Object Design Document

Studenti

Colella Maria Immacolata	05121**10431**
Mattiello Michele	05121**10185**
Sorrentino Carlo	05121**10884**

Object Design Document

Object Design Document

1. Introduzione

L'Object Design è la fase, nel ciclo di vita del software, in cui vengono definite le scelte finali da prendere prima della fase di implementazione del sistema.
In questo documento verranno descritte tutte le funzionalità individuate nelle fasi precedenti.
In particolare, i Trade-off con le scelte della progettazione degli oggetti, le Linee guida della documentazione delle interfacce, i Packages ed infine le Interfacce delle Classi.

1.1 Object Design Trade-offs

- **Comprensibilità VS Tempo**
  Il codice deve essere quanto più comprensibile possibile per facilitare la fase di testing ed eventuali future modifiche. Questa caratteristica ridurrà il tempo per il testing ed eventuale manutenzione.
- **Interfaccia VS Usabilità**
  L'interfaccia grafica deve essere molto semplice, chiara e concisa, fa uso di pulsanti disposti in maniera da rendere semplice l'utilizzo del sistema a quanti più utenti possibili.
- **Sicurezza VS Efficienza**
  La sicurezza, come descritto nei requisiti non funzionali del Requirement Analysis Document, rappresenta uno degli aspetti chiave della piattaforma. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati sulla crittografia della password degli utenti.
- **Response time VS Hardware**
  Il sistema garantisce una certa reattività alle richieste, e quindi è in grado di poter comunque offrire una contemporaneità dei servizi agli utenti. Ovviamente questa caratteristica sarà limitata dall'hardware del sistema
- **Prestazioni VS Costi**
  Il sistema prevede l'utilizzo di fogli di stile semplici e open per mantenere prestazioni elevate, e cioè non s'intende utilizzare librerie grafiche esterne a pagamento, essendo il progetto sprovvisto di budget.

1.2 Linee guida per la documentazione delle interfacce

Per la scrittura del codice, gli sviluppatori dovranno seguire le seguenti convenzioni e linee guida:

NAME CONVENTIONS

Il nome delle variabili, dei metodi e delle classi deve fornire informazioni utili relative al loro scopo mantenendo una lunghezza medio-corta; inoltre devono seguire la cosiddetta "CamelCase" (ogni lettera iniziale di una parola deve iniziare con una lettera maiuscola)
I nomi delle variabili devono cominciare con una lettera minuscola, se il nome della variabile è costituito da più parole solo l'iniziale delle altre parole sarà maiuscola.
Il nome del metodo tipicamente consiste di un verbo che identifica un'azione, seguito dal nome di un oggetto. I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e createNomeVariabile().
I nomi delle classi devono iniziare con una lettera maiuscola, le parole contenute al suo interno devono cominciare con lettera maiuscola.

1.3 Definizioni, acronimi e abbreviazioni

“*RAD*” sottintende il Requirement Analysis Document,
“*SDD*” sottintende il System Design Document,
“*ODD*” sottintende l’Object Design Document.

1.4 Riferimenti

Questo documento fa riferimento al RAD e al SDD di StudyHall forniti precedentemente.

2. Packages

L'implementazione del back-end si declina nei seguenti pacchetti:

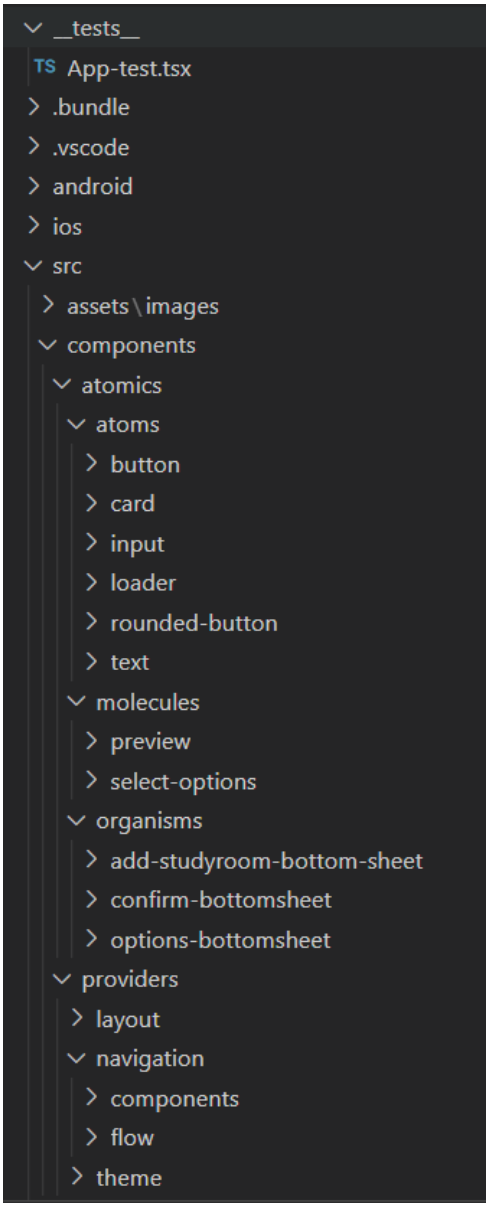
- config
- middleware
- models
- routes
- sdk
- .env

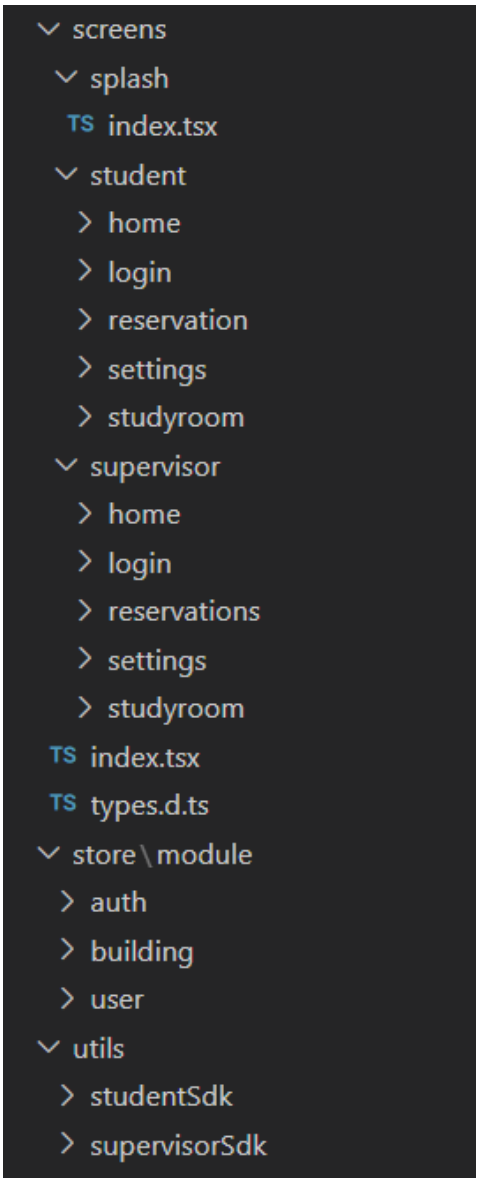
L'implementazione del front-end si declina nei seguenti pacchetti:

- src
    - assets
    - components
        - atomics
            - atoms
            - molecules
            - organisms
        - providers
            - navigation
                - components
                - flow
    - screens
        - splash
        - student
        - supervisor
    - store
    - utils

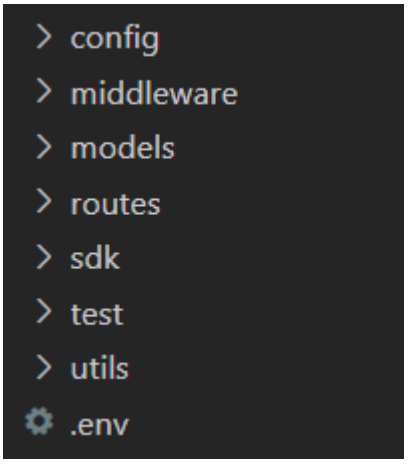
Organizzazione dei packages

Front-end





Back-end



2.1 Back-end Packages

config	
Classe	**Descrizione aggiusta**
data	Lista di fasce orarie consentite per la prenotazione delle aule studio, in quanto queste seguono un formato ben preciso.
firebase	File di configurazione che inizializza la comunicazione col servizio Firebase

middleware	
Classe	**Descrizione**
auth	Classe utilizzata per riconoscere la tipologia di utente che effettua una richiesta

models	
Classe	**Descrizione**
building	Modello che delinea le caratteristiche di un edificio
favorite	Modello che delinea le aule studio preferite dall'utente
reservation	Modello che delinea una prenotazione effettuata da un utente
studyroom	Modello che delinea le caratteristiche di un'aula studio
user	Modello che delinea un utente

routes	
Classe	**Descrizione**
building	Route che gestisce gli edifici
favorite	Route che gestisce le aule preferite da uno studente
organization	Route che gestisce le associazioni
reservation	Route che gestisce le prenotazioni
student	Route che gestisce gli studenti
studyroom	Route che gestisce le aule studio
user	Route che gestisce gli utenti in modo generico

sdk	
Classe	**Descrizione**
Esse3	Classe che gestisce la comunicazione col servizio Esse3
Firebase	Classe che gestisce la comunicazione col servizio Firebase

utils	
Classe	**Descrizione**
data	Directory contenente dati da caricare a priori sul database
dependencies	Directory contenente le dipendenze per effettuare il caricamento dei dati

utils	
Classe	**Descrizione**
.env	Variabili d'ambiente utilizzate. Contiene le API di firebase, l'URL di Esse3, l'URL di MongoDB, il nome della directory in cui salviamo i media, l'URL di base del Server, l'IP di base e la porta.

2.2 Front-end Packages

atoms	
Classe	**Descrizione**
button	Descrive lo stile di un button
card	Descrive lo stile di una card
input	Descrive lo stile di un input standard o image
loader	Descrive lo stile del caricamento
rounded-button	Descrive lo stile di un button circolare
text	Descrive lo stile del testo

molecules	
Classe	**Descrizione**
preview	Descrive lo stile della presentazione dell'aula studio
select-options	Componente che gestisce un input a scelta multipla

organisms	
Classe	**Descrizione**
add-studyroom-bottom-sheet	Gestisce l'UI dell'aggiunta di una nuova aula studio
confirm-bottomsheet	Componente grafica che gestisce la conferma di un'azione
options-bottomsheet	Componente grafica che mostra gli input a scelta multipla

providers	
Classe	**Descrizione**
layout	Classe che gestisce il layout generale delle pagine
theme	Classe che gestisce il tema generale dell'app

component	
Classe	**Descrizione**
back-header	Classe che gestisce l'header per tornare alla schermata precedente
header	Classe che gestisce l'header delle schermate principali
tab-bottom-bar	Classe che gestisce la barra di navigazione nella homepage

flow	
Classe	**Descrizione**
student	Classe che gestisce il flusso delle schermate dello studente
supervisor	Classe che gestisce il flusso delle schermate del gestore
unlogged	Classe che gestisce il flusso di schermate di un utente non loggato
index	Classe che gestisce il flusso generale dell'app

screens	
Classe	**Descrizione**
splash	Schermata iniziale dell'app

student	
Classe	**Descrizione**
home	Schermata che gestisce la home dello studente
login	Schermata che gestisce il login dello studente
reservation	Schermata che gestisce le prenotazioni dello studente
settings	Schermata che gestisce le impostazioni dello studente
studyroom	Schermata che gestisce le informazioni dell'aula studio da mostrare allo studente

supervisor	
Classe	**Descrizione**
home	Schermata che gestisce la home del gestore
login	Schermata che gestisce il login del gestore
reservation	Schermata che gestisce le prenotazioni di un'aula studio di cui è responsabile il gestore
settings	Schermata che gestisce le impostazioni del gestore
studyroom	Schermata che gestisce le informazione dell'aula studio da mostrare al gestore

store/module	
Classe	**Descrizione**
auth	Classe che gestisce lo storage delle informazioni riguardanti l'autenticazione
content	Classe che gestisce lo storage dei dati

utils	
Classe	**Descrizione**
studentSdk	Classe che gestisce l'interazione con il backend riguardante lo studente
supervisorSdk	Classe che gestisce l'interazione con il backend riguardante il gestore

3. Class Interface

3.1 Back-end

3.1.1 Config

Nome classe	Auth
Descrizione	Classe utilizzata per riconoscere la tipologia di utente che effettua una richiesta
Signature dei metodi	# student (Request, Response, any) : void action: ● checkEsse3Loggon #organization (Request, Response, any) : void action: ● verifyFirebaseToken
Pre-condizioni	token
Post-condizioni	
Invariante	

3.1.2 Models

Nome classe	User
Descrizione	Modello che delinea un utente
Signature dei metodi	• checkAndSaveOrganization (username, email, token) : void • checkAndSaveStudent(username, token) : void
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	Studyroom
Descrizione	Modello che delinea le caratteristiche di un'aula studio
Signature dei metodi	• checkAndSave (name, seats, floor, image, isactive, building, owner) : void
Pre-condizioni	
Post-condizioni	
Invariante	

3.1.3 Routes

Nome classe	Building
Descrizione	Route che gestisce gli edifici
Signature dei metodi	# get (“/”, (Request, Response)) : void action ● allBuildings
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	Favorite
Descrizione	Route che gestisce le aule preferite da uno studente
Signature dei metodi	# get (“/”, Auth.student, (Request, Response) : void) : void action ● findByUser # get (“/:id/create”, Auth.student, (Request, Response) : void) : void action ● create # delete (“/:id”, Auth.student, (Request, Response) : void) : void action ● delete
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	Organization
Descrizione	Route che gestisce le associazioni
Signature dei metodi	# post(“/”, Auth.organization, (Request, Response) : void) : void action ● checkAndSaveOrganization
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	Reservation
Descrizione	Route che gestisce le prenotazioni
Signature dei metodi	# post(“/create”, Auth.student, (Request, Response) : void) : void # get(“/”, Auth.student, (Request, Response) : void) : void # get(“/active”, Auth.student, (Request, Response) : void) : void # get(“/expired”, Auth.student, (Request, Response) : void) : void # delete(“/:id”, Auth.student, (Request, Response) : void) : void # post(“/”, Auth.student, (Request, Response) : void) : void ● action ○ countReservationsByDateAndStudyroom # get(“/:id”, Auth.organization, (Request, Response) : void) : void ● action ○ getReservationsByStudyroom
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	Student
Descrizione	Route che gestisce gli studenti
Signature dei metodi	# get(“/login”, (Request, Response) : void) : void ● action ○ checkAndSaveStudent
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	Studyroom
Descrizione	Route che gestisce le aule studio
Signature dei metodi	# get(“/”, (Request, Response) : void) : void ● action ○ getStudyroomGroupByBuilding # get(“/supervisor”, Auth.organization, (Request, Response) : void) : void # post(“/create”, Auth.organization, (Request, Response) : void) : void # get(“/:id/changestatus”, Auth.organization, (Request, Response) : void) : void # get(“/:id”, (Request, Response) : void) : void # delete(“/:id”, Auth.organization, (Request, Response) : void) : void # patch(“/:id”, Auth.organization, (Request, Response) : void) : void
Pre-condizioni	
Post-condizioni	
Invariante	

3.1.4 Sdk

Nome classe	Esse3
Descrizione	Classe che gestisce la comunicazione col servizio Esse3
Signature dei metodi	# login(username: string, password: string): {username: string, token: string} # checkLogon(token: string): {check: boolean}
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	Firebase
Descrizione	Classe che gestisce la comunicazione col servizio Firebase
Signature dei metodi	# login(username: string, password: string): {username: string, email: string, token: string} # verifyToken(token: string): {email: string}
Pre-condizioni	
Post-condizioni	
Invariante	

3.2 Front-end

3.2.1 Store

Nome classe	AuthStore
Descrizione	Classe che gestisce lo storage delle informazioni riguardanti l'autenticazione
Signature dei metodi	# loginStudent(state,{username: string, token: string}): void # loginSupervisor(state, {email: string, username: string, token: string}): void # logout(state): void
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	ContentStore
Descrizione	Classe che gestisce lo storage dei dati

Signature dei metodi	# setLoading(state, {boolean}): void
	# setBuildings(state,
	{
	_id: string,
	name: string,
	coords:{string, string}
	}[]
	): void
	# setStudyroom(state,
	{
	_id: string
	name: string
	building: string
	image: string
	seats: string
	floor: string
	isactive: string
	}
	): void
	# setStudyrooms(state,
	Studyroom[]
	): void

	# setStudyroomsGroupByBuildings(state,
	{
	_id: string
	studyrooms: Studyroom[]
	}
	): void
	# setReservations(state,
	{
	_id: string
	building: string
	studyroom: Studyroom
	}
	): void
	# setActiveReservations(state,
	{
	_id: string
	building: string
	studyroom: Studyroom
	date: string
	start: string
	end: string
	}
	): void
	# setExpiredReservations(state,
	Reservation[]
	): void
	# setFavorites(state,
	{
	_id: string
	studyroom: string
	}
	): void
	# resetState(state): void

Pre-condizioni	
Post-condizioni	
Invariante	

3.2.2 Utils

Nome classe	StudentSDK
Descrizione	Classe che gestisce l'interazione con il backend riguardante lo studente
Signature dei metodi	# login(username: string, password: string): void # getBuilding(): void # getUser(): void # getStudyrooms(): void # getStudyroom(): void # createReservation(id: string, date: Date, start: string, end: string): void # getReservations(): void # getActiveReservations(): void # deleteReservation(id: string): void # getAssignedSeats(id: string, date: Date): void # createFavorite(id: string): void # deleteFavorite(id: string): void # getFavorites(): void
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	SupervisorSDK
Descrizione	Classe che gestisce l'interazione con il backend riguardante lo studente
Signature dei metodi	# login(email: string, password: string): void # getBuilding() : void # createStudyroom(name: string, building: string, floor: string, seats: string, image: string) : void # updateStudyroom(id: string, name: string, building: string, floor: string, seats: string, image: string) : void # getStudyrooms() : void # getStudyroom(id: string) : void # deleteStudyroom(id: string) : void # changeStatusStudyroom(id: string) : void # getUser() : void # getReservations(id: string) : void
Pre-condizioni	
Post-condizioni	
Invariante	