Università degli Studi di Salerno
Corso di Ingegneria del Software

MatchDay
Object Design Document
Versione 0.1

MatchDay

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**

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 5
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.

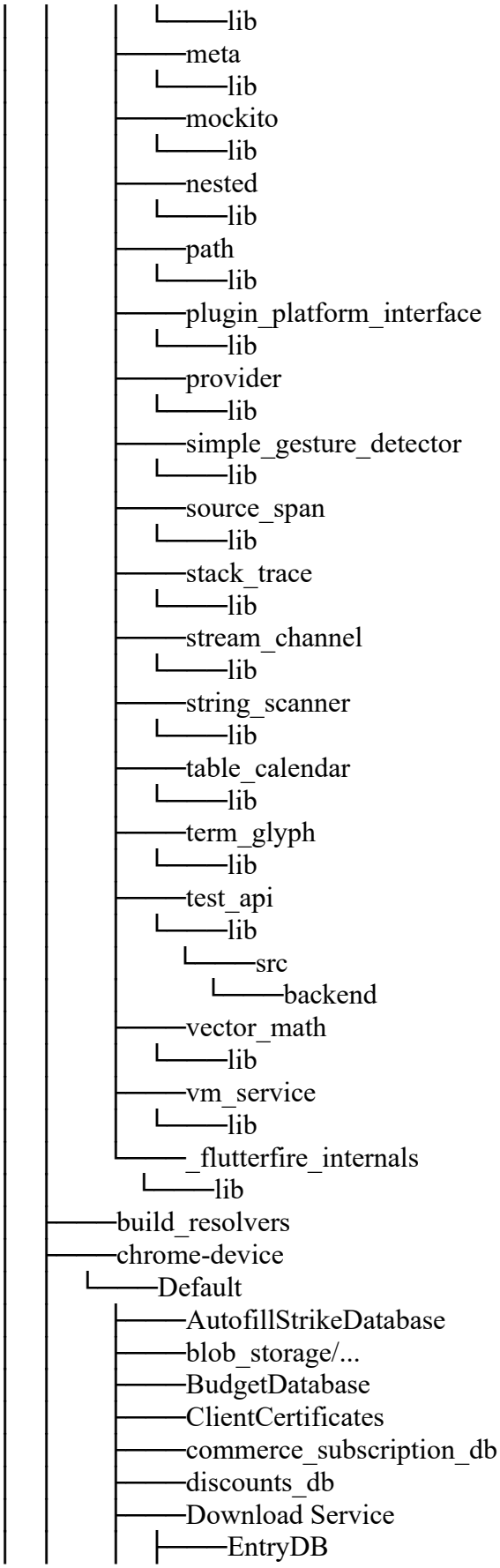
2. Directories

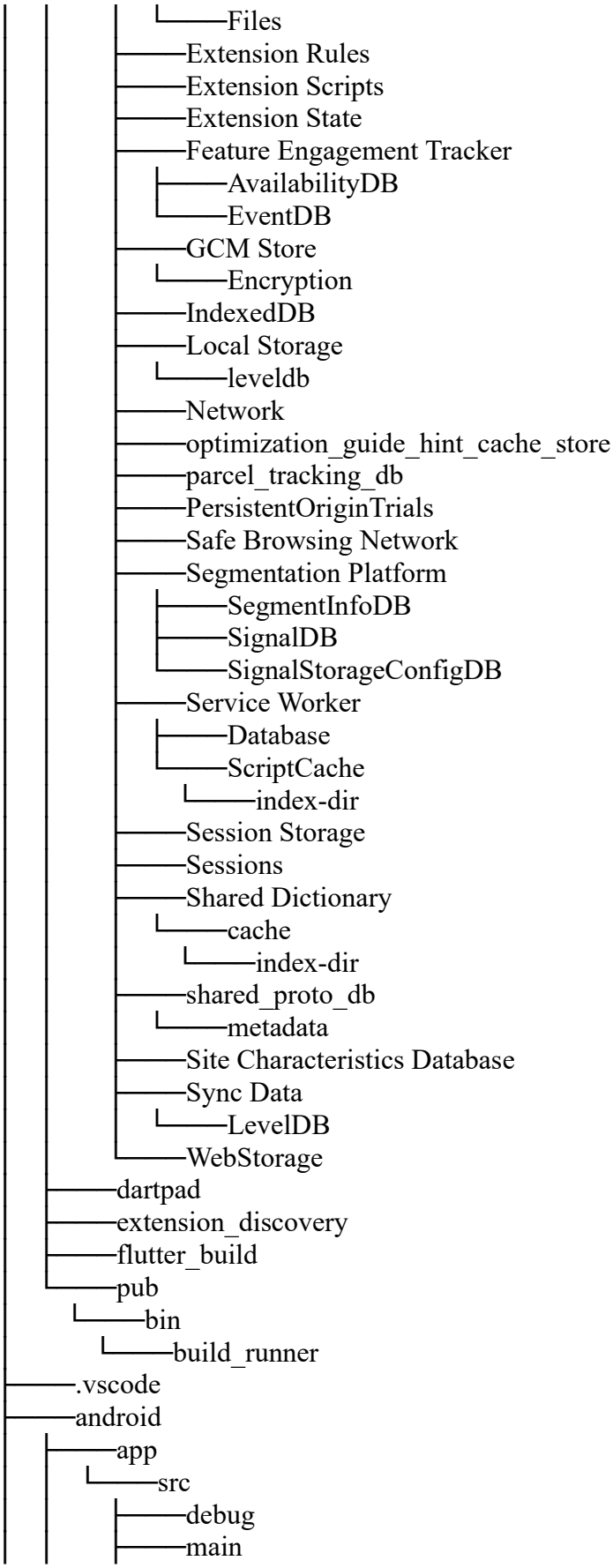
Tutti file e le cartelle sono contenute all'interno della cartella lib creata autoamaticamente da Flutter quando si crea un nuovo progetto. Gli unici files che non sono contenuti in altre sottocartelle sono main.dart e firebase_options.dart.

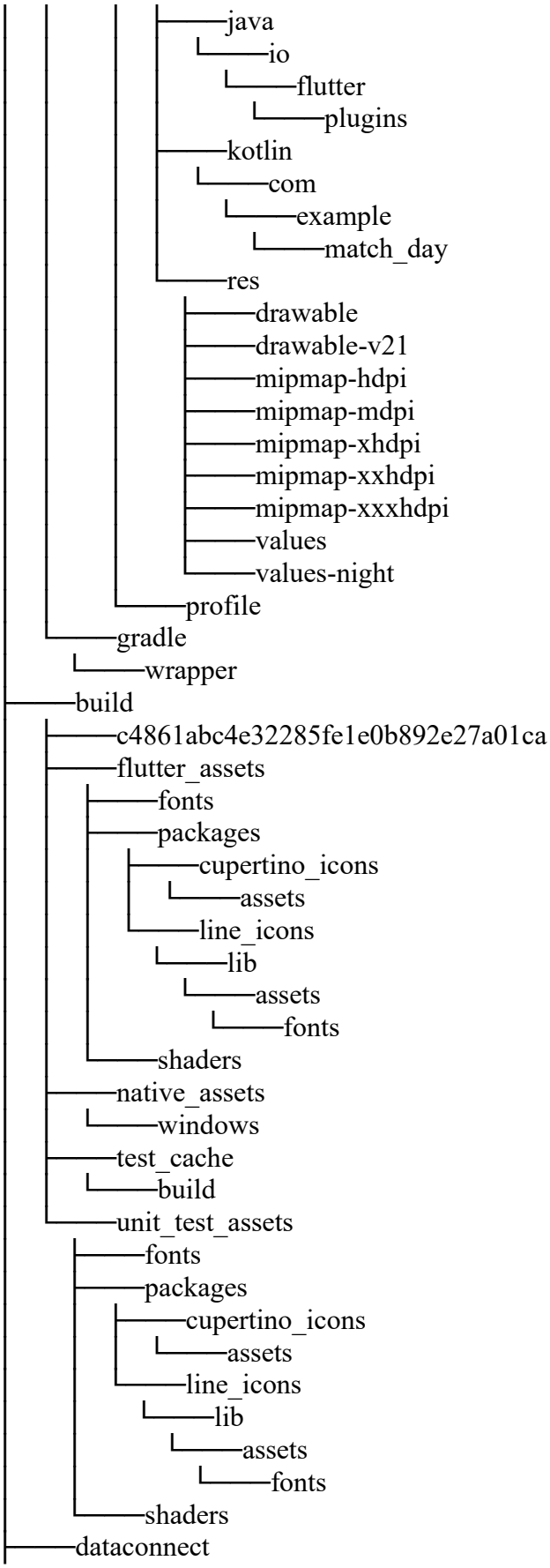
```
lib/
├── main.dart
├── firebase_options.dart
│
├── Admin/
│   ├── admin_home.dart
│   ├── campoSelected.dart
│   ├── prenotazioni.dart
│   └── settings.dart
│
├── Components/
│   ├── adminNavbar.dart
│   ├── custom_snackbar.dart
│   ├── customTbCalendar.dart
│   └── userNavbar.dart
│
├── DAO/
│   └── 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
```

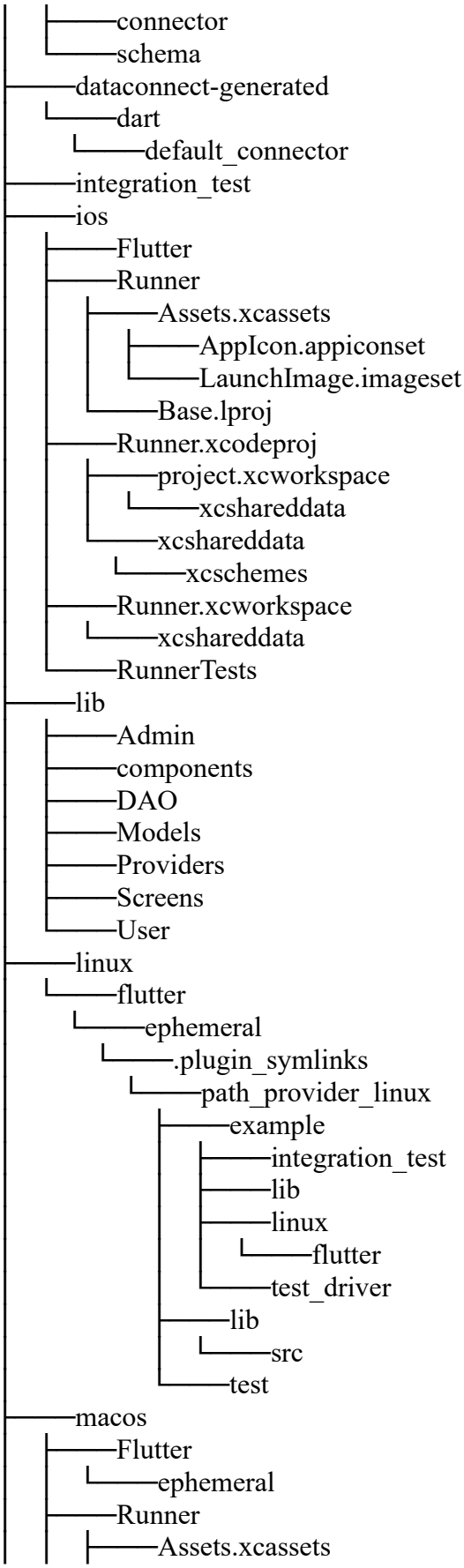
Questa invece e' la struttura completa del progetto:

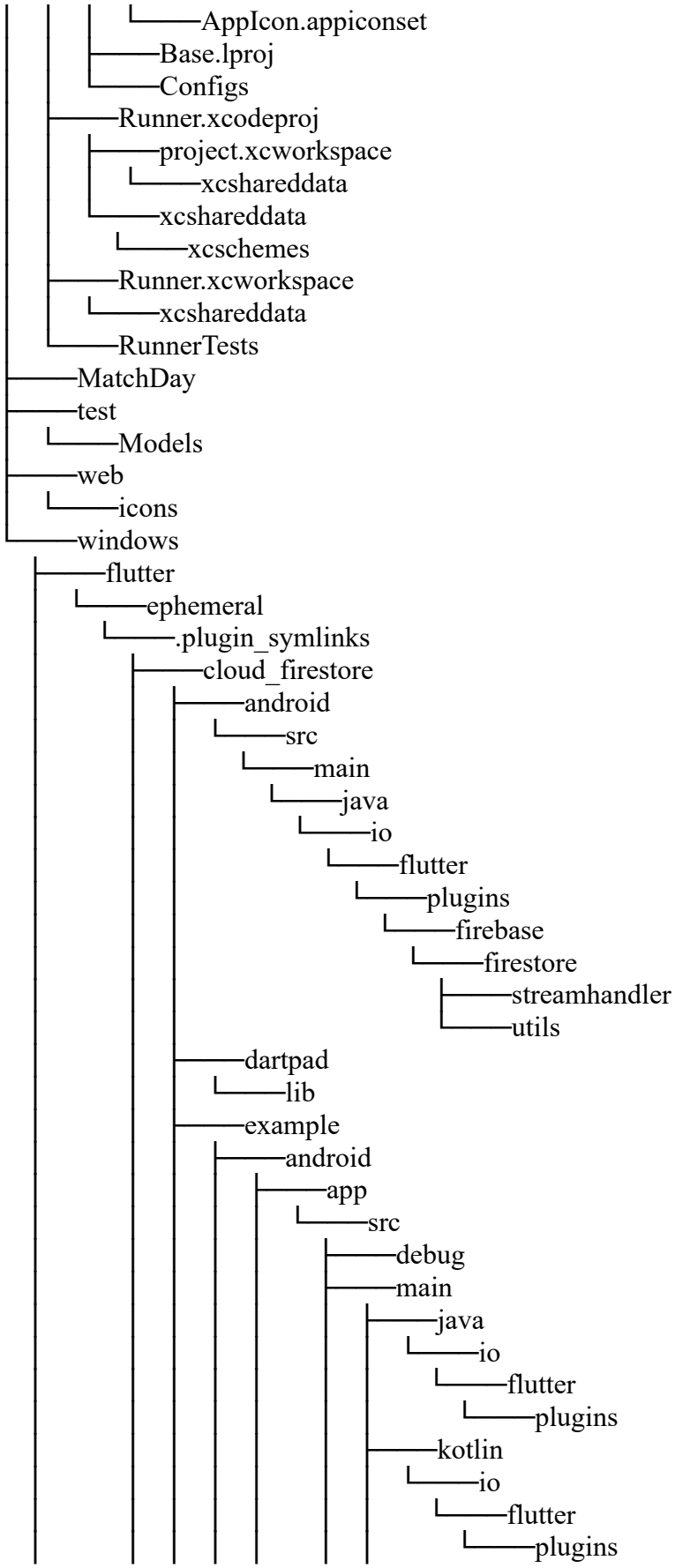
```
C:.
├───.dart_tool
│   ├───build
│   │   ├───entrypoint
│   │   ├───fcd1995bc647fb959e82ea360c6c2c9a
│   │   └───generated
│   │       ├───async
│   │       │   └───lib
│   │       ├───boolean_selector
│   │       │   └───lib
│   │       ├───characters
│   │       │   └───lib
│   │       ├───clock
│   │       │   └───lib
│   │       ├───cloud_firestore
│   │       │   └───lib
│   │       ├───cloud_firestore_platform_interface
│   │       │   └───lib
│   │       ├───collection
│   │       │   └───lib
│   │       ├───fake_async
│   │       │   └───lib
│   │       ├───firebase_auth
│   │       │   └───lib
│   │       ├───firebase_auth_platform_interface
│   │       │   └───lib
│   │       ├───firebase_core
│   │       │   └───lib
│   │       ├───firebase_core_platform_interface
│   │       │   └───lib
│   │       ├───flutter
│   │       │   └───lib
│   │       ├───flutter_test
│   │       │   └───lib
│   │       ├───intl
│   │       │   └───lib
│   │       ├───leak_tracker
│   │       │   └───lib
│   │       ├───leak_tracker_flutter_testing
│   │       │   └───lib
│   │       ├───leak_tracker_testing
│   │       │   └───lib
│   │       ├───matcher
│   │       │   └───lib
│   │       │       └───src
│   │       │           └───expect
│   │       ├───material_color_utilities
```

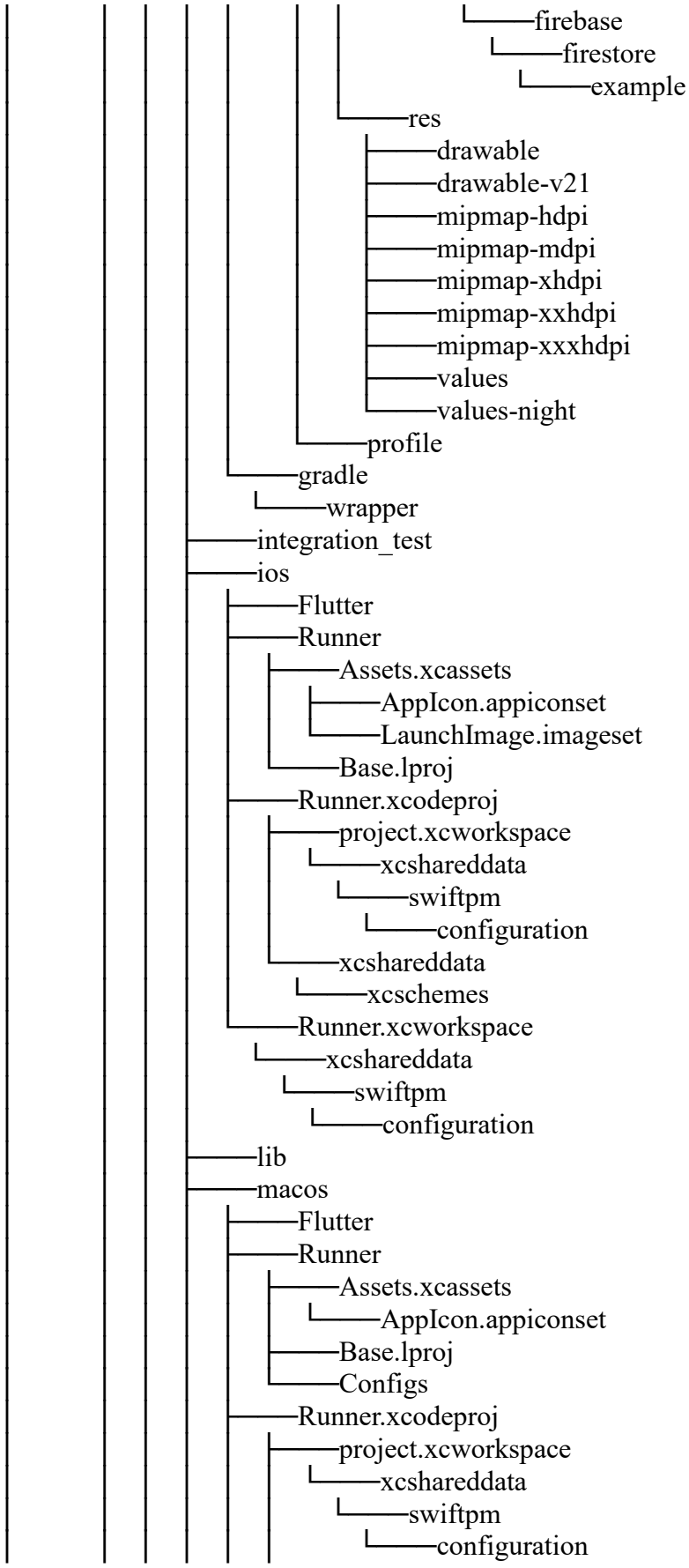

lib
meta
lib
mockito
lib
nested
lib
path
lib
plugin_platform_interface
lib
provider
lib
simple_gesture_detector
lib
source_span
lib
stack_trace
lib
stream_channel
lib
string_scanner
lib
table_calendar
lib
term_glyph
lib
test_api
lib
src
backend
vector_math
lib
vm_service
lib
_flutterfire_internals
lib
build_resolvers
chrome-device
Default
AutofillStrikeDatabase
blob_storage/...
BudgetDatabase
ClientCertificates
commerce_subscription_db
discounts_db
Download Service
EntryDB

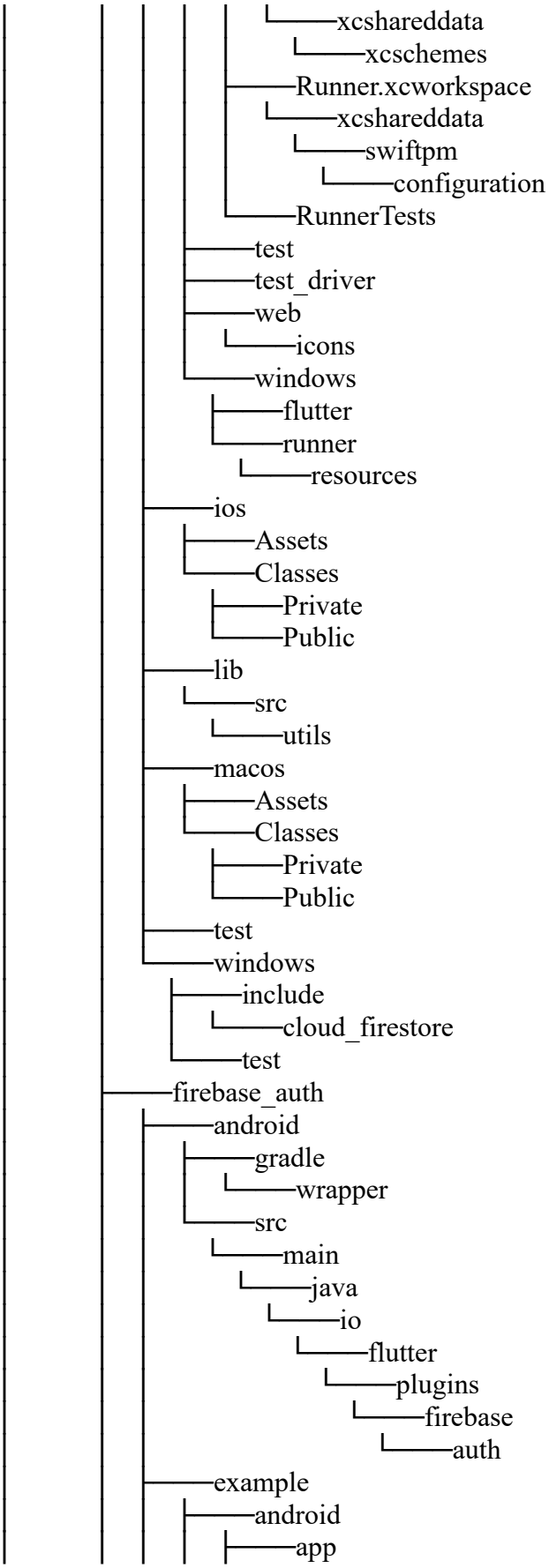

Files
Extension Rules
Extension Scripts
Extension State
Feature Engagement Tracker
AvailabilityDB
EventDB
GCM Store
Encryption
IndexedDB
Local Storage
leveldb
Network
optimization_guide_hint_cache_store
parcel_tracking_db
PersistentOriginTrials
Safe Browsing Network
Segmentation Platform
SegmentInfoDB
SignalDB
SignalStorageConfigDB
Service Worker
Database
ScriptCache
index-dir
Session Storage
Sessions
Shared Dictionary
cache
index-dir
shared_proto_db
metadata
Site Characteristics Database
Sync Data
LevelDB
WebStorage
dartpad
extension_discovery
flutter_build
pub
bin
build_runner
.vscode
android
app
src
debug
main

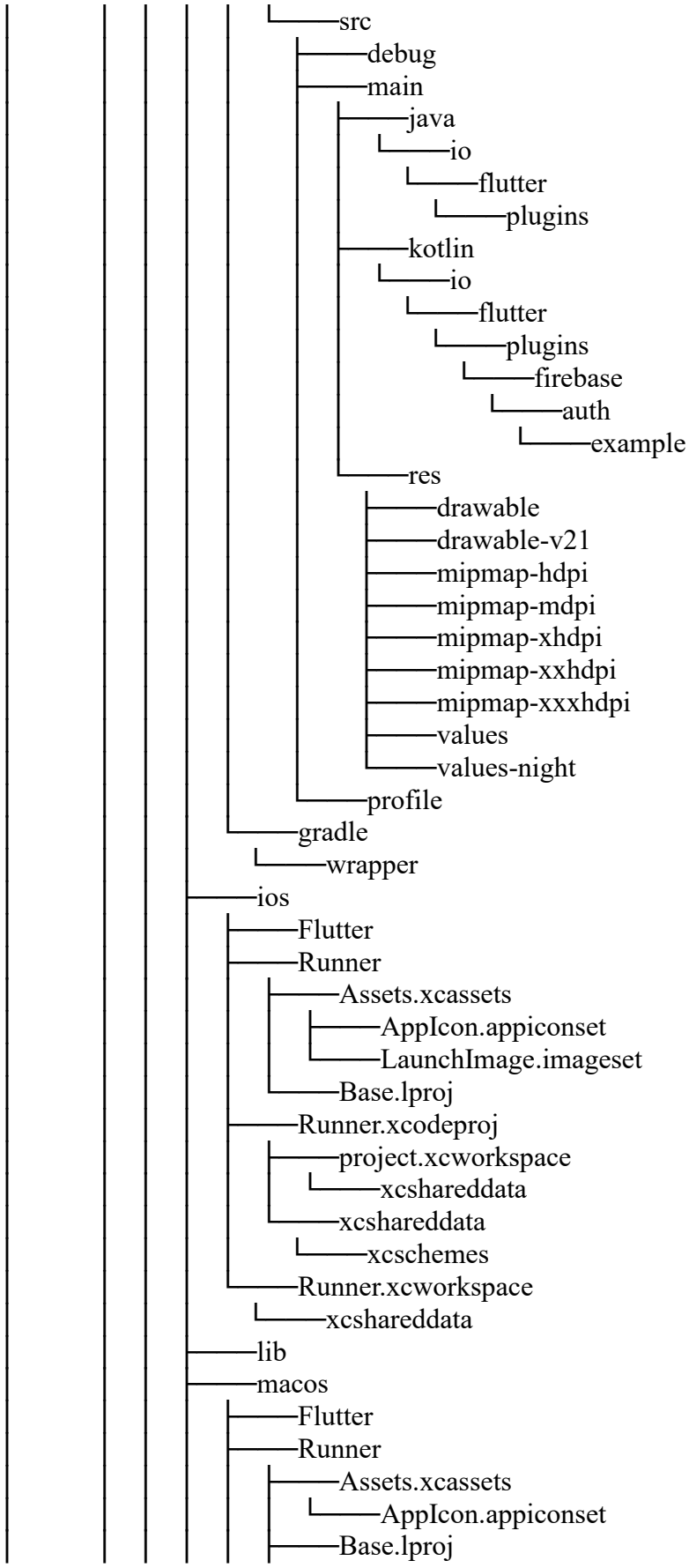

java
io
flutter
plugins
kotlin
com
example
match_day
res
drawable
drawable-v21
mipmap-hdpi
mipmap-mdpi
mipmap-xhdpi
mipmap-xxhdpi
mipmap-xxxhdpi
values
values-night
profile
gradle
wrapper
build
c4861abc4e32285fe1e0b892e27a01ca
flutter_assets
fonts
packages
cupertino_icons
assets
line_icons
lib
assets
fonts
shaders
native_assets
windows
test_cache
build
unit_test_assets
fonts
packages
cupertino_icons
assets
line_icons
lib
assets
fonts
shaders
dataconnect

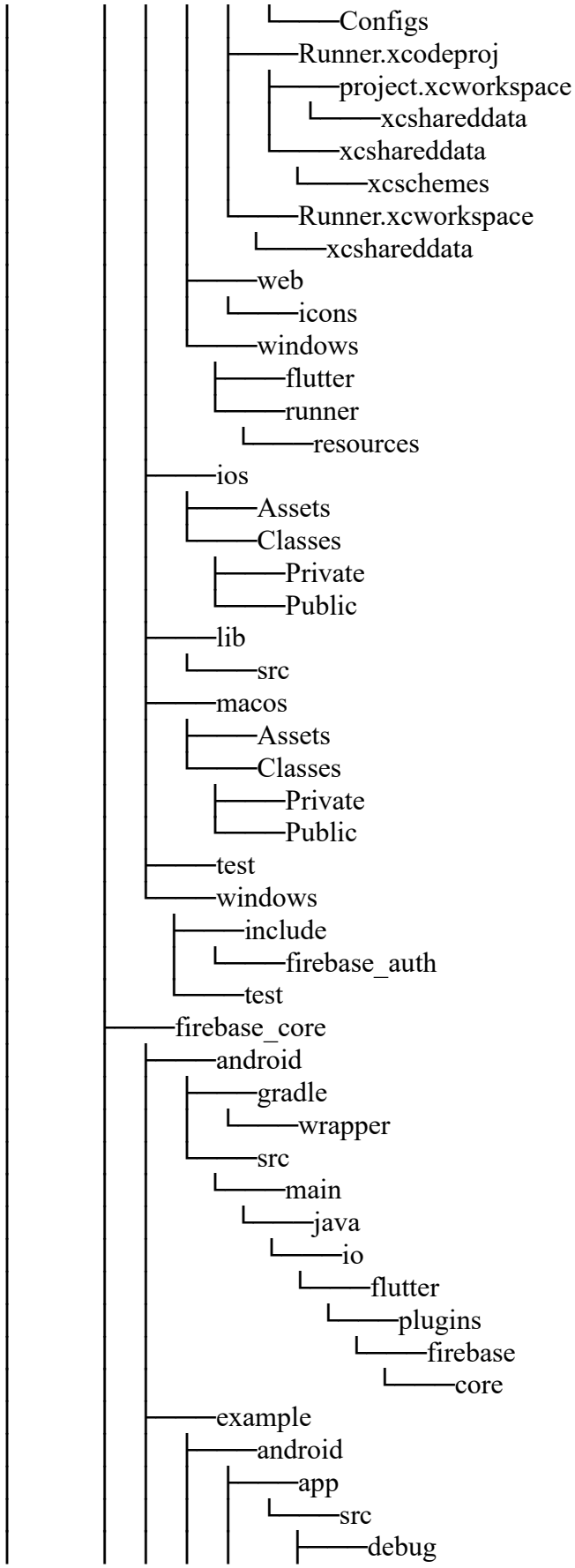

connector
schema
dataconnect-generated
dart
default_connector
integration_test
ios
Flutter
Runner
Assets.xcassets
AppIcon.appiconset
LaunchImage.imageset
Base.lproj
Runner.xcodeproj
project.xcworkspace
xcshareddata
xcshareddata
xcschemes
Runner.xcworkspace
xcshareddata
RunnerTests
lib
Admin
components
DAO
Models
Providers
Screens
User
linux
flutter
ephemeral
.plugin_symlinks
path_provider_linux
example
integration_test
lib
linux
flutter
test_driver
lib
src
test
macos
Flutter
ephemeral
Runner
Assets.xcassets

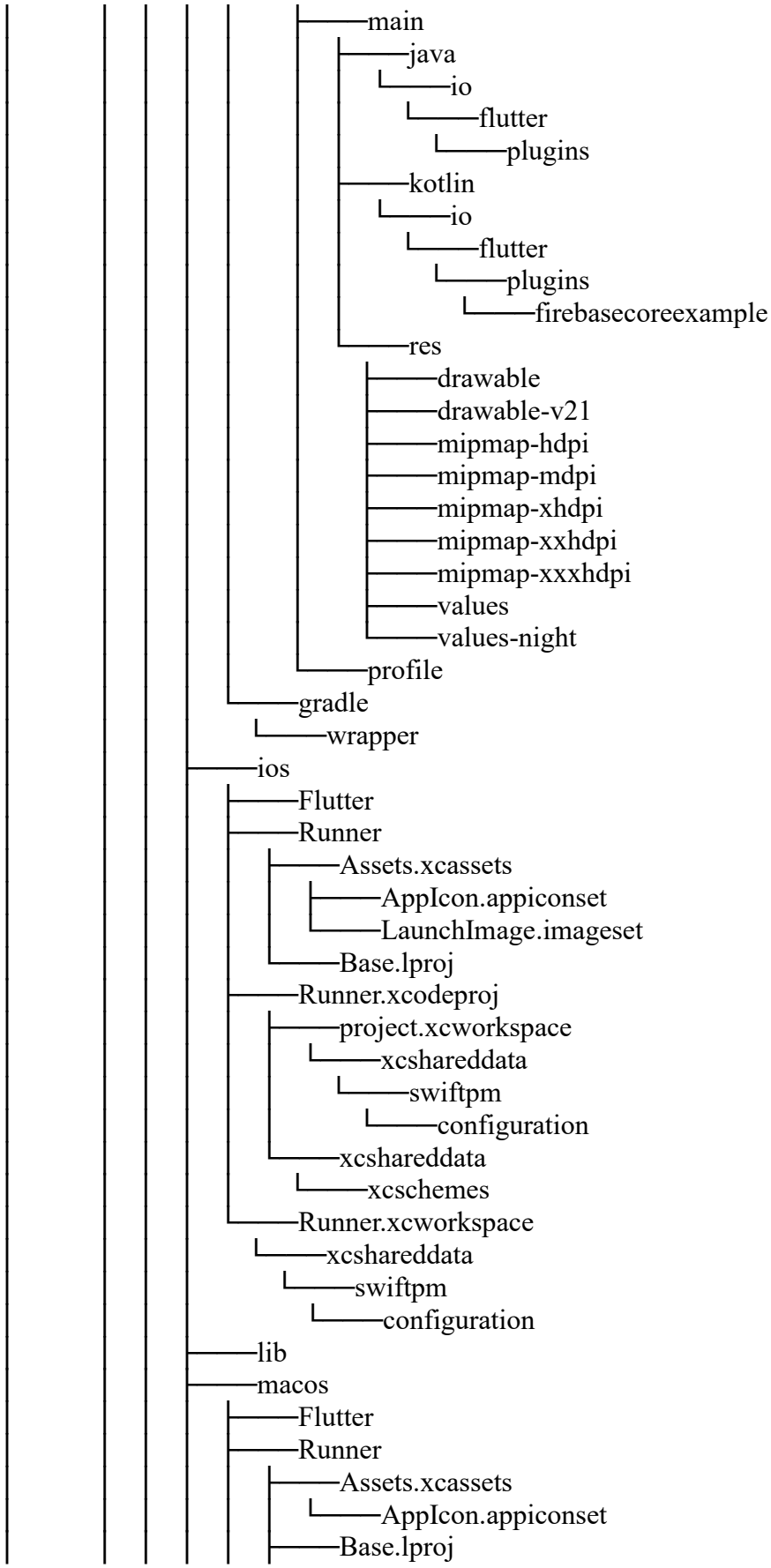

AppIcon.appiconset
Base.lproj
Configs
Runner.xcodeproj
project.xcworkspace
xcshareddata
xcshareddata
xcschemes
Runner.xcworkspace
xcshareddata
RunnerTests
MatchDay
test
Models
web
icons
windows
flutter
ephemeral
.plugin_symlinks
cloud_firestore
android
src
main
java
io
flutter
plugins
firebase
firestore
streamhandler
utils
dartpad
lib
example
android
app
src
debug
main
java
io
flutter
plugins
kotlin
io
flutter
plugins

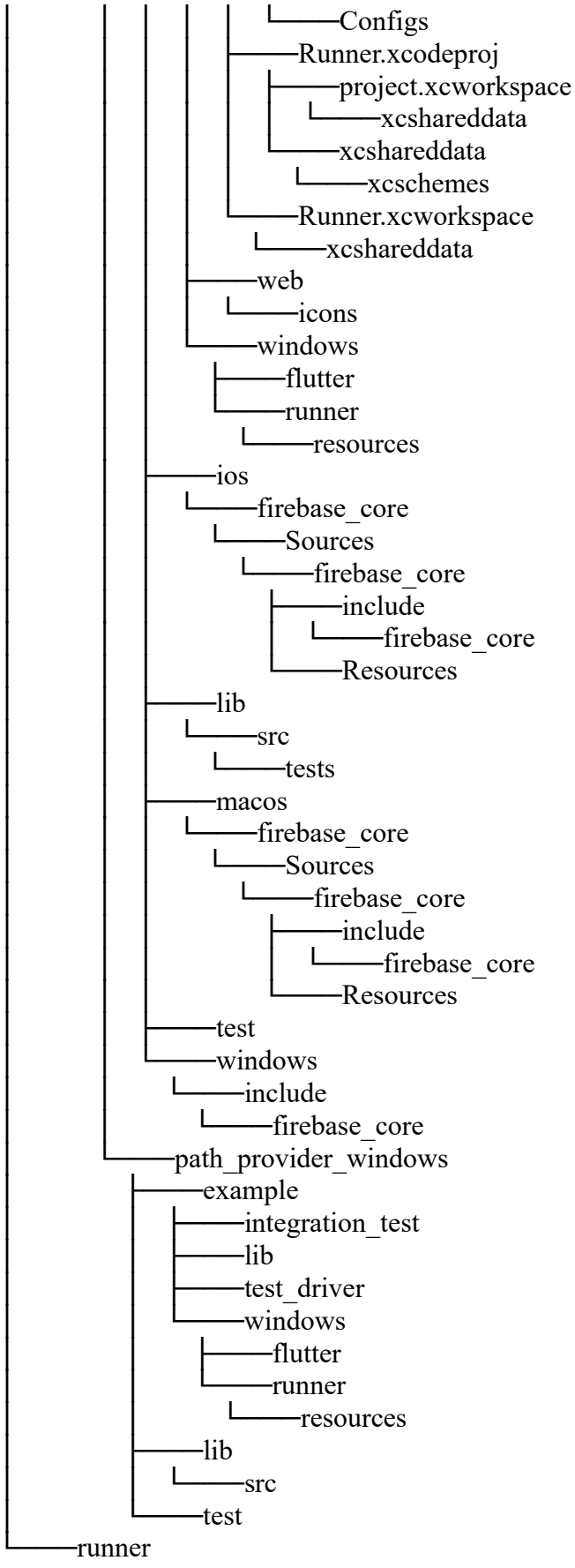

firebase
firestore
example
res
drawable
drawable-v21
mipmap-hdpi
mipmap-mdpi
mipmap-xhdpi
mipmap-xxhdpi
mipmap-xxxhdpi
values
values-night
profile
gradle
wrapper
integration_test
ios
Flutter
Runner
Assets.xcassets
AppIcon.appiconset
LaunchImage.imageset
Base.lproj
Runner.xcodeproj
project.xcworkspace
xcshareddata
swiftpm
configuration
xcshareddata
xcschemes
Runner.xcworkspace
xcshareddata
swiftpm
configuration
lib
macos
Flutter
Runner
Assets.xcassets
AppIcon.appiconset
Base.lproj
Configs
Runner.xcodeproj
project.xcworkspace
xcshareddata
swiftpm
configuration


xcshareddata
xcschemes
Runner.xcworkspace
xcshareddata
swiftpm
configuration
RunnerTests
test
test_driver
web
icons
windows
flutter
runner
resources
ios
Assets
Classes
Private
Public
lib
src
utils
macos
Assets
Classes
Private
Public
test
windows
include
cloud_firestore
test
firebase_auth
android
gradle
wrapper
src
main
java
io
flutter
plugins
firebase
auth
example
android
app


src
debug
main
java
io
flutter
plugins
kotlin
io
flutter
plugins
firebase
auth
example
res
drawable
drawable-v21
mipmap-hdpi
mipmap-mdpi
mipmap-xhdpi
mipmap-xxhdpi
mipmap-xxxhdpi
values
values-night
profile
gradle
wrapper
ios
Flutter
Runner
Assets.xcassets
AppIcon.appiconset
LaunchImage.imageset
Base.lproj
Runner.xcodeproj
project.xcworkspace
xcshareddata
xcshareddata
xcschemes
Runner.xcworkspace
xcshareddata
lib
macos
Flutter
Runner
Assets.xcassets
AppIcon.appiconset
Base.lproj


Configs
Runner.xcodeproj
project.xcworkspace
xcshareddata
xcshareddata
xcschemes
Runner.xcworkspace
xcshareddata
web
icons
windows
flutter
runner
resources
ios
Assets
Classes
Private
Public
lib
src
macos
Assets
Classes
Private
Public
test
windows
include
firebase_auth
test
firebase_core
android
gradle
wrapper
src
main
java
io
flutter
plugins
firebase
core
example
android
app
src
debug


main
java
io
flutter
plugins
kotlin
io
flutter
plugins
firebasecoreexample
res
drawable
drawable-v21
mipmap-hdpi
mipmap-mdpi
mipmap-xhdpi
mipmap-xxhdpi
mipmap-xxxhdpi
values
values-night
profile
gradle
wrapper
ios
Flutter
Runner
Assets.xcassets
AppIcon.appiconset
LaunchImage.imageset
Base.lproj
Runner.xcodeproj
project.xcworkspace
xcshareddata
swiftpm
configuration
xcshareddata
xcschemes
Runner.xcworkspace
xcshareddata
swiftpm
configuration
lib
macos
Flutter
Runner
Assets.xcassets
AppIcon.appiconset
Base.lproj


Configs
Runner.xcodeproj
project.xcworkspace
xcshareddata
xcshareddata
xcschemes
Runner.xcworkspace
xcshareddata
web
icons
windows
flutter
runner
resources
ios
firebase_core
Sources
firebase_core
include
firebase_core
Resources
lib
src
tests
macos
firebase_core
Sources
firebase_core
include
firebase_core
Resources
test
windows
include
firebase_core
path_provider_windows
example
integration_test
lib
test_driver
windows
flutter
runner
resources
lib
src
test
runner

└———resources

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.

DAO

AUTHDAO

Descrizione	La classe `AuthDao` è una classe che gestisce l'autenticazione degli utenti e le operazioni correlate utilizzando Firebase Authentication e Firestore. È progettata per fornire metodi per la creazione di account, il login, il reset della password e altre funzionalità relative alla gestione degli utenti.
Invariante	currentUser != null & email != null & phone != null & ruolo != null & nome != null &cognome != null user-id != null ruolo == 'admin' se l'utente e' admin ruolo == 'user' se l'utente e' un user

createAccount	context AuthDao::createAccount(email: String, password: String, phone: String, nome: String, cognome: String, ruolo: String, context: BuildContext, formKey: GlobalKey<FormState>) pre: email <> null & password <> null & phone <> null & nome <> null & cognome <> null & ruolo <> null & context <> null & formKey <> null ---------- context AuthDao::createAccount(email: String, password: String, phone: String, nome: String, cognome: String, ruolo: String, context: BuildContext, formKey: GlobalKey<FormState>) post: userCredential.user.email = email & userCredential.user.password = password & userCredential.user.phone = phone & userCredential.user.nome = nome & userCredential.user.cognome = cognome & userCredential.user.ruolo = ruolo & userCredential.user.uid <> null & Firestore.collection('users').doc(userCredential.user.uid).set({ 'email': email, 'phone': phone, 'Ruolo': ruolo, 'nome': nome, 'cognome': cognome, 'user-id': userCredential.user.uid }) & formKey.currentState!.reset() & Navigator.pushReplacement(context, MaterialPageRoute(builder: (context) => const Login())) & CustomSnackbar.show(context, "Utente creato e salvato in Firestore.")

login	context AuthDao::login(email: String, password: String, context: BuildContext) pre: email <> null & password <> null & context <> null ---------- context AuthDao::login(email: String, password: String, context: BuildContext) post: userCredential.user.email = email & userCredential.user.password = password & Firestore.collection('users').doc(userCredential.user.uid).get() -> DocumentSnapshot userDoc & userDoc['Ruolo'] <> null & (userDoc['Ruolo'] == 'admin' => Navigator.pushReplacement(context, MaterialPageRoute(builder: (context) => const AdminHomePage()))) & (userDoc['Ruolo'] == 'user' => Navigator.pushReplacement(context, MaterialPageRoute(builder: (context) => const CampoSelectionPage()))) & CustomSnackbar.show(context, "Login effettuato con successo.")
resetPassword	context AuthDao::resetPassword(email: String, context: BuildContext) pre: email <> null & context <> null ---------- context AuthDao::resetPassword(email: String, context: BuildContext) post: FirebaseAuth.instance.sendPasswordResetEmail(email: email) & CustomSnackbar.show(context, "Email di reset inviata a $email.")

PRENOTAZIONIDAO

Descrizione	La classe `PrenotazioniDao` è responsabile della gestione delle operazioni di lettura, scrittura e aggiornamento delle prenotazioni nel database Firestore. È progettata per interagire con la

	collezione `prenotazionie` gestire le prenotazioni degli utenti.
Invariante	currentUser != null & email != null & phone != null & ruolo != null & nome != null & cognome != null & user-id != null ruolo == 'admin' se l'utente è un amministratore. ruolo == 'user' se l'utente è un utente normale. slot != null & slot.disponibile == true prenotazione-id != null stato == 'inAttesa' se la prenotazione è in attesa di approvazione. stato == 'accettata' se la prenotazione è stata approvata. stato == 'rifiutata' se la prenotazione è stata rifiutata. stato == 'richiestaModifica' se la prenotazione è stata modificata dall'utente e la modifica è in attesa di approvazione. slot.prenotato == false

createPrenotazione	context PrenotazioneDao::createPrenotazione(Prenotazione prenotazione, BuildContext context)
	pre:
	  prenotazione <> null
	  & prenotazione.campoId <> null
	  & prenotazione.slotId <> null
	  & prenotazione.data <> null
	  & prenotazione.stato <> null
	  & context <> null
	--- 	
	context PrenotazioneDao::createPrenotazione(Prenotazione prenotazione, BuildContext context)
	post:
	  Firestore.collection('prenotazioni').add({
	  'campoId': prenotazione.campoId,

	'slotId': prenotazione.slotId, 'data': prenotazione.data, 'stato': prenotazione.stato, 'userId': currentUser.uid }) & CustomSnackbar.show(context, "Prenotazione effettuata con successo.")
getPrenotazioniUtente	context PrenotazioneDao::getPrenotazioniUtente(String userId, BuildContext context) pre: userId <> null & context <> null --- context PrenotazioneDao::getPrenotazioniUtente(String userId, BuildContext context) post: prenotazioniList = Firestore.collection('prenotazioni').where('userId', isEqualTo: userId).get() & prenotazioniList <> null & (prenotazioniList.length > 0 => CustomSnackbar.show(context, "Prenotazioni recuperate con successo.")) & (prenotazioniList.length == 0 => CustomSnackbar.show(context, "Nessuna prenotazione trovata."))
modificaPrenotazione	context PrenotazioneDao::modificaPrenotazione(String prenotazioneId, Prenotazione nuovaPrenotazione, BuildContext context) pre: prenotazioneId <> null & nuovaPrenotazione <> null & nuovaPrenotazione.campoId <> null & nuovaPrenotazione.slotId <> null & nuovaPrenotazione.data <> null & nuovaPrenotazione.stato <> null & context <> null --- context PrenotazioneDao::modificaPrenotazione(String prenotazioneId, Prenotazione nuovaPrenotazione, BuildContext context) post:

	Firestore.collection('prenotazioni').doc(prenotazioneId).update({ 'campoId': nuovaPrenotazione.campoId, 'slotId': nuovaPrenotazione.slotId, 'data': nuovaPrenotazione.data, 'stato': nuovaPrenotazione.stato }) & CustomSnackbar.show(context, "Prenotazione modificata con successo.")
annullaPrenotazione	context PrenotazioneDao::annullaPrenotazione(String prenotazioneId, BuildContext context) pre: prenotazioneId <> null & context <> null --- context PrenotazioneDao::annullaPrenotazione(String prenotazioneId, BuildContext context) post: Firestore.collection('prenotazioni').doc(prenotazioneId).delete() & CustomSnackbar.show(context, "Prenotazione annullata con successo.")
accettaPrenotazione	context PrenotazioneDao::accettaPrenotazione(String prenotazioneId, BuildContext context) pre: prenotazioneId <> null & context <> null --- context PrenotazioneDao::accettaPrenotazione(String prenotazioneId, BuildContext context) post: Firestore.collection('prenotazioni').doc(prenotazioneId).update({ 'stato': 'confermata' }) & CustomSnackbar.show(context, "Prenotazione accettata.")
rifiutaPrenotazione	context PrenotazioneDao::rifiutaPrenotazione(String prenotazioneId, BuildContext context) pre:

	prenotazioneId <> null & context <> null ---------- context PrenotazioneDao::rifiutaPrenotazione(String prenotazioneId, BuildContext context) post: Firestore.collection('prenotazioni').doc(prenotazioneId).update({ 'stato': 'rifiutata' }) & CustomSnackbar.show(context, "Prenotazione rifiutata.")

SLOTDAO

Descrizione	La classe SlotDao gestisce le operazioni relative agli slot di prenotazione per il campo sportivo. Essa interagisce con Firebase Firestore per aggiungere, rimuovere, aggiornare e recuperare gli slot di un campo sportivo per una data specifica. Gli slot vengono utilizzati per indicare gli orari disponibili per le prenotazioni e per gestire la loro disponibilità (disponibile o meno).
Invariante	currentUser != null & email != null & phone != null & ruolo != null & nome != null & cognome != null & user-id != null ruolo == 'admin' se l'utente è un amministratore. ruolo == 'user' se l'utente è un utente normale. prenotazione-id != null stato == 'inAttesa' se la prenotazione è in attesa di approvazione. stato == 'accettata' se la prenotazione è stata approvata. stato == 'rifiutata' se la prenotazione è stata rifiutata. stato == 'richiestaModifica' se la prenotazione è stata modificata dall'utente e la modifica è in attesa di approvazione. slot.prenotato == false

	slots != null & slots.isNotEmpty == true data == '${data.year}-${data.month}-${data.day}' slot.disponibile == false documentoCalendario.exists == false => documentoCalendario.set({'slots': []})
createPrenotazione	context PrenotazioneDao::createPrenotazione(Prenotazione prenotazione, BuildContext context) pre: prenotazione != null & prenotazione.campoId != null & prenotazione.slotId != null & prenotazione.data != null & prenotazione.stato != null & context != null --- context PrenotazioneDao::createPrenotazione(Prenotazione prenotazione, BuildContext context) post: Firestore.collection('prenotazioni').add({ 'campoId': prenotazione.campoId, 'slotId': prenotazione.slotId, 'data': prenotazione.data, 'stato': prenotazione.stato, 'userId': currentUser.uid }) & CustomSnackbar.show(context, "Prenotazione effettuata con successo.")
updatePrenotazione	context PrenotazioneDao::updatePrenotazione(String prenotazioneId, Prenotazione prenotazione, BuildContext context) pre: prenotazioneId != null & prenotazione != null & prenotazione.campoId != null & prenotazione.slotId != null & prenotazione.data != null & prenotazione.stato != null

	& context != null --- context PrenotazioneDao::updatePrenotazione(String prenotazioneId, Prenotazione prenotazione, BuildContext context) post: Firestore.collection('prenotazioni').doc(prenotazioneId).update({ 'campoId': prenotazione.campoId, 'slotId': prenotazione.slotId, 'data': prenotazione.data, 'stato': prenotazione.stato }) & CustomSnackbar.show(context, "Prenotazione aggiornata con successo.")
deletePrenotazione	context PrenotazioneDao::deletePrenotazione(String prenotazioneId, BuildContext context) pre: prenotazioneId != null & context != null --- context PrenotazioneDao::deletePrenotazione(String prenotazioneId, BuildContext context) post: Firestore.collection('prenotazioni').doc(prenotazioneId).delete() & CustomSnackbar.show(context, "Prenotazione cancellata con successo.")
getPrenotazioniByUser	context PrenotazioneDao::getPrenotazioniByUser(String userId, BuildContext context) pre: userId != null & context != null --- context PrenotazioneDao::getPrenotazioniByUser(String userId, BuildContext context) post: List<Prenotazione> prenotazioni = [] Firestore.collection('prenotazioni').where('userId'

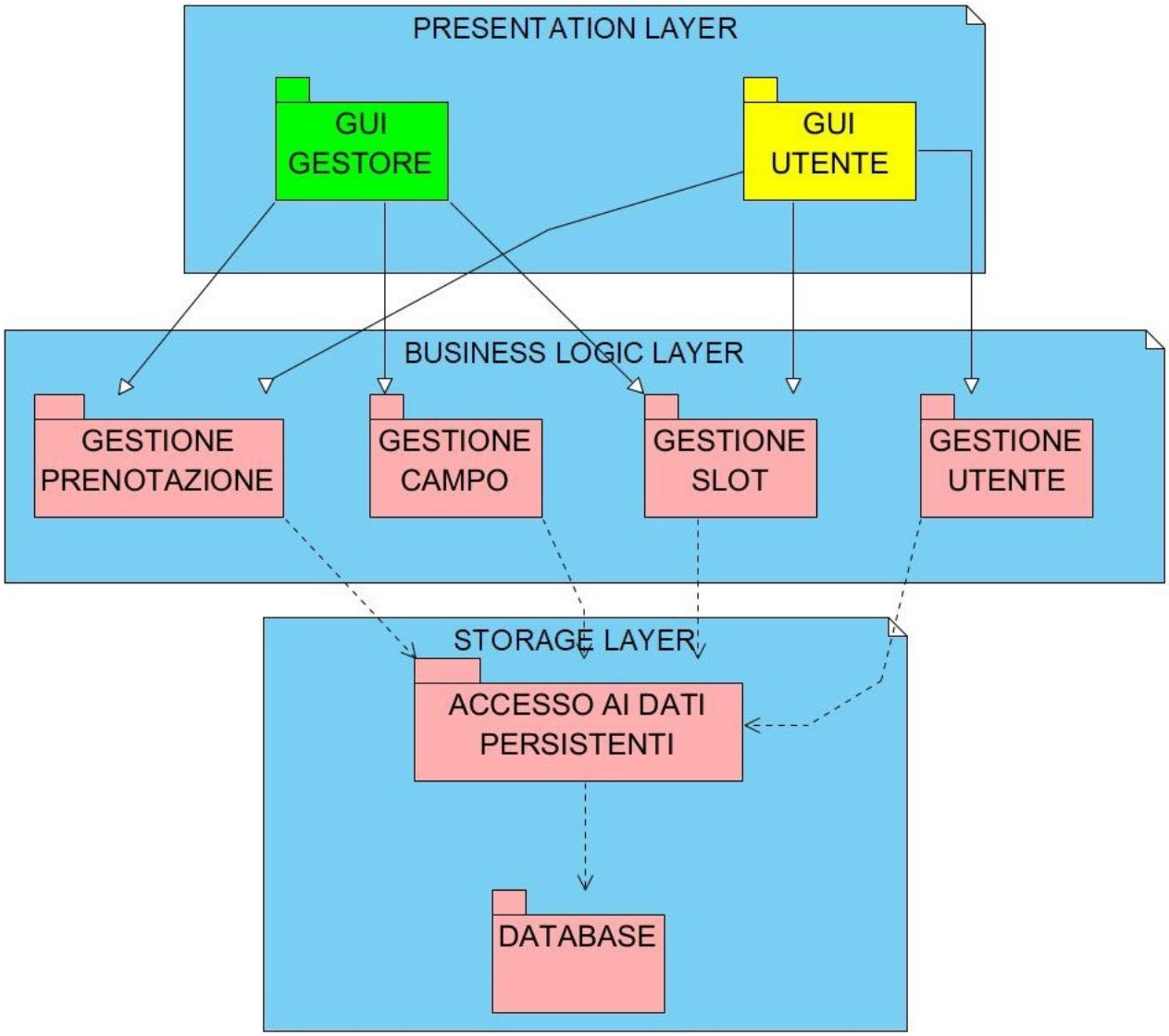
	, isEqualTo: userId).get().then((snapshot) { prenotazioni = snapshot.docs.map((doc) => Prenotazione.fromDocument(doc)).toList() }) & CustomSnackbar.show(context, "${prenotazioni.length} prenotazioni trovate.")

getPrenotazioneById()	context
	PrenotazioneDao::getPrenotazioneById(String prenotazioneId, BuildContext context)
	pre:
	prenotazioneId != null
	& context != null
	--- 	
	context
	PrenotazioneDao::getPrenotazioneById(String prenotazioneId, BuildContext context)
	post:
	Prenotazione prenotazione = null
	
	Firestore.collection('prenotazioni').doc(prenotazioneId).get().then((doc) {
	if (doc.exists) {
	prenotazione = Prenotazione.fromDocument(doc)
	}
	})
	& CustomSnackbar.show(context, prenotazione != null ? "Prenotazione trovata." : "Prenotazione non trovata.")

Avendo usato i Provider che fanno da wrapper ai metodi DAO, e' ininfluente aggiungerli nel documento.

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, la gestione del profilo utente (nome, email, password, ecc.) e la gestione dei permessi (ad esempio, determinando se un utente è un admin 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.