Relazione del progetto del corso di Tecnologie e Applicazioni Web [CT0142]

Restaurant App

Baccega Sandro, 865024

Progetto realizzato da:

Baccega Sandro, Scodeller Giovanni

Indice dei Contenuti

Come avviare l'applicazione

Dipendenze:

- Yarn
- Node.js 10.x
- Android Sdk (per compilare Android)

Per avviare l'applicazione basterà eseguire i seguenti comandi:

Per installare tutti i `node_modules` necessari:

```
yarn install
```

Per avviare il back-end:

```
yarn start-backend
```

Per avviare uno dei front-end utlizzare:

```
yarn start-web
yarn start-android
yarn start-desktop
```

Il front-end web del progetto è accessibile presso questo indirizzo: http://localhost:4200

Sono già esistenti alcuni account di test all'interno del database:

- waiter@gmail.com

  password: waiter

  ruolo: Cameriere
- chef@gmail.com

  password: chefff

  ruolo: Cuoco
- bartender@gmail.com

  password: bartender

  ruolo: Barman
- cashier@gmail.com

  password: cashier

  ruolo: Cassiere

Architettura di Sistema

Il progetto é stato sviluppato seguendo lo stack di sviluppo **MEAN** con qualche aggiunta:

- Un database realizzato con **MongoDB** per la persistenza dei dati, hostato sul servizio **Atlas**, che é un cloud service gratuito (fino ad una certa soglia) offerto direttamente da *MongoDB, Inc.*, l'azienda che mantiene MongoDB.
- Un front-end realizzato con una *web application*, construita con **Angular**, ed accessibile tramite 3 piattaforme diverse:
    - Un qualunque browser
    - Un applicazione nativa Desktop, realizzato con **Electron**
    - Un applicazione nativa mobile, realizzato con **Apache Cordova**
- Un webserver per il back-end realizzato con **NodeJS** per l'interazione tra il front-end e il database. É realizzato utilizzando il middleware **Express.js**, per una semplice struttura, **MongooseJS** per una semplice interazione con il database e **Socket.io** per la gestione dei websockets.

Come packet-manager è stato scelto **Yarn**, per la sua semplicità d'uso.

Front-end

La parte front-end, scritto con **Typescript** utlizzando il framework **Angular**, è stata realizzata mettendo insieme dei componenti provenienti da **Angular Material**, una libreria che fornisce componenti Angular utilizzabili conformi alle *guidelines* di **Material Design**.

Per utlizzare Angular Material, e mantenere organizzato il progetto, abbiamo importato un modulo apposito **MaterialComponentsModule**, cioè un modulo creato da noi che racchiude tutte le importazioni da Angular Material.

Questo è un esempio di importazione da Angular Material, presente sul MaterialComponentsModule:

```
import { NgModule } from "@angular/core";
// ...
import { MatCardModule } from "@angular/material/card";

@NgModule({
  declarations: [],
  exports: [
    // ...
    MatCardModule,
  ]
})
export class MaterialComponentsModule {}
```

Per utlizzare un componente di Angular Material si procede così:

```
<mat-card class="table-block">
    ...
</mat-card>
```

SCSS

Per i fogli di stile abbiamo utlizzato **SCSS** (Sassy CSS), un superset di CSS che permette una scrittura di CSS molto più semplice e organizzata.

Questo è un esempio di codice SCSS utilizzato in *app.component.scss*:

```
@import "./css-variables.scss";
.main-container {
  height: 100%;
  width: 100%;
  background-color: $background;
  .content-container {
    display: grid;
    grid-template-rows: 8vh auto;
  }
  .sidenav {
    width: 60vw;
  }
}
```

Come si può vedere, la struttura delle classi gerarchica di scss, è molto più organizzata e meno *error prone* di un normale script CSS. Ovviamente ogni codice CSS valido è anche codice SCSS valido, in quanto SCSS, come già detto, è un superset di CSS.

Backend

I compiti principali del server sono:

- Fornire i dati richiesti dal client provenienti dal database.
- Aggiornare i dati provenienti dal front-end sul database e su tutti gli altri client collegati.
- Fornire un API per l'autenticazione tramite JWT di tutti gli utenti.
- Verificare che ogni richiesta di dati sia legittima, controllando il ruolo dell'utente nel JWT usato per la richiesta.

Il server Node interagisce con il front-end in due modalità:

- Tramite richieste **HTTP**

  Infatti il server utilizza un architettura **REST**, in modo da gestire facilmente ogni tipo di richiesta.

- Tramite websockets

  Il server utilizza la libreria **Socket.io** per una facile e veloce comunicazione in tempo reale con il client, in modo da poterlo notificare di aggiornamenti dello stato di alcune risorse, come per esempio l'aggiornamento dello stato di un ordine.

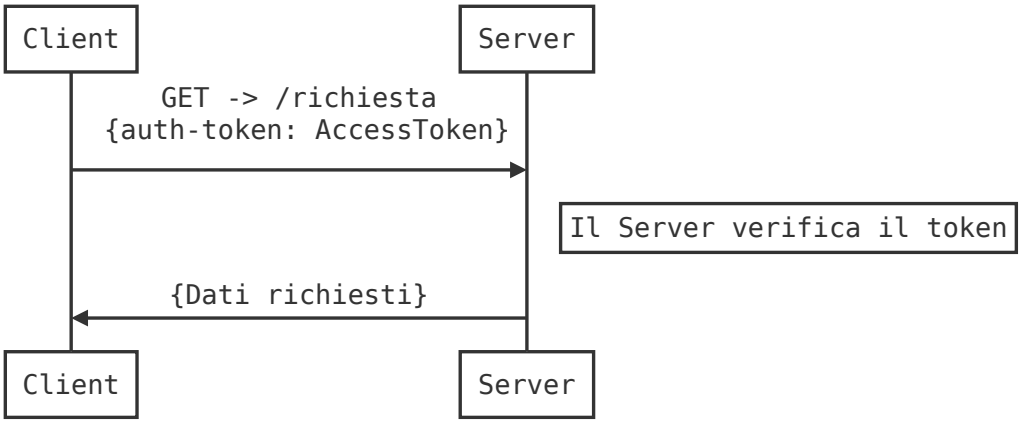
Il server su ogni API protetta utilizza questa funziona di verifica del token:

```
module.exports = function(req, res, next) {
  const token = req.header("auth-token");
  if (!token) return res.status(401).send("Access Denied!");
  try {
    const verified = jwt.verify(token, process.env.TOKEN_SECRET);
    req.user = verified;
    next();
  } catch (err) {
    res.status(400).send("Invalid Token!");
  }
};
```

Autenticazione con JWT (JSON Web Token)

Per verificare che le richieste HTTP siano provenienti da utenti legittimi, ad ogni richiesta HTTP (a parte per l'autenticazione) è necessario fornire un JWT tramite il campo `auth-token` dentro l'header della richiesta.

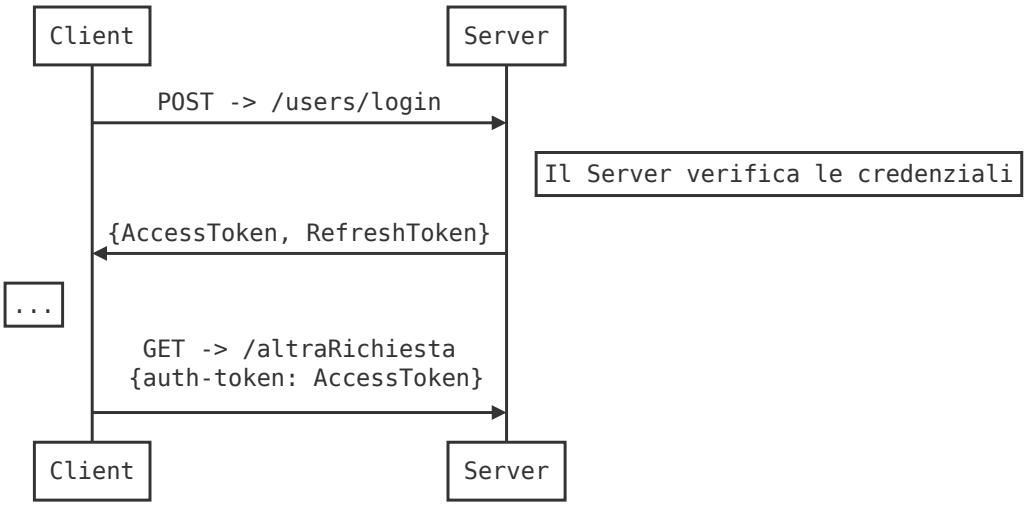
Questo è un esempio di richiesta HTTP:



Durante la verifica del token, il server controlla anche se il ruolo dell'utente coincide con i dati che ha richiesto (Un cuoco non può creare un utente nuovo).

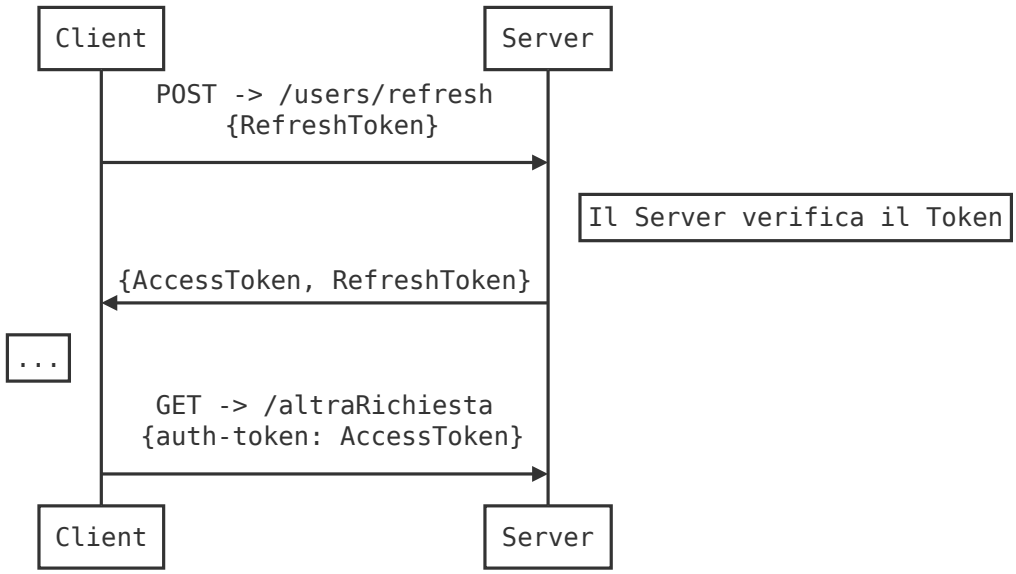
Tuttavia dato che i JWT hanno una data di scadenza, abbiamo utilizzato la tecnica dei **Refresh Token** per un veloce rimpiazzo, ovvero un token extra fornito durante l'autenticazione che permette il rinnovo automatico dell'access token.

Questo è un esempio di autenticazione:



Se la sessione dura a lungo, è possibile che scada quando invece è ancora necessario.

A quel punto il client, prima di fare un altra richiesta, richiederà un nuovo token:



Modelli Dati

Per la realizzazione dell'applicazione sono stati scelti i seguenti modelli:

```
Dish {
  dishId: number;
  name: string;
  price: number;
  quantity: number;
  preparation: string;
  status: DishStatus;
  category: string;
}

Course {
  courseId: number;
  category: string;
  dishes: Dish[];
}

Order {
  orderId: number;
  dishes: Dish[];
  foodStatus: OrderStatus;
  drinkStatus: OrderStatus;
  table: number;
  waiter: string;
}

Table {
  number: String;
  seats: number;
  free: boolean;
  waiter: String;
```

```
}

Bill {
  billId: number;
  total: number;
  date: Date;
  dishes: Object[];
  table: string;
  customerNumber: Number;
  waiter: string;
}

User {
  userId: number;
  name: string;
  email: string;
  role: Role;
  dailyPlate: number;
  totalPlate: number;
}

--- STATUSES & ROLES ---

DishStatus {
  Waiting = 0,
  Started = 1,
  Finished = 2
}

OrderStatus {
  Waiting = 0,
  Preparing = 1,
  Ready = 2,
  Delivered = 3
}

Role {
  Waiter = 0,
  Chef = 2,
  Bartender = 3
  Cashier = 4
}
```

Endpoints

/users

Method	Body	Query Params	Description	Users
GET		{role : String}	Ricevi tutti i dipendenti con ruolo *role*, altrimenti tutti	Cassa

/users/login

Method	Body	Query Params	Description	Users
POST	{email:String, password:String}		Utilizzato per il login dell'utente	

/users/register

Method	Body	Query Params	Description	Users
POST	{email:String, password:String, role: String}		Utilizzato per registrare un utente	

/users/:id

Method	Body	Query Params	Description	Users
GET		{id: Number}	Ricevi il dipendente :id	Cassa
DELETE		{id: Number}	Cancella il dipendente :id	Cassa

/users/refresh

Method	Body	Query Params	Description	Users
POST	{AccessToken: String}		Ricevi il dipendente :id	Cassa Cameriere Barman Chef

/bills

Method	Body	Query Params	Description	Users
GET			Ricevi tutti i conti	Cassa
POST	{ table: Number, customer:		Crea un nuovo conto vuoto per il tavolo Tablenumber	Cameriere

Method	Body	Query Params	Description	Users
POST	{customer: Number}		per il tavolo :tableNumber, occupandolo	Cameriere

/bills/:tableId

Method	Body	Query Params	Description	Users
POST	{total: Number, dishes: Dish[]}	{tableId: Number}	Ricevi il conto del tavolo :id	Cassa

/orders

Method	Body	Query Params	Description	Users
GET			Ricevi tutti gli ordini in base al ruolo	Barman Cuoco Cameriere
POST	{table: Number, dishes: Dish[] }		Invia un nuovo ordine nel tavolo specificato	Cameriere

/orders/:id

Method	Body	Query Params	Description	Users
GET		{id: Number}	Ricevi l'ordine :id	Tutti
POST	{ food*: FoodStatus, drink*: DrinkStatus }		Cambia lo stato dell'ordine	Tutti

/orders/tables/:id

Method	Body	Query Params	Description	Users
GET		{table: Number}	Ricevi gli ordini del tavolo :table	Tutti

/orders/:id/:dish

Method	Body	Query Params	Description	Users
GET			Ricevi il piatto :dish dell'ordine :id	Tutti
POST	{status:DishStatus}		Cambia lo stato del piatto :dish dell'ordine :id	Barman Cuoco

			:dish dell'ordine :id	Cuoco
Method	**Body**	**Query Params**	**Description**	**Users**

/courses

Method	Body	Query Params	Description	Users
GET			Ricevi la lista di menù	Tutti

/courses/createCourse

Method	Body	Query Params	Description	Users
POST	{category:String}		Crea una nuova categoria per il menù	

/courses/newPlate

Method	Body	Query Params	Description	Users
POST	{Dish}		Crea un nuovo piatto per il menù	

/tables

Method	Body	Query Params	Description	Users
GET			Ricevi tutti i miei tavoli	Cameriere Cassa
POST	{ Table }		Crea un nuovo tavolo	Cassa

/tables/:id

Method	Body	Query Params	Description	Users
GET			Ricevi il tavolo :id	Tutti

/tables/freeTables

Method	Body	Query Params	Description	Users
GET			Ritorna tutti i tavoli liberi	Cameriere Cassa

/tables/myTables

Method	Body	Query Params	Description	Users
GET			Ricevi i tavoli che sono impegnati dal quel cameriere	Cameriere

/statistics/daily

Method	Body	Query Params	Description	Users
GET			Ricevi le statistiche della giornata	Cassa

/statistics/user/:id

Method	Body	Query Params	Description	Users
GET		{id: String}	Ricevi le statistiche dell'utente :id	Cassa

Componenti di Angular

L'applicazione si divide in 22 componenti.

- **Il cameriere (Waiter)**
  - **waiter-dashboard**

    Pagina principale che crea altri 2 sotto componenti:

    - **waiter-table-list**

      La lista dei tavoli appartenenti al cameriere
    - **waiter-table-detail (Utilizzando un router-outlet)**

      La lista di ordini di un determinato tavolo
  - **waiter-new-table**

    Pagina secondaria, accessibile via sidenav, dove un cameriere può occupare un tavolo
  - **waiter-new-order**

    Pagina secondaria, accessibile via FAB (Floating Action Button) in basso a destra nella dashboard (Solo quando un tavolo è selezionato)
- **Il Cuoco (Chef)**
  - **chef-dashboard**

    Pagina principale che crea altri 2 sotto componenti:

    - **order-list (source = "chef")**

      Componente condiviso tra chef e bartender che mostra gli ordini in attesa secondo la logica FIFO (First In First Out)
    - **chef-order-detail**

      La lista dei piatti di un selezionato ordine, con possibilità di iniziare e completare la preparazione di un certo piatto
- **Il Barista (Bartender)**
  - **bartender-dashboard**

    Pagina principale che crea altri 2 sotto componenti:

    - **order-list (source = "bartender")**

      Componente condiviso tra chef e bartender che mostra gli ordini in attesa secondo la logica FIFO (First In First Out)
    - **bartender-order-detail**

      La lista dei piatti di un selezionato ordine, con possibilità di iniziare e completare la preparazione dell'intero ordine
- **La Cassa (Cashier)**
  - **cashier-dashboard**

    Pagina principale mostra i tavoli occupati e liberi
  - **cashier-view-order**

    Pagina secondaria, accessibile da cashier-dashboard selezionando un tavolo, dove è possibile vedere lo stato degli ordini di un determinato tavolo
  - **cashier-bill**

Pagina secondaria, accessibile da cashier-view-order usando il FAB in basso a destra, dove è possibile visualizzare il conto totale di un tavolo e poterlo chiudere

- **cashier-statistics**

  Pagina secondaria, accessibile dalla sidenav, dove è possibile vedere le statistiche della giornataa, la lista degli utenti, e dove è possibile chiudere la giornata azzerando le statistiche del giorno

- **cashier-statistics-user**

  Pagina secondaria, accessibile da cashier-statistics selezionando un utente, dove è possibile vedere le statistiche di un utente e dove è possibile eliminarlo dall'applicazione

- **cashier-new-user**

  Pagina secondaria, accessibile da cashier-statistics selezionando l'icona con il simbolo "+", dove è possibile aggiungere un utente all'applicazione

- Altri componenti
  - **dummy**

    Componente vuoto, utile per il routing

  - **header**

    Componente che forma la navbar dell'applicazione

  - **sidenav**

    Componete che forma la sidenav dell'applicazione

  - **login**

    Componente che contiene il form di login

  - **logout**

    Componente per il logout, accessibile dalla sidenav

  - **snackbar**

    Componente per creare una snackbar

Servizi di Angular

L'applicazione è formata da 11 servizi Angular:

- **auth.service**

  Contiene le richieste per l'autenticazione/registrazione degli utenti, ed anche la logica per la gestione dei tokens.

- **bill.service**

  Contiene le richieste per la creazione e il fetch di una *bill* (conto).

- **dish.service**

  Contiene le richieste per la gestione del menù.

- **notification.service**

  Contiene la logica per la notfica dei camerieri (*waiter*).

- **order.service**

  Contiene le richieste per la gestione degli ordini.

- **socket.service**

  Contiene il codice per la creazione e l'utilizzo di un socket.

- **statistics.service**

  Contiene le richieste per le statistiche.

- **table.service**

  Contiene le richieste per la gestione dei tavoli.

- **user.service**

  Contiene le richieste per la gestione degli utenti.

- **utils.service**

  Contiene alcuni **Subject** utili, usati da molti components, per la gestione dell'applicazione:

  - titleWatcher: Subject per la gestione del titolo nell'Header.
  - sidebarWatcher: Subject che monitora lo stato di apertura o chiusura della sidenav.
  - progressbarWatcher: Subject per la getione della *progress bar* (Un componente di Angular Material), che indica una richiesta in attesa di risposta del server.
  - idWatcher: Subject per una semplice gestione degli id, usato per mitigare alle limitazioni del Router.

Routes di Angular

Queste sono tutte le routes utlizzate nel front-end:

```
const routes: Routes = [
  {
    path: "",
    redirectTo: "login",
    pathMatch: "full"
  },
  {
    path: "login",
    component: LoginComponent,
    pathMatch: "full"
  },
  {
    path: "logout",
    component: LogoutComponent,
    pathMatch: "full"
  },
  {
    path: "waiter/new-table",
    canActivate: [AuthGuard],
    data: {
      expectedRole: "waiter"
    },
    component: WaiterNewTableComponent
  },
  {
    path: "waiter/new-order/:id",
    canActivate: [AuthGuard],
    data: {
      expectedRole: "waiter"
    },
    component: WaiterNewOrderComponent
  },
  {
    path: "waiter",
    component: WaiterDashboardComponent,
    canActivate: [AuthGuard],
    data: {
      expectedRole: "waiter"
    },
    children: [
      {
        path: "",
        component: DummyComponent,
        pathMatch: "full"
      },
      { path: ":id", component: WaiterTableDetailComponent },
      { path: "**", redirectTo: "" }
    ]
  },
  {
    path: "bartender",
    component: BartenderDashboardComponent,
```

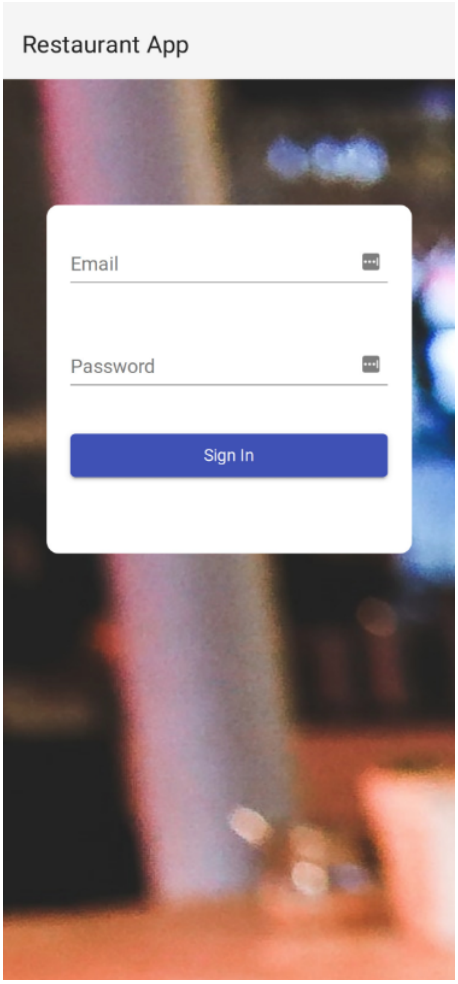
```
    canActivate: [AuthGuard],
    data: {
      expectedRole: "bartender"
    },
    children: [
      {
        path: "",
        component: DummyComponent,
        pathMatch: "full"
      },
      { path: ":id", component: BartenderOrderDetailComponent },
      { path: "**", redirectTo: "" }
    ]
  },
  {
    path: "chef",
    component: ChefDashboardComponent,
    canActivate: [AuthGuard],
    data: {
      expectedRole: "chef"
    },
    children: [
      {
        path: "",
        component: DummyComponent,
        pathMatch: "full"
      },
      { path: ":id", component: ChefOrderDetailComponent },
      { path: "**", redirectTo: "" }
    ]
  },
  {
    path: "cashier",
    canActivate: [AuthGuard],
    data: {
      expectedRole: "cashier"
    },
    children: [
      { path: "", redirectTo: "dashboard", pathMatch: "full" },
      { path: "dashboard", component: CashierDashboardComponent },
      { path: "tables/:table", component: CashierViewOrderComponent },
      { path: "tables/:table/bill", component: CashierBillComponent },
      { path: "statistics", component: CashierStatisticsComponent },
      {
        path: "statistics/users/new",
        component: CashierNewUserComponent
      },
      {
        path: "statistics/users/:user",
        component: CashierStatisticsUserComponent
      },
      { path: "**", redirectTo: "dashboard" }
    ]
  },
  { path: "**", redirectTo: "logout" }
];
```

Esempio di Workflow dell'applicazione

- **Login:**

  Questa è la pagina principale di login dell'applicazione.

  Basta riempire il form ed è possible eseguire il login.



- **Cameriere (Waiter) workflow:**

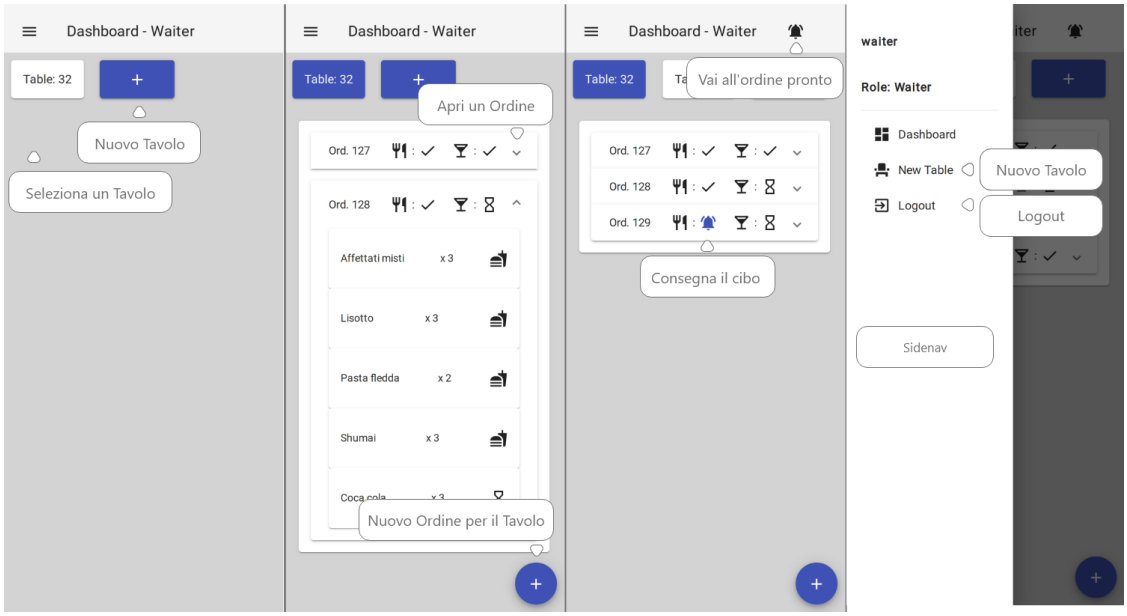
    - Dashboard:

      Qui é possibile vedere la lista dei tavoli occupati e la lista degli ordini di un selezionato tavolo.

      In basso a destra si puó aggiungere un ordine al tavolo selezionato. (2)

      Premendo sulla campanella delle notifiche si naviga verso l'ordine che é pronto. Premendo su una campanella blu si segnala la consegna del cibo (oppure bevande). (3)

      Premendo sull'icona del menù in alto a sinistra si accede alla sidenav, dove si può aggiungere un nuovo tavolo o eseguire il logout (4)



- Nuovo tavolo:

  Con il + e - si cambia il numero di clienti che occuperanno il tavolo (1)

  In basso a sinistra si può tornare alla Dashboard (1)

  Premendo su un tavolo lo si seleziona e lo si può occupare (2)

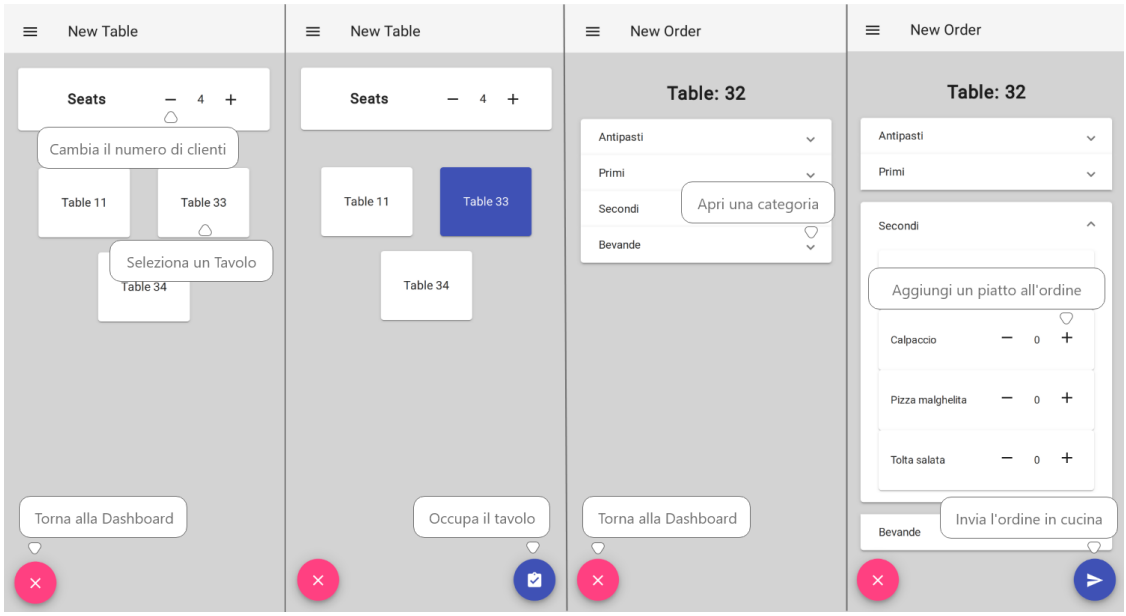
- Nuovo ordine:

  Premendo su una categoria la si può espandere (3)

  In basso a sinistra si può tornare alla Dashboard (3)

  Con il + e - si cambia il numero di piatti ordinati (4)

  In basso a destra si può inviare l'ordine alla cucina (o al bar) (4)



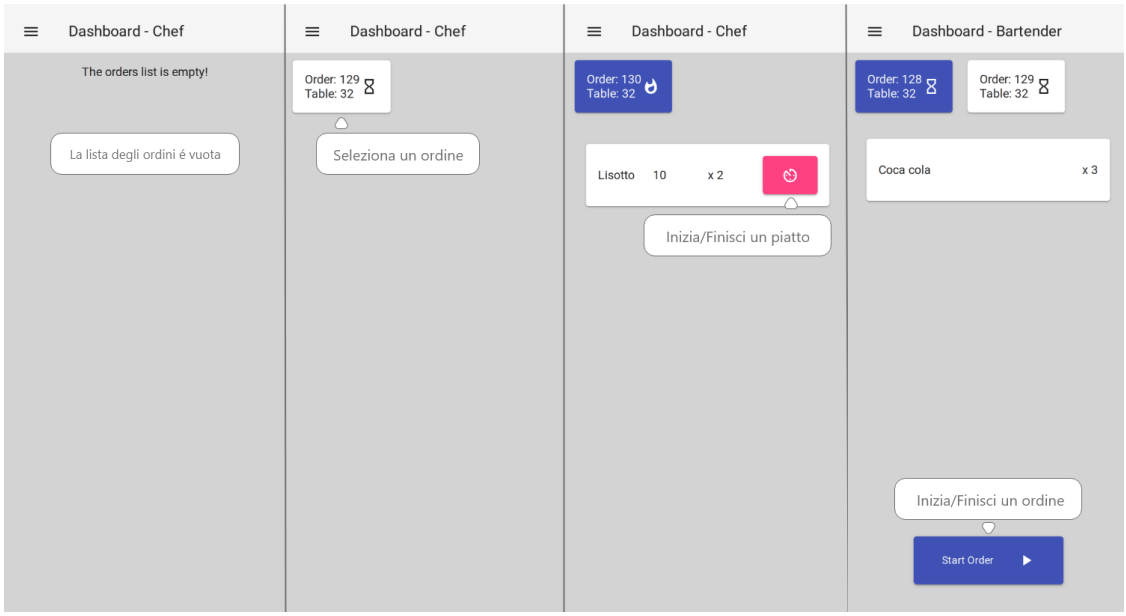
- **Cuoco (Chef) e barista (Bartender) workflow:**

    - Dashboard (1/2/3/4)

      Quando appare un ordine lo si può selezionare (2)

      Per il cuoco bisogna segnalare l'inizio e la fine di ogni piatto di un ordine per completarlo (Un piatto consiste in: nome, tempo di preparazione previsto, quantità e status) (3)

      Per il barista bisogna segnalare l'inizio e la fine di ogni ordine per completarlo (4)



- **Cassiere (Cashier) workflow:**

    - Dashboard (1/2)

      Qui é possibile vedere la lista dei tavoli occupati e la lista dei tavoli liberi. (1/2)
    - Visualizza ordini di un tavolo (3)

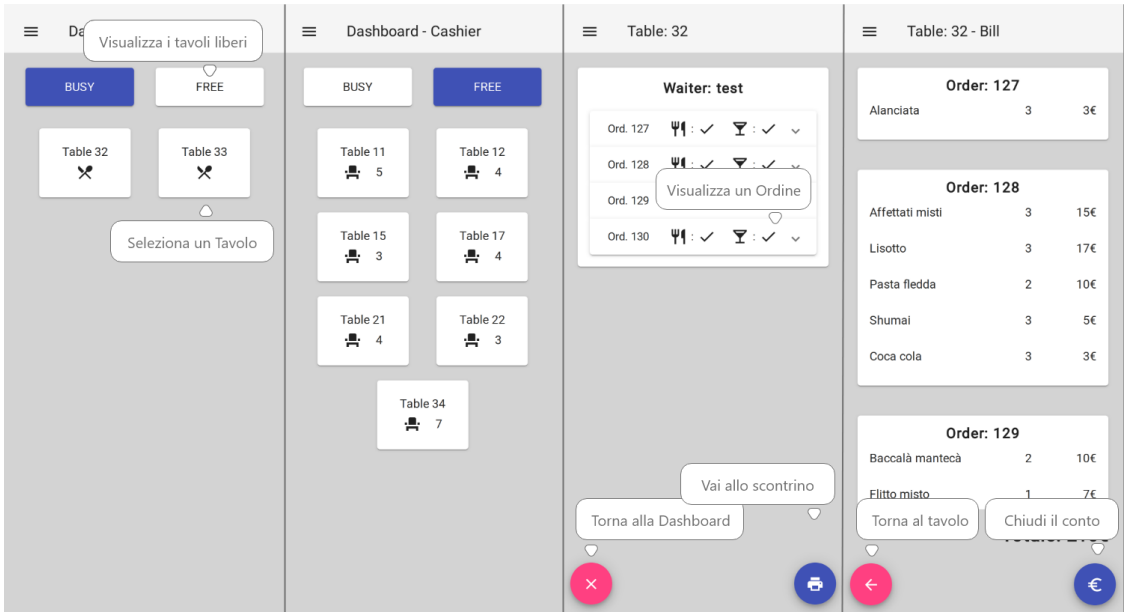
      Selezionando un tavolo occupato si può visualizzare lo stato dei suoi ordini (3)

      In basso a sinistra si può tornare alla Dashboard (3)

      In basso a destra si può andare alla pagina dello scontrino (3)
    - Visualizza lo scontrino di un tavolo (3)

      In basso a sinistra si può tornare al tavolo selezionato (4)

      In basso a destra si può chiudere il conto e liberare il tavolo (3)



Premendo sull'icona del menù in alto a sinistra si accede alla sidenav, dove si può andare nella pagina di statistiche e gestione utenti o eseguire il logout (1)

- Pagina statistiche (2)

  Si può selezionare un utente per vederne le statistiche (2)

  Si può selezionare andare alla pagina di creazione di un utente (2)

  Si può chiudere la giornata per azzerare le statistiche (2)

- Statistiche di un utente selezionato (3)

  In basso a sinistra si può tornare alla pagina delle statistiche (3)

  In basso a sinistra si può tornare alla pagina delle statistiche (3)

- Aggiunta di un nuovo utente nell'applicazione (4)

  In basso a sinistra si può inserire l'utente inserito nel form (4)

  In basso a sinistra si può tornare alla pagina delle statistiche (4)