Università degli Studi di Salerno

Corso di Ingegneria del Software

MatchDay
Object Design Document
Versione 0.2

Data: 14/12/2024

Progetto: Nome Progetto	Versione: 0.2
Documento: Titolo Documento	Data: 14/12/2024

Coordinatore del progetto:

Nome	**Matricola**

Partecipanti:

Nome	**Matricola**
Vincenzo Vitale	0512113542
Nicola Moscufo	0512114886
Francesco Moscufo	0512115027

Scritto da:	Vincenzo Vitale, Nicola Moscufo, Francesco Moscufo

Revision History

Data	**Versione**	**Descrizione**	**Autore**
23/01/2025	0.2	Modifica delle class interfaces	Nicola Moscufo, Vincenzo Vitale, Francesco Moscufo

Sommario

1. INTRODUZIONE....4
1.1. Object design Trade-Off....4
1.2. Linee guida per la documentazione dell'interfaccia....4
1.3. Riferimenti....4
2. Directories....4

3. Design Patterns 19
4. Class Interfaces 20
5. Packages 30

1. INTRODUZIONE

L'Object Design Document (ODD) è un documento essenziale nel processo di sviluppo software. Tale documento si basa sui documenti RAD e SDD, che consolidano le informazioni raccolte durante l'analisi dei requisiti e la progettazione, l'ODD offre una guida dettagliata su come strutturare e implementare il sistema. Descrive le interfacce delle classi, le operazioni supportate, i tipi di dati utilizzati, i parametri delle procedure, i signature dei sottosistemi, trade-off . Questo documento include anche strategie per gestire compromessi di progettazione durante lo sviluppo, fungendo da "piano di costruzione" per il software.

1.1. Object design Trade-Off

- **Robustezza vs Velocità di implementazione**

  Nel gestire i dati in ingresso, sappiamo quanto sia importante controllarli bene. Tuttavia, per rilasciare la prima versione del sistema il più velocemente possibile, abbiamo deciso di accettare che i controlli iniziali non siano perfetti. Ci impegniamo comunque a migliorare questi controlli nelle versioni future del sistema. Questa scelta ci permette di uscire rapidamente con la prima versione, sapendo che rafforzeremo la robustezza nei prossimi aggiornamenti.

- **Chiarezza vs Velocità di implementazione**

  Per semplificare il testing, sarebbe ideale scrivere codice molto chiaro. Tuttavia, con i tempi di sviluppo stretti per questa prima versione, non sempre potremo mantenere i nostri standard abituali. Nelle versioni future del sistema, potremo migliorare questo aspetto.

1.2. Linee guida per la documentazione dell'interfaccia

- Le classi hanno dei nomi comuni singolari.

- I metodi sono denominati con frasi verbali.

- Gli Error Status sono restituiti attraverso eccezioni

1.3. Riferimenti

SDD: ci si riferisce al SDD quando si spiega l'organizzazione dei package, dato che quest'ultima è stata creata proprio a partire dalla suddivisione in sottosistemi.

2. Directories

Tutti file e le cartelle sono contenute all'interno della cartella lib creata autoamaticamente da Flutter quando si crea un nuovo progetto. Gli unici files che non sono contenuti in altre sottocartelle sono main.dart e firebase_options.dart.

```
lib/
├── main.dart
├── firebase_options.dart
│
├── Admin/
```

```
│   ├── admin_home.dart
│   ├── campoSelected.dart
│   ├── prenotazioni.dart
│   └── settings.dart
│
├── Components/
│   ├── adminNavbar.dart
│   ├── custom_snackbar.dart
│   ├── customTbCalendar.dart
│   └── userNavbar.dart
│
├── DAO/
│   └── auth_dao.dart
│
├── Models/
│   ├── campo.dart
│   ├── prenotazione.dart
│   ├── slot.dart
│   └── users.dart
│
├── Providers/
│   ├── authDaoProvider.dart
│   ├── prenotazioniProvider.dart
│   └── slotProvider.dart
│
├── Screens/
│   ├── login.dart
│   ├── register.dart
│   └── reset.dart
│
└── User/
    ├── editBooking.dart
    ├── prenotazioniUser.dart
    ├── selezionaCampo.dart
    ├── selezionaSlot.dart
    └── userSettings.dart
```

3. Design Patterns

Nell'implementazione sono stati usati diversi design patterns.

1. **Singleton**: usato per istanziare la classe di accesso a Firebase.
2. **Observer**: usato tramite l'interfaccia ChangeNotifier di Flutter.

3. **Facade**: usato per ricreare componenti personalizzati.

4. Class Interfaces

In questa sezione descriviamo le interfacce pubbliche delle classi. In Flutter si usa la convenzione dell'underscore "_", prima di un metodo o una variabile sta a significare che la visibilità' sarà' privata. Qui mostriamo solo i metodi pubblici delle classi con le rispettive informazioni come la descrizione di ogni metotdo, dipendenze associate, attributi pubblici e le Operazioni pubbliche (metodi). Ogni metodo avra'la sua descrizione, parametri, valore di ritorno, eccezioni sollevate, pre, post condizioni e invarianti.

AUTH MANAGEMENT

AuthDaoProvider	
Descrizione	La classe AuthDaoProvider è un provider per la gestione dell'autenticazione degli utenti utilizzando Firebase Authentication. Fornisce metodi per la creazione di account, il login, il logout, il reset della password, e altre operazioni correlate alla gestione dell'autenticazione.
Dipendenze	• AuthDao • Firebase_auth (package) • flutter/material.dart
Attributi Pubblici	AuthDao authdao
Operazioni Pubbliche	
createAccount()	
Descrizione	createAccount(email, password, ruolo, phone, nome, cognome, context, formKey): Void
Parametri	[email, password, ruolo, phone, nome, cognome, context, formKey]
Contesto	AuthDaoProvider::createAccount(email: String, password: String, ruolo: String, phone: String, nome: String, cognome: String, context: BuildContext, formKey: FormKey)
Valore di ritorno	Void
Eccezzioni sollevate	FirebaseAuthException, Exception

Pre-condizioni	• EmailValida: email <> null and email.matches("[a-zAZ0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}") • PasswordSicura: password <> null and password.size() >= 8 • FormValidato: formKey.isValid()
Post-condizioni	• AccountCreato: FirebaseAuth.instance.currentUser <> null implies FirebaseAuth.instance.currentUser.email = email

Invarianti	• inv AuthDaoInitialized: authdao <> null
Logout()	

Descrizione	Esegue il logout dell'utente dall'applicazione.
Contesto	AuthDaoProvider::logout(context: BuildContext)
Parametri	BuildContext context: Il contesto per il logout.
Valore di ritorno	Future<void>
Eccezzioni sollevate	Exception
Pre-condizioni	UtenteAutenticato: FirebaseAuth.instance.currentUser <> null
Post-condizioni	• UtenteDisconnesso: FirebaseAuth.instance.currentUser = null
Invarianti	• FirebaseAuthDisponibile: FirebaseAuth.instance <> null
signIn()	
Descrizione	Esegue il login dell'utente con le credenziali fornite.
Contesto	AuthDaoProvider::signIn(email: String, password: String, context: BuildContext)

Parametri	String email, String password, BuildContext context
Valore di ritorno	Future<void>
Eccezzioni sollevate	FirebaseAuthException, Exception
Pre-condizioni	• CredenzialiValide: email <> null and password <> null
Post-condizioni	• UtenteAutenticato: FirebaseAuth.instance.currentUser <> null implies FirebaseAuth.instance.currentUser.email = email
Invarianti	• FirebaseAuthValido: FirebaseAuth.instance <> null • AuthDaoInizializzato: authdao <> null
sendPasswordResetEmail()	
Descrizione	Invia un'email per il reset della password all'utente.
Contesto	AuthDaoProvider::sendPasswordResetEmail(email: String)
Parametri	String email
Valore di ritorno	Future<void>
Eccezzioni sollevate	FirebaseAuthException, Exception
Pre-condizioni	• EmailValida: email <> null and email.matches("[a-zAZ0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}")
Post-condizioni	• EmailInvitata: FirebaseAuth.instance.isPasswordResetEmailSent(email)
Invarianti	• FirebaseAuthDisponibile: FirebaseAuth.instance <> null

SLOTMANAGEMENT

FirebaseSlotProvider

Descrizione	La classe FirebaseSlotProvider gestisce le operazioni relative agli slot di prenotazione, come l'aggiunta, l'aggiornamento, il recupero e la rimozione degli slot su Firebase. Si appoggia al SlotDao per interagire con il database e utilizza il pattern ChangeNotifier per notificare i cambiamenti agli ascoltatori.
Dipendenze	• SlotDao: Classe responsabile della gestione dei dati su Firebase. • Slot: Modello che rappresenta uno slot di prenotazione. • Prenotazione: Modello che rappresenta una prenotazione. • ChangeNotifier: Notifica i cambiamenti agli ascoltatori.
Attributi pubblici	Nessuno
Operazioni Pubbliche	
fetchSlotsStream()	
Descrizione	Restituisce uno stream di slot per un determinato campo e giorno.
Contesto	FirebaseSlotProvider::fetchSlotsStream(campoId: String, selectedDay: DateTime)
Parametri	• String campoId: ID del campo sportivo. • DateTime selectedDay: Giorno per cui si vogliono recuperare gli slot.
Valore di ritorno	Stream<List<Slot>>

Eccezioni sollevate	Nessuna
Pre-condizioni	• CampoEsistente: self.database.contains(campoId) • DataEsistente: selectedDay <> null

Post-condizioni	StreamRestituito: result <> null and result->forAll(s | s.oclIsTypeOf(Slot))
add Slot()	
Descrizione	Aggiunge un nuovo slot per un campo e un giorno specifico su Firebase.
Contesto	FirebaseSlotProvider::addSlot(id: String, selectedDay: DateTime, slot: Slot)
Parametri	String id, DateTime selectedDay, Slot slot

Valore di ritorno	Future<void>
Eccezioni sollevate	Exception
Pre-condizioni	• CampoEsistente: self.database.contains(id) • SlotValido: slot <> null
Post-condizioni	SlotAggiunto: self.database.getSlots(id, selectedDay)->includes(slot)
RemoveSlot()	
Descrizione	Rimuove uno slot specifico da Firebase per un determinato campo e giorno
Contesto	FirebaseSlotProvider::removeSlot(campoId: String, selectedDay: DateTime, slot: Slot)
Parametri	String campoId, DateTime selectedDay, Slot slot
Valore di ritorno	Future<void>
Eccezioni sollevate	Exception
Pre-condizioni	• CampoEsistente: self.database.contains(campoId) • SlotEsistente: self.database.getSlots(campoId, selectedDay)->includes(slot)
Post-condizioni	• SlotRimosso: not self.database.getSlots(campoId, selectedDay)->includes(slot) • NotificaAscoltatori: self.notifyListeners()
Invarianti	• StrutturaDatabaseIntatta: self.checkDatabaseStructureIntegrity() • RimozioneNonInfluenzaAltriGiorni: self.database.getSlots(campoId, selectedDay + 1) = old(self.database.getSlots(campoId, selectedDay + 1))
GenerateHourlySlots()	
Descrizione	Genera slot orari per un campo in un intervallo di tempo specifico e li aggiunge a Firebase.
Contesto	FirebaseSlotProvider::generateHourlySlots(startHour: DateTime, endHour: DateTime, campoId: String, selectedDay: DateTime)
Parametri	• startHour: Ora di inizio (oggetto DateTime). • endHour: Ora di fine (oggetto DateTime). • campoId: ID del campo sportivo. • selectedDay: Giorno del calendario in cui gli slot devono essere generati.

Valore di ritorno	Future<void>
Eccezioni sollevate	Exception
Pre-condizioni	• StartHourPrimaDiEndHour: startHour < endHour • CampoEsistente: self.database.contains(campoId) • DataValida: selectedDay <> null
Post-condizioni	• SlotGenerati: self.database.getSlots(campoId, selectedDay)>size() = old(self.database.getSlots(campoId, selectedDay)->size()) + ((endHour.hour - startHour.hour) * numeroSlotPerOra) • NotifyListeners: self.notifyListeners()
Invarianti	• DatabaseStructureIntegrity: self.checkDatabaseStructureIntegrity() • NessunaSovrascritturaSlotEsistenti: self.database.getSlots(campoId, selectedDay)>forAll(s | old(self.database.getSlots(campoId, selectedDay)->includes(s)) implies self.database.getSlots(campoId, selectedDay)>includes(s))

PRENOTAZIONIMANAGEMENT

PrenotazioniProvider	
Descrizione	La classe PrenotazioniProvider gestisce le operazioni relative alle prenotazioni, come l'aggiunta, rimozione, accettazione, rifiuto, accetta modifica e rifiuta modifica. Si appoggia a PrenotazioniDao per interagire con il database e utilizza il pattern ChangeNotifier per notificare i cambiamenti ai listeners.
Dipendenze	• cloud_firestore • flutter/material.dart • PrenotazioniDao • ChangeNotifier
Attributi pubblici	Nessuno

Operazioni pubbliche

addPrenotazione()	
Descrizione	Aggiunge una nuova prenotazione nel database Firebase.
Contesto	PrenotazioniProvider::addPrenotazione(prenotazione: Prenotazione)
Parametri	prenotazione: Istanza di Prenotazione
Valore di ritorno	Future<void>
Eccezioni sollevate	Exception
Pre-condizioni	PrenotazioneValida: prenotazione <> null and prenotazione.isValid()
Post-condizioni	PrenotazioneSalvata: self.database.getPrenotazioni()->includes(prenotazione)
Invarianti	StrutturaDatabaseIntatta: self.checkDatabaseStructureIntegrity()
rifiutaPrenotazione()	
Descrizione	Rifiuta una prenotazione specificata aggiornandola su Firebase.
Contesto	PrenotazioniProvider::rifiutaPrenotazione(prenotazioneId: String, campoId: String, slotId: String, dataPrenotazione: String)
Parametri	String prenotazioneId, String campoId, String slotId, String dataPrenotazione
Valore di ritorno	Future<void>
Eccezioni sollevate	Exception
Pre-condizoni	PrenotazioneEsistente: self.database.getPrenotazioneById(prenotazioneId) <> null

Post-condizioni	• StatoRifiutato: self.database.getPrenotazioneById(prenotazioneId).stato = "rifiutata" • NotificaAscoltatori: self.notifyListeners()
Invarianti	NessunaModificaAltrePrenotazioni: self.database.getPrenotazioni()->forAll(p|p.id<> prenotazioneId implies p.stato = old(p.stato))
accettaPrenotazione()	
Descrizione	Accetta una prenotazione, aggiornando il suo stato a "confermata".
Contesto	PrenotazioniProvider::accettaPrenotazione(prenotazioneId: String)
Parametri	String prenotazioneId: ID della prenotazione da accettare.
Valore di ritorno	Future<void>

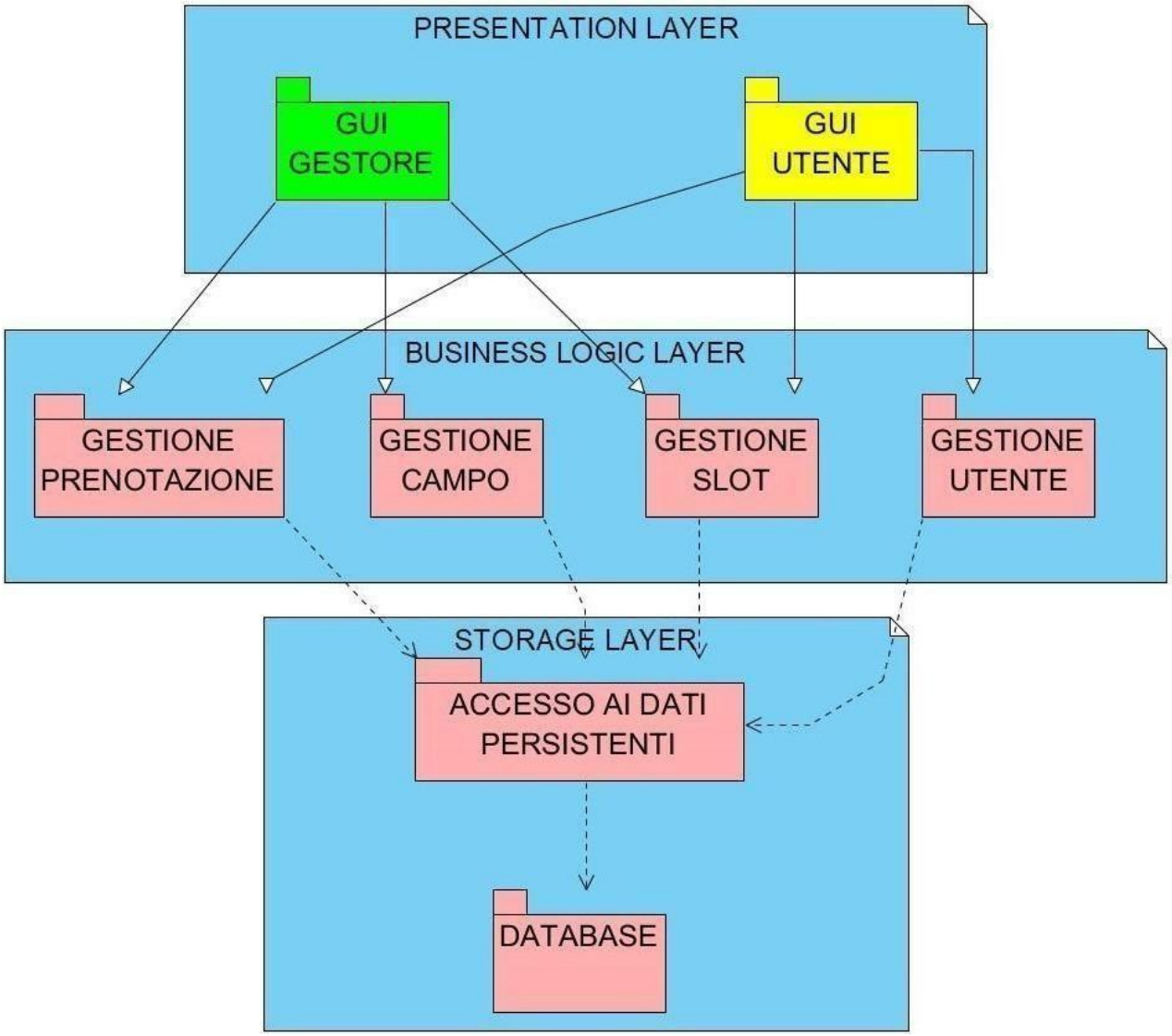
Eccezioni sollevate	Exception
Pre-condizioni	PrenotazioneEsistente: self.database.getPrenotazioneById(prenotazioneId) <> null
Post-condizioni	StatoConfermato: self.database.getPrenotazioneById(prenotazioneId).stato = "confermata"
Invarianti	NessunaModificaAltrePrenotazioni: self.database.getPrenotazioni()->forAll(p | p.id <> prenotazioneId implies p.stato = old(p.stato))
accettaModificaPrenotazione()	
Descrizione	Questo metodo accetta la richiesta di modifica della prenotazione. Aggiorna lo stato della prenotazione da ‘richiestaModifica’ a "confermata".
Contesto	PrenotazioniProvider::accettaModificaPrenotazione(id: String, idCampo: String, slotId: String, dataPrenotazione: String, orarioSlot: String)

Parametri	• id: Stringa che rappresenta l'ID della prenotazione. • idCampo: Stringa che rappresenta l'ID del campo associato alla prenotazione. • slotId: Stringa che rappresenta l'ID del nuovo slot selezionato. • dataPrenotazione: Stringa che rappresenta la data della prenotazione. • orarioSlot: Stringa che rappresenta l'orario del nuovo slot selezionato.
Valore di ritorno	Future<void>
Eccezioni sollevate	FirebaseException, Exception
Pre-condizioni	• StatoRichiestaModifica: self.database.getPrenotazioneById(id).stato = "richiestaModifica"
Post-condizioni	• PrenotazioneAggiornata: self.database.getPrenotazioneById(id).slotId = slotId and self.database.getPrenotazioneById(id).stato = "confermata"
Invarianti	• StatoPrenotazione: self.database.getPrenotazioneById(id).stato = "confermata" or old(self.database.getPrenotazioneById(id).stato) = "richiestaModifica"
rifiutaModificaPrenotazione()	
Descrizione	Rifiuta la richiesta di modifica della prenotazione, lo slot viene reso disponibile e la prenotazione cambia in stato ‘annullata’
Contesto	PrenotazioniProvider::rifiutaModificaPrenotazione(id: String)
Parametri	String id: Stringa che rappresenta l'ID della prenotazione.
Valore di ritorno	Future<void>

Eccezioni sollevate	FirebaseException, Exception
Pre-condizioni	• StatoRichiestaModifica: self.database.getPrenotazioneById(id).stato = "richiestaModifica" • PrenotazioneEsistente: self.database.getPrenotazioneById(id) <> null
Post-condizioni	• StatoRipristinato: self.database.getPrenotazioneById(id).stato = old(self.database.getPrenotazioneById(id).stato) • NessunaModificaSlot: self.database.getPrenotazioneById(id).slotId = old(self.database.getPrenotazioneById(id).slotId)
Invarianti	• StatoPrenotazioneCoerente: self.database.getPrenotazioneById(id).stato = old(self.database.getPrenotazioneById(id).stato) or self.database.getPrenotazioneById(id).stato = "annullata"

5. Packages

Il sistema è diviso nei seguenti sottosistemi



A partire da questo sottosistema sono stati quindi prodotti i seguenti packages:

- **Gestione Prenotazione**: Questo package si occupa di tutte le operazioni legate alla gestione delle prenotazioni. Include la creazione, la modifica, l'eliminazione e la visualizzazione delle prenotazioni. In questa sezione, vengono gestiti anche gli stati delle prenotazioni, come "in attesa", "confermata", "annullata" e altre transizioni possibili, come la modifica degli slot prenotati. Inoltre, si occupa di gestire la logica di validazione per la prenotazione (ad esempio, verificare se uno slot è disponibile) e può interagire con il sistema di backend per archiviare e recuperare le prenotazioni dal database.
- **Gestione Campo**: Questo package è dedicato alla gestione delle informazioni relative ai campi sportivi. Gestisce l'inserimento, la modifica e la visualizzazione delle informazioni relative ai campi disponibili per la prenotazione. Può includere la gestione di dettagli come il nome del campo, i suoi orari di disponibilità tramite il calendario.

- **Gestione Slot**: Il package di gestione degli slot si occupa della creazione, visualizzazione, modifica e gestione della disponibilità degli slot di prenotazione per i vari campi. Gestisce anche lo stato degli slot (disponibili, non disponibili, prenotati, ecc.) e garantisce che gli utenti possano prenotare solo slot che sono effettivamente disponibili. Si interfaccia anche con il sistema di backend per aggiornare lo stato degli slot in tempo reale e per recuperare gli slot disponibili in base alla data selezionata.
- **Gestione Utente**: Questo package si occupa della gestione delle informazioni relative agli utenti. Include funzionalità per il login, la registrazione, e la gestione dei permessi (ad esempio, determinando se un utente è un gestore o un utente normale). Gestisce anche la gestione della sessione dell'utente, come il logout, e offre metodi per il recupero delle credenziali nel caso in cui l'utente dimentichi la password.
- **Accesso ai Dati Persistenti**: Questo package è responsabile dell'accesso ai dati persistenti, come il recupero e l'archiviazione delle informazioni nel database (Firestore nel nostro caso). Gestisce le operazioni CRUD (Create, Read, Update, Delete) per le prenotazioni, gli utenti, i campi e gli slot, interfacciandosi direttamente con il database per garantire che tutte le informazioni siano correttamente memorizzate e recuperabili in modo efficiente.