	
Descrizione	Classe che gestisce le operazioni sugli Appuntamenti
getAllAppuntamenti	Metodo che ritorna una collezione di tutti gli appuntamenti salvati nel sistema
saveAppuntamento	Metodo che salva un appuntamento all'interno del sistema **context** AppuntamentoController::save(appuntamento : Appuntamento) **pre**: appuntamento <> null **context** AppuntamentoController::save(appuntamento : Appuntamento) **post**: result = 200 || result = 500
removeAppointment	Metodo che rimuove un appuntamento all'interno del sistema **context** AppuntamentoController::removeAppointment(appuntamento : Appuntamento) **pre**: appuntamento <> null **context** AppuntamentoController::removeAppointment(appuntamento : Appuntamento) **post**: result = 200

DipendneteController	
Descrizione	Classe che gestisce le operazioni dei Dipendenti
getAllEmployee	Metodo che ritorna una collezione di tutti i dipendenti salvati nel sistema
saveEmployee	Metodo che salva un appuntamento all'interno del sistema **context** DipendenteController::saveEmployee(dipendente : Dipendente) **pre**: dipendente <> null **context** DipendenteController::saveEmployee(dipendente : Dipendente) **post**: result = 200 || result = 500
delete	Metodo che elimina un dipendente all'interno del sistema **context** DipendenteController::delete(dipendente : Dipendente) **pre**: dipendente <> null **context** DipendenteController::delete(dipendente : Dipendente) **post**: result = 200
update	Metodo che aggiorna le informazioni del dipendente all'interno del sistema **context** DipendenteController::update(dipendente : Dipendente) **pre**: dipendente <> null **context** DipendenteController::update(dipendente : Dipendente) **post**: result = 200 || result = 500
getDipendentiByDate	Metodo che ritorna una collezione di dipendenti liberi in base alla data e all'ora

ServizioController	
Descrizione	Classe che gestisce le operazioni sui servizi
getAllServices	Metodo che ritorna una collezione di tutti i servizi presenti all'interno del sistema
saveServizio	Metodo che salva un servizio all'interno del sistema **context** ServizioController::saveServizio(servizio : Servizio) **pre**: servizio <> null **context** ServizioController::saveServizio(servizio : Servizio) **post**: result = 200
deleteServizio	Metodo che elimina un servizio all'interno del sistema **context** ServizioController::deleteServizio(servizio : Servizio) pre: servizio <> null **context** ServizioController::deleteServizio(servizio : Servizio) post: result = 200
updateService	Metodo che aggiorna le informazioni di un servizio all'interno del sistema **context** ServizioController::updateService(servizio : Servizio) **pre**: servizio <> nll **context** ServizioController::updateService(servizio : Servizio) **post**: result = 200 **||** result = 501

TitolareController	
Descrizione	Classe che gestisce le operazioni dei Titolari
getAllTitolari	Metodo che restituisce una collezione di tutti i titolari all'interno del sistema
saveTitolari	Metodo che salva un titolare all'interno del sistema

	context TitolareController::saveTitolari(titolare : Titolare) **pre**: titolare <> null **context** TitolareController::saveTitolari(titolare : Titolare) **post**: result = 200 || result = 500
delete	Metodo che elimina un titolare all'interno del sistema **context** TitolareController::delete(titolare : Titolare) **pre**: titolare <> null **context** TitolareController::delete(titolare : Titolare) **post**: result = 200
update	Metodo che aggiorna le informazioni di un titolare all'interno del sistema **context** TitolareController::update(titolare : Titolare) **pre**: titolare <> null **context** TitolareController::update(titolare : Titolare) **post**: result = 200 || result = 500 || result = 501

Cliente	
Descrizione	Classe che rappresenta un cliente all'interno del sistema
Invariante	**context** Cliente **inv**: id > 0 & (nome <> null & nome <> "") & (cognome <> null & cognome <> “)& (email <> null & email <> "" & email.matches('^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$')) & (password <> null & password <> "") & (appuntamenti<> null & appuintamenti.size() >= 0)

Dipendnete	
Descrizione	Classe che rappresenta un dipendente del salone all'interno del sistema
Invariante	**context** Dipendente **inv**: id > 0 & (nome <> null & nome <> "") & (cognome <> null & cognome <> "") & (email <> null & email <> "" & email.matches('^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$')) & (password <> null & password <> "") & (appuntamenti<> null & appuntamenti.size() >= 0)

Titolare	
Descrizione	Classe che rappresenta un titolare del salone all'interno del sistema
Invariante	**context** Titolare **inv**: id > 0 & (nome <> null & nome <> "") & (cognome <> null & cognome <> "") & (email <> null & email <> "" & email.matches('^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$')) & (password <> null & password <> "") & (appuntamenti<> null & appuntamenti.size() >= 0) & (servizi <> null & servizi.size() >= 0)

Appuntamento	
Descrizione	Classe che rappresenta un appuntamento all'interno del sistema
Invariante	**context** Appuntamento **inv**: id > 0 & date <> null & time <> null & cliente <> null & dipendente <> null & servizio <> null
Servizio	
Descrizione	Classe che rappresenta un servizio del salone all'interno del sistema
Invariante	**context** Servizio **inv**:

	id > 0 & (tipo <> null & tipo <> "") & (assetImage <> null & assetImage <> "") & costo > 0

AppuntamentoDAO	
Descrizione	Classe di tipo Data Access Object che gestisce la persistenza di oggetti di tipo Appuntamento
getAll	Metodo che restituisce una collezione di tutti le occorrenze di tipo Appuntamento presenti all'interno del sistema
saveAppointment	Metodo che salva in maniera persistente all'interno del sistema un'occorrenza di tipo Appuntamento **context** AppuntamentoDAO::saveAppointment(appuntamento : Appuntamento) **pre**: appuntamento <> null
deleteAppointment	Metodo che elimina dal sistema un'occorrenza di tipo Appuntamento **context** AppuntamentoDAO::deleteAppointment(appuntamento : Appuntamento) **pre**: appuntamento <> null
checkAppuntamento	Metodo che controlla la presenza all'interno del sistema di un appuntamento registrato in un determinato giorno **context** AppuntamentoDAO::checkAppuntamento(cliente : Cliente, appuntamento : Appuntamento) **pre**: cliente <> null & appuntamento <> null
getAppuntamentiByCliente	Metodo che ritorna una collezione contente tutte le occorrenze di tipo Appuntamento collegati ad un dato cliente **context** AppuntamentoDAO::getAppuntamentiByCliente(cliente : Cliente) **pre**: cliente <> null

ClienteDAO	
Descrizione	Classe di tipo Data Access Object che gestisce la persistenza di oggetti di tipo Cliente

getClienteById	Metodo che ricerca all'interno del sistema un determinato cliente con un dato id **context** ClienteDAO::getClienteById(id : int) **pre**: id > 0 **context** ClienteDAO::getClienteById(id : int) **post**: result = occorrenza di cliente con quel dato id || result = null
checkCliente	Metodo che verifica la presenza o meno di un'occorrenza di cliente con una determinata email **context** ClienteDAO::checkCliente(email : String) **pre**: email <> null & email <> ""
saveCliente	Metodo che salva in maniera persistente all'interno del sistema un'occorrenza di tipo Cliente **context** ClienteDAO::saveCliente(cliente : Cliente) **pre**: cliente <> null
getAllClienti	Metodo che ritorna una collezione contenente tutte le occorrenze di tipo Cliente presenti all'interno del sistema
deleteCliente	Metodo che elimina dal sistema un'occorrenza di tipo Cliente **context** ClienteDAO::deleteCliente(cliente : Cliente) **pre**: cliente <> null
updateClient	Metodo che permette di aggiornare le informazioni di un'occorrenza di tipo Cliente **context** ClienteDAO::updateClient(cliente : Cliente) **pre**: cliente <> null

DipendenteDAO	
Descrizione	Classe di tipo Data Access Object che gestisce la persistenza di oggetti di tipo Dipendente
getDipendenteById	Metodo che ricerca all'interno del sistema un determinato dipendente con un dato id **context** DipendenteDAO::getDipendenteById(id : int) **pre**: id > 0 **context** DipendenteDAO::getDipendenteById(id : int) **post**: result = occorrenza di dipendente con quel dato id || result = null

checkDipendente	Metodo che controlla all'interno del sistema la presenza di un'occorrenza di tipo Dipendente con una determinata email **context** DipendenteDAO::checkDipendente(email : String) **pre**: email <> null & email <> ""
getEmployee	Metodo che ritorna una collezione contente tutte le occorrenza presenti all'interno del sistema di tipo Dipenente
saveDipendente	Metodo che salva in maniera persistente all'interno del sistema un'occorrenza di tipo Dipendente **context** DipendenteDAO::saveDipendente(dipendente : Dipendente) **pre**: dipendente <> null
deleteDipendente	Metodo che elimina all'interno del sistema un'occorrenza di tipo Dipendente **context** DipendenteDAO::deleteDipendente(dipendente : Dipendente) **pre**: dipendente <> nul
updateEmployee	Metodo che aggiorna le informazioni di una determinata occorrenza di tipo Dipendente presente all'interno del sistema **context** DipendenteDAO::updateEmployee(dipendente : Dipendente) **pre**: dipendente <> null
getEmployeeByDate	Metodo che ritorna una collezione contenente tutte le occorrenze di tipo Dipendente in base ad una determinata data e orario **context** DipendenteDAO::getEmployeeByDate(data : DateTime, ora : DateTime) **pre**: data <> null & ora <> null

ServizioDAO	
Descrizione	Classe di tipo Data Access Object che gestisce la persistenza di oggetti di tipo Servizio
getAll	Metodo che ritorna una collezione contente ogni occorrenza presente all'interno del sistema di tipo Servizio
saveService	Metodo che salva all'interno del sistema un'occorrenza di tipo Servizio **context** ServizioDAO::saveService(servizio : Servizio) **pre**: servizio <> null
deleteService	Metodo che elimina dal sistema un'occorrenza di tipo Servizio

	context ServizioDAO::deleteService(servizio : Servizio) **pre**: servizio <> null
serviceUpdate	Metodo che aggiorna le informazioni di una determinata occorrenza di tipo Servizio **context** ServizioDAO::serviceUpdate(servizio : Servizio) **pre**: servizio <> null

TitolareDAO	
Descrizione	Classe di tipo Data Access Object che gestisce la persistenza di oggetti di tipo Titolare
getTitolareById	Metodo che ricerca all'interno del sistema una determinata occorrenza di tipo Titolare con un dato id **context** TitolareDAO::getTitolareById(id : int) **pre**: id > 0 **context** TitolareDAO::getTitolareById(id : int) **post**: result = occorrenza di titolare con quel dato id || result = null
checkTitolare	Metodo che verifica la presenza o meno di un'occorrenza di tipo Titolare con una determinata email **context** TitolareDAO::checkTitolare(email : String) **pre**: email <> null & email <> ""
getAll	Metodo che ritorna ogni occorrenza presente all'interno del sistema di tipo Titolare
saveTitolare	Metodo che salva in maniera persistente all'interno del sistema un'occorrenza di tipo Titolare **context** TitolareDAO::saveTitolare(titolare : Titolare) **pre**: titolare <> null
deleteTitolare	Metodo che elimina dal sistema un'occorrenza di tipo Titolare **context** TitolareDAO::deleteTitolare(titolare : Titolare) **pre**: titolare <> null

updateTitolare	Metodo che aggiorna le informazioni di un'occorrenza di tipo Titolare presente all'interno del sistema **context** TitolareDAO::updateTitolare(titolare : Titolare) **pre**: titolare <> null

Mapping Data Storage Schema

Nel progettare il data storage schema per la nostra applicazione utilizzando Spring Boot, abbiamo adottato una struttura basata su diverse entità chiave: cliente, dipendente, titolare, appuntamento e servizio. Questa decisione è stata presa per riflettere accuratamente la logica e le relazioni fondamentali presenti nel nostro dominio di business.

Per quanto riguarda le entità cliente, dipendente e titolare, abbiamo identificato una serie di attributi comuni cruciali per il nostro sistema. Questi includono un identificatore univoco (id) per ciascuna entità, insieme a dettagli essenziali come nome, cognome, email e password. Questi attributi sono fondamentali per la gestione degli account utente e per consentire l'accesso sicuro e personalizzato alla nostra piattaforma.

Parallelamente, l'entità appuntamento è stata inclusa nel nostro schema per gestire efficacemente la pianificazione e la gestione degli incontri tra clienti e dipendenti. Gli attributi dell'appuntamento, tra cui id, data e ora, sono stati scelti per fornire un quadro dettagliato delle prenotazioni, facilitando la loro organizzazione e sincronizzazione all'interno del sistema.

Infine, l'entità servizio è stata introdotta per consentire una gestione efficiente e organizzata delle prestazioni offerte dal salone. Oltre all'identificatore (id), abbiamo incluso attributi quali tipo e costo per definire chiaramente i diversi servizi disponibili e i relativi costi associati ed un attributo contenente informazioni sul path in cui sono conservate le immagini dei servizi. Abbiamo quindi optato di non mantenere le immagini sul database, ma di mantenerle all'interno del nostro filesystem.

Complessivamente, la scelta di strutturare il nostro database utilizzando queste entità riflette la nostra volontà di creare un sistema flessibile, intuitivo e ben organizzato, in grado di soddisfare le esigenze operative e funzionali della nostra attività.

Per quanto riguarda il mapping delle associazioni abbiamo introdotto all'entità cliente una collezione di appuntamenti al fine di tenere traccia di ogni singola prenotazione effettuata sulla piattaforma, ed analogamente abbiamo inserito all'entità appuntamento un riferimento al cliente. A quest'ultimo, inoltre, abbiamo incluso un riferimento all'entità titolare per ricavare informazioni riguardante la cancellazione,

e un riferimento al tipo di servizio che il cliente ha scelto in fase di prenotazione. All'entità titolare, allo stesso modo, abbiamo inserito una collezione di appuntamenti ed una collezione di servizi con l'obbiettivo di conservare notizia di chi ha inserito un servizio o eliminato un appuntamento. Conformemente all'entità cliente, abbiamo inserito all'entità dipendente una collezione di appuntamenti per tener traccia di tutte le prenotazioni effettuate con la scelta di quel determinato barbiere. L'ultima entità è servizio, in cui è stato inserito un riferimento ad un titolare ed una collezione di appuntamenti che richiedono quel determinato servizio.