UNIVERSITÀ DEGLI STUDI DI PALERMO

OBJECT DESIGN DOCUMENT

Progetto “Noleggio auto/moto/bici/monopattino on-demand”

Gli studenti:
*Salvatore Gabriele **K**arra*
*Salvatore **A**cquaviva*
*Maria **R**ausa*
*Giuseppe **M**aggio*

Professori:
Valeria Seidita
Roberto Pirrone

Tesina 2020/21

INDICE

1. Compromesse e scelte nella progettazione degli oggetti

1.1. Suddivisione in livelli

Al fine di garantire la modularità del sistema e per facilitarne lo sviluppo, il software è stato diviso in livelli differenti.
Lo sviluppo è semplificato in quanto le componenti del sistema, in questo modo, vengono sviluppate autonomamente rispetto alle altre. Inoltre, ciò facilita l'identificazione di problemi del sistema in fase di implementazione.
I livelli da noi identificati sono:

- Interfaccia grafica, gestita mediante React e Javascript.
- Gestione della sessione, mediante l'utilizzo della libreria passport e token.
- Gestione delle funzionalità di base del sistema mediante API, React ed Express.
- DBMS gestito tramite mongoose, libreria per la gestione di DataBase di MongoDB

1.2. Gestione e progettazione dell'interfaccia grafica

Per la gestione delle interfacce grafiche si è scelto di utilizzare un'architettura web-based mediante **JavaScript** e il framework di sviluppo **ReactJS**, una libreria Javascript sviluppata da Facebook per la creazione di User Interface interattive, in quanto facilita la realizzazione di interfacce dinamiche in funzione dei dati utilizzati.

Inoltre si è scelto di affidarsi ai framework React-Bootstrap e a **Material UI** per quanto riguardo la realizzazione dei componenti grafici, in quanto consentono la realizzazione di interfacce responsive in modo rapido (pur non trascurando la loro resa grafica), al fine di rendere disponibile il sistema su piattaforme differenti (rendendolo multipiattaforma).

Per la realizzazione di richieste HTTPS è stato scelto **axios**, un client HTTPS promise-based che facilita l'implementazione di richieste asincrone.

Per impostare il file di routing al fine di reindirizzare i componenti è stato utilizzato il modulo **react-router-dom**.

I motivi e i vantaggi di queste scelte sono i seguenti:

1. Incremento della modularità del sistema mediante **React**, il quale permette la costruzione di interfacce complesse mediante l'aggregazione di "componenti" differenti.

2. Riuso facilitato, grazie alla suddivisione in componenti.
3. Gestione degli eventi semplificata, mediante le funzionalità di **Javascript**.
4. Progettazione agevole del layout, mediante **JSX** (la cui sintassi è simile, per certi versi, a quella del linguaggio di markup HTML. JSX, inoltra, sfrutta anche le potenzialità di Javascript), **CSS**, **JavaScript** e i componenti offerti da React-Bootstrap e **Material UI**.
5. Gestione delle finestre pop-up semplice mediante gli alert e la classe **Modal** di React-Bootstrap.
6. Indipendenza dal sistema con cui l'utente accede al software (il sistema è, di fatto, multipiattaforma).

Le boundary del sistema presenti nel RAD sono state implementate mediante pagine JSX, le quali comunicano e interagiscono in modo dinamico con il sistema mediante i rispettivi controller e mediante le API di Javascript.

E' stato fatto uso inoltre di API di Google Maps per la visualizzazione dei parcheggi su una mappa in modo dinamico

1.3. Gestione e progettazione della componente server

Per la gestione della componente server si è scelto di utilizzare **Node.js**, un ambiente a runtime Javascript comandato da eventi asincroni, che risulta particolarmente appropriato per lo sviluppo di applicazioni di rete scalabili, in quanto è orientato agli eventi: pertanto, le operazioni di I/O possono essere svolte in modo asincrono. Grazie a questo, risulta un ambiente altamente scalabile e che garantisce un alto throughput.

Nello specifico, come framework per lo sviluppo web è stato scelto **Express.js**. Esso facilita la gestione delle richieste HTTPS, in quanto permette di configurare i middleware per interagire con le richieste HTTPS (effettuate tramite axios) del front-end mediante diversi URL (routes). Express permette di rispondere in modo dinamico alle pagine JSX e ai component JS.

Le Boundary del sistema presenti nel RAD sono state tradotte e programmate in componenti React, le Control precedentemente individuate sono state mappate nelle rotte di Express e per le Entity si sono utilizzati gli Hooks di React e il localStorage della Web Storage API di Javascript.

Per interfacciarsi al database è stato fatto ampio uso di Axios, che ci hanno permesso di creare richieste XMLHttpRequest a partire dal frontend e che ha permesso inoltre l'uso di Promise e il catching delle risposte del server.

Di seguito sono riportati vari moduli utilizzati per lo sviluppo del front-end:

- @material-ui/core@4.11.4
- @material-ui/icons@4.11.2
- @material-ui/lab@4.0.0-alpha.58
- @react-google-maps/api@2.2.0
- @testing-library/jest-dom@5.14.1
- @testing-library/react@11.2.7
- @testing-library/user-event@12.8.3
- axios@0.21.1
- bootstrap@4.6.0
- classnames@2.3.1
- is-empty@1.2.0
- jwt-decode@3.1.2
- react-bootstrap@1.6.1
- react-calendar@3.4.0
- react-datepicker@4.1.1
- react-dom@17.0.2
- react-redux@7.2.4
- react-router-dom@5.2.0
- react-scripts@4.0.3
- react@17.0.2
- redux-thunk@2.3.0
- redux@4.1.0
- validator@13.6.0
- web-vitals@1.1.2

E per lo sviluppo del back-end:

- bcryptjs@2.4.3
- body-parser@1.19.0
- concurrently@6.2.0
- express@4.17.1
- is-empty@1.2.0
- jsonwebtoken@8.5.1
- mongoose@5.12.14
- nodemon@2.0.12
- passport-jwt@4.0.0
- passport@0.4.1
- validator@13.6.0
- nodemailer
- codice-fiscale-js

1.4. Gestione e progettazione dell'interfaccia mobile

L'interfaccia mobile è stata realizzata mediante l'uso di React-Native grazie all'uso di Cordova, framework per lo sviluppo di applicativi per dispositivi mobili. È stata inoltre realizzata un'interfaccia tramite WebView.

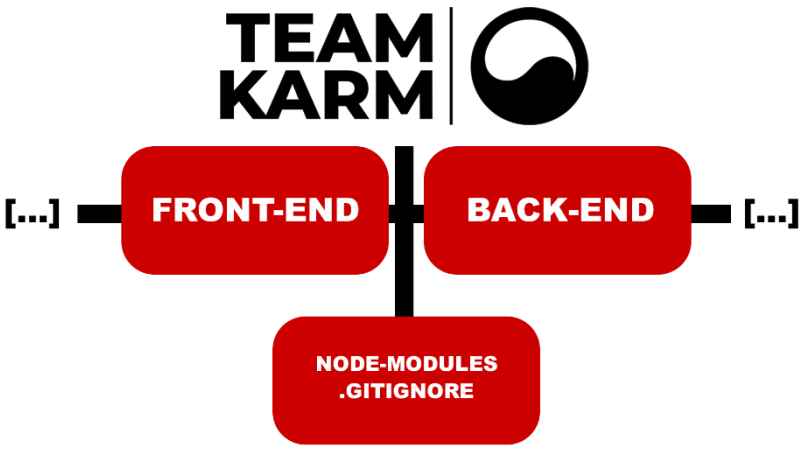
1.5. Connessione al database

Per interfacciarsi con il database, si è scelto di utilizzare la libreria mongoose di MongoDB, uno strumento di modellazione di oggetti MongoDB progettato per funzionare in un ambiente asincrono. Mongoose supporta sia le promise che i callback.e ci ha permesso di definire i Model, ossia le tabelle e gli attributi del nostro database

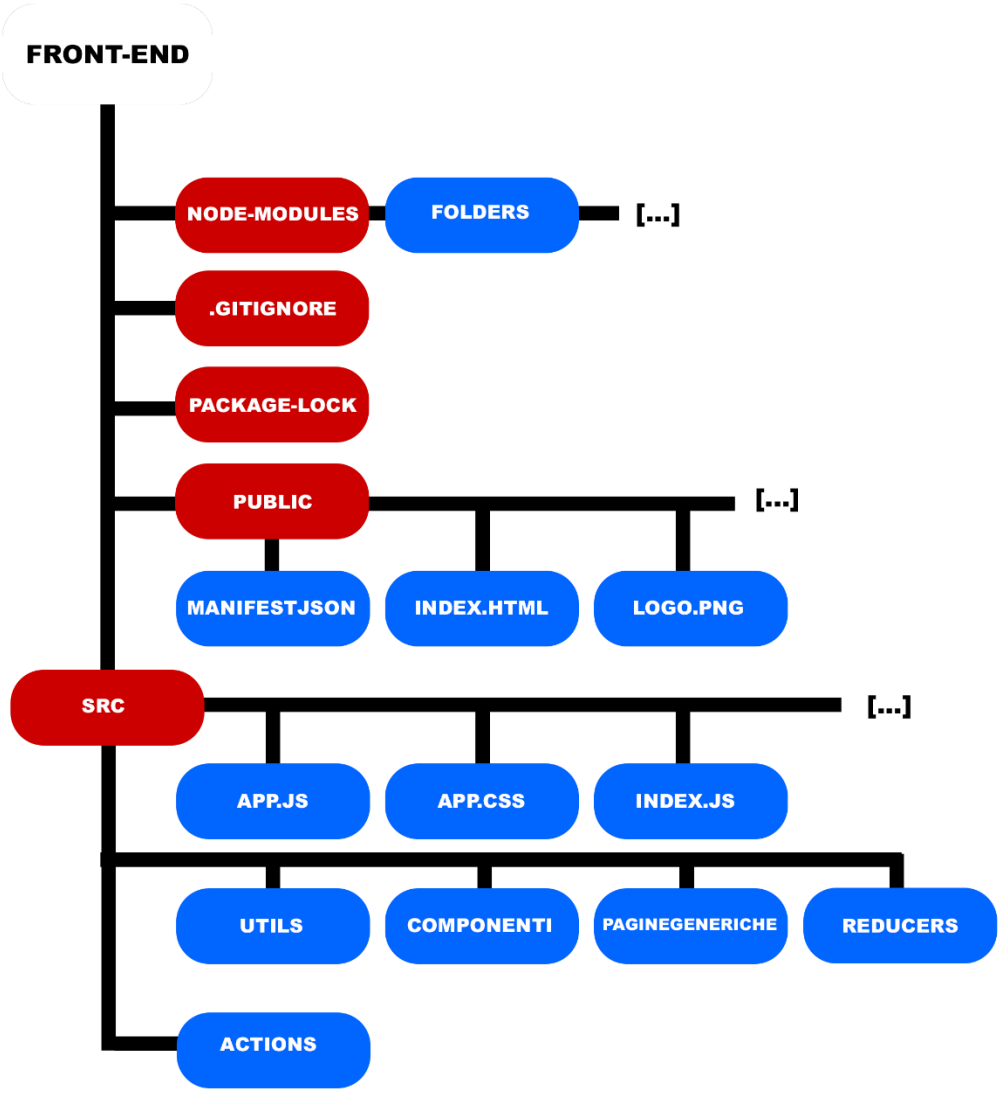
2. Package

Durante lo sviluppo del software si è scelto di dividere i componenti in packages, per avere un'organizzazione migliore e fedele alla documentazione presentata. Uniformemente all'architettura scelta, si è deciso di suddividere l'applicativo in due package principale, cioè frontend e backend.
Di seguito è riportata la struttura dei package.

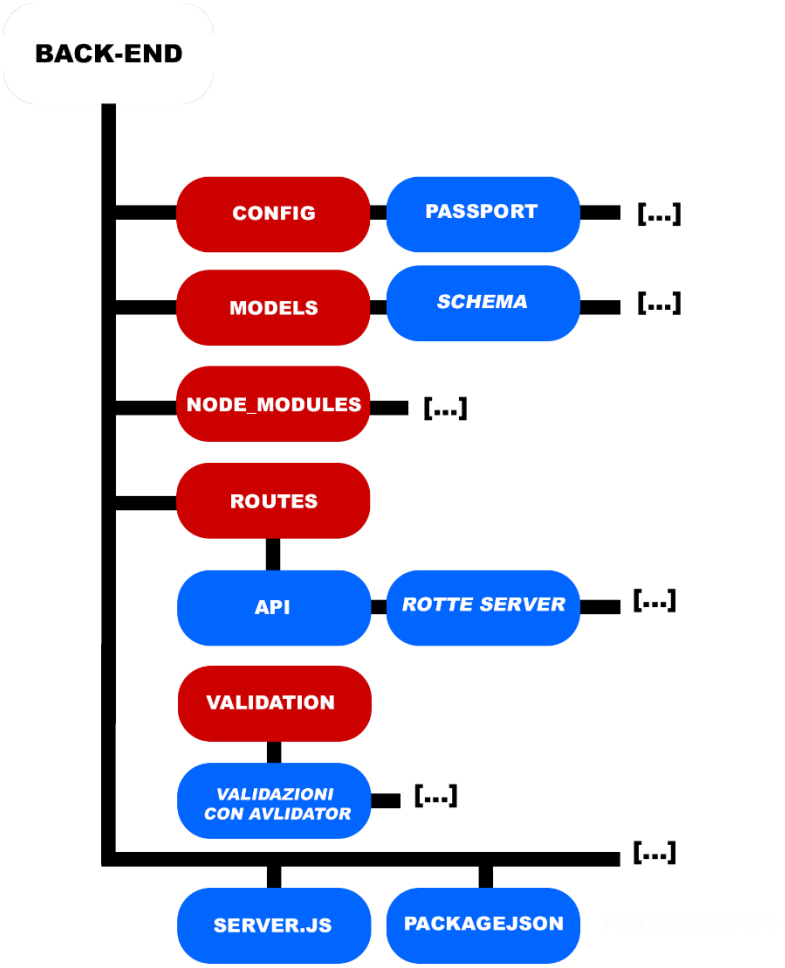
2.1. Directory



2.2. Front-End


FRONT-END
NODE-MODULES
FOLDERS
[...]
.GITIGNORE
PACKAGE-LOCK
PUBLIC
[...]
MANIFESTJSON
INDEX.HTML
LOGO.PNG
SRC
[...]
APP.JS
APP.CSS
INDEX.JS
UTILS
COMPONENTI
PAGINEGENERICHE
REDUCERS
ACTIONS

2.3. Back-End



Il sistema è accessibile al seguente link: karm.zapto.org