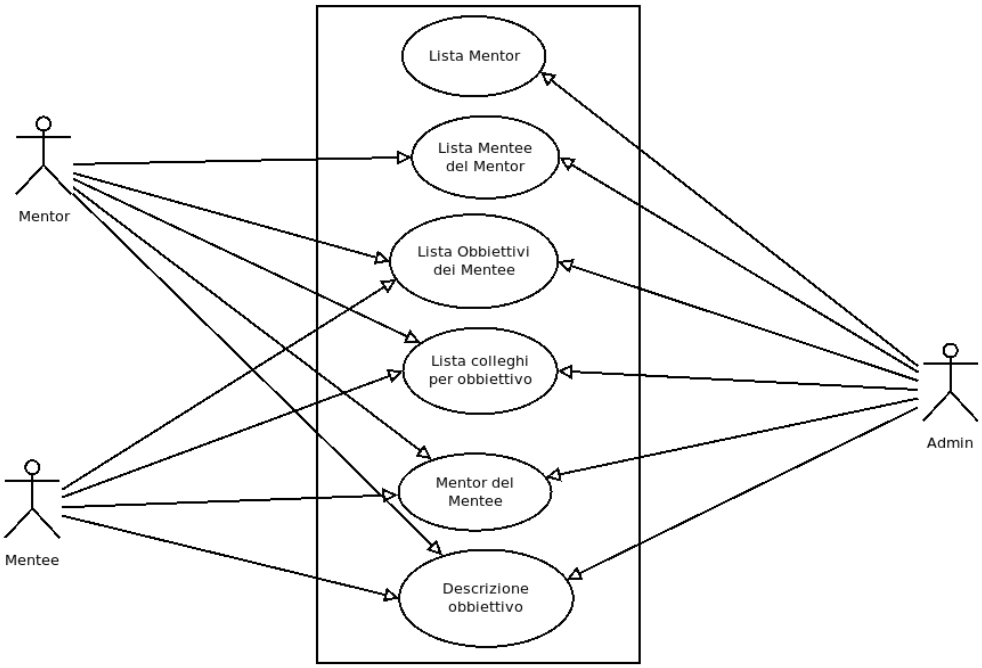
Progetto Appraisal

Casi d'Uso

Lo scopo dell'app proposta è quello di interfacciarsi al sistema di appraisal, di cui ne verrà realizzata una variante sottoforma di pagina web, in maniera tale da fornire all'utente una più veloce e comoda fruizione dell'andamento degli obbiettivi aziendali.

Possiamo notare che vi sono tre tipologie di attori coinvolti `Mentor` `Mentee` `Admin` e che le funzionalità offerte siano via via inferiori per numero in base al ruolo che ha l'utente al momento del'accesso all'app.



Dallo schema si notano le possibili funzionalità offerte per ciascuna tipologia di utente:

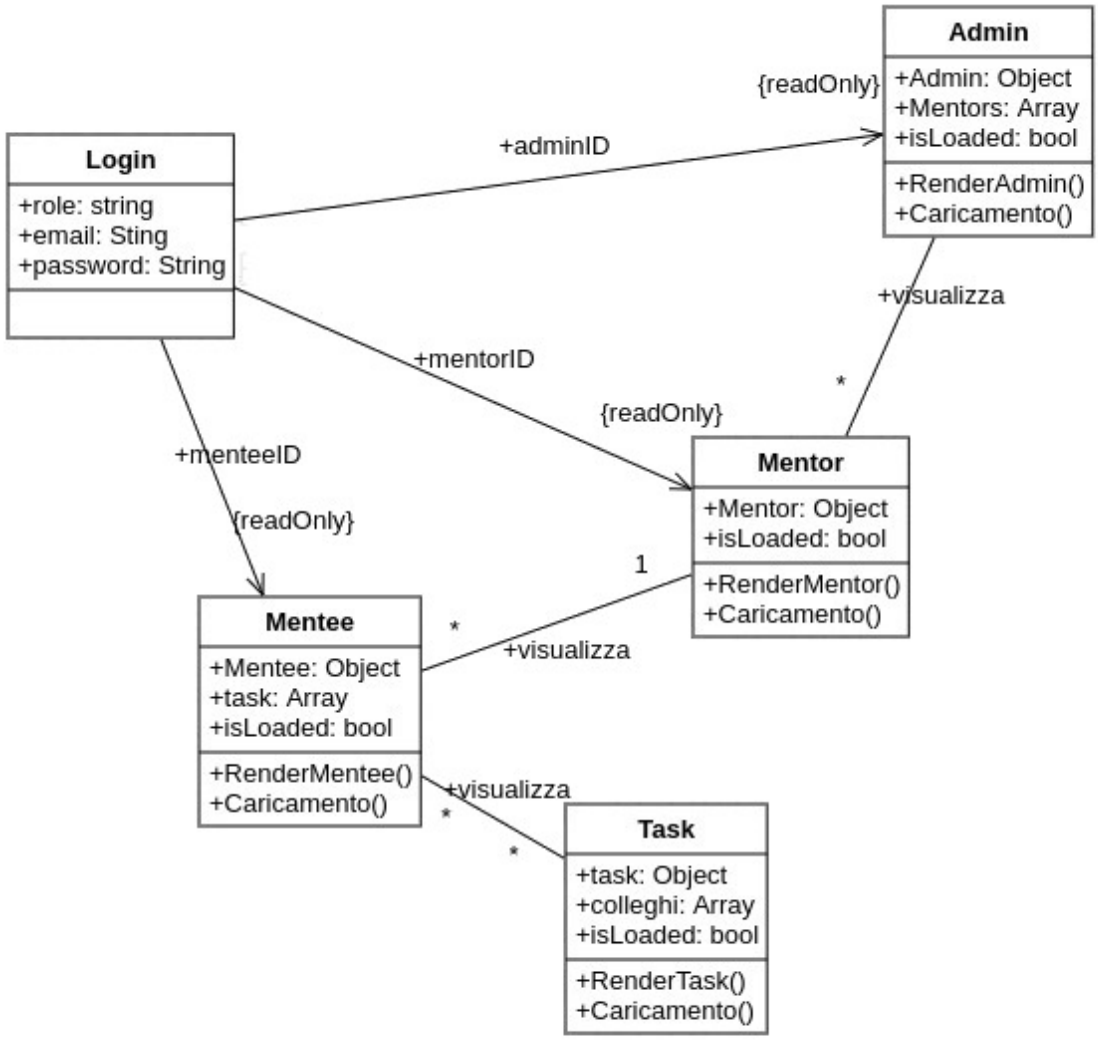
- Un Mentee può:
    - vedere gli obbiettivi ad esso assegnati
    - sapere chi altro sta lavorando all'obiettivo assegnato
    - sapere chi è il proprio Mentor
    - avere dettagli riguardanti l'obiettivo assegnato
- Un mentor può:

    - avere una lista di tutti i suoi mentee e relativi dati
    - [tutte le precedenti]
- Un Admin può:
    - avere una lista di tutti i Mentor e relativi dati
    - [tutte le precedenti]

Applicazione

L'Applicazione che si sta sviluppando è costituita da uno `Stack Screen` di pagine:

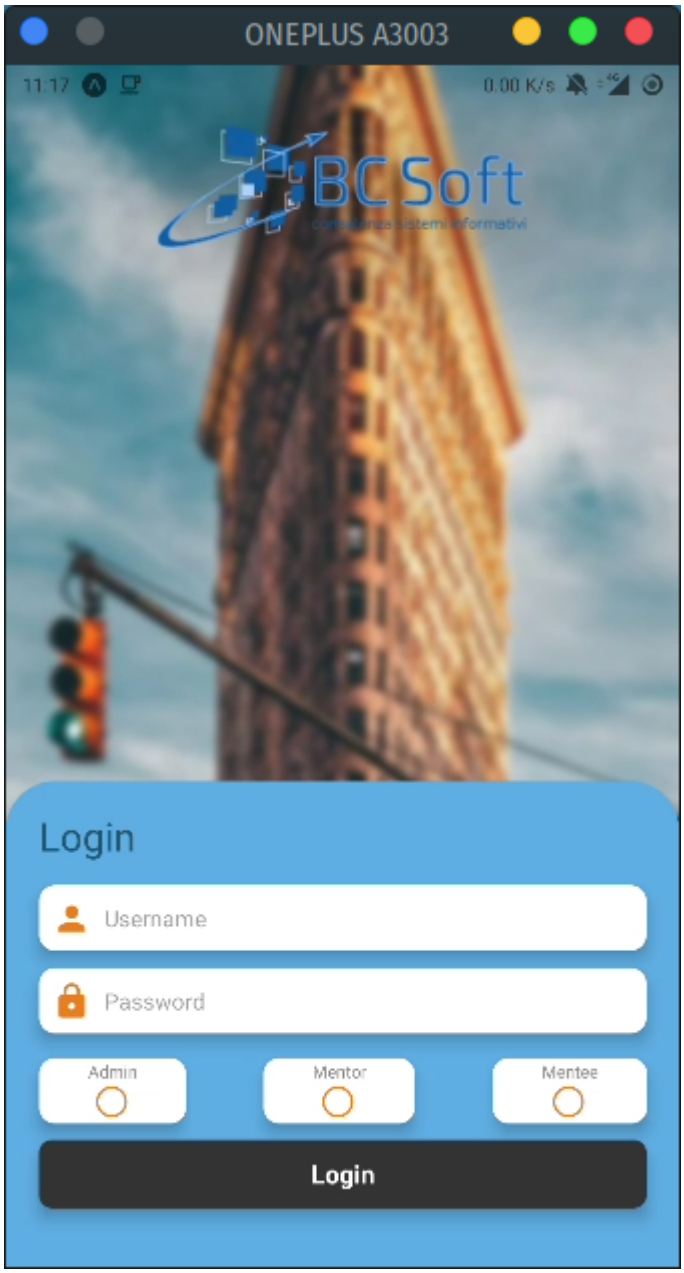
- login
- home page Admin
- home page Mentor
- home page Mentee
- pagina dedicata al singolo obiettivo



La prima schermata che si mostra all'utente è la pagina di login, questa è composta da:

- due campi in cui inserire del testo `email` e `password`,
- una serie di radio button per selezionare con quale livello si intende accedere al servizio.

Questi radio button sono fondamentali perché lo stesso utente (individuato con username & password) potrebbe avere due o più livelli di accesso, inoltre in questo modo è più semplice effettuare una query al database per sapere se i dati inseriti sono corretti o meno.



Home page Admin

Questa pagina riceve dal componente login una props che ho chiamato `adminId` , tale valore è un numero che identifica l'admin all'interno dell'azienda.

In questa, come in tutte le altre pagine, tali valori vengono utilizzati per ottenere dati dall'api, e di conseguenza aggiornare lo stato. A loro volta questo stato viene passato come props ad un altro componente che ha lo scopo di renderizzare tali dati e all'occorrenza passare ad un altra pagina.

La variabile admin è un oggetto inizializzato con la funzione `setAdmin` evocata una sola volta quando la pagina viene visualizzata, ovviamente dopo aver inviato una *request* per ottenere i dati dell'admin individuato da `adminId`.

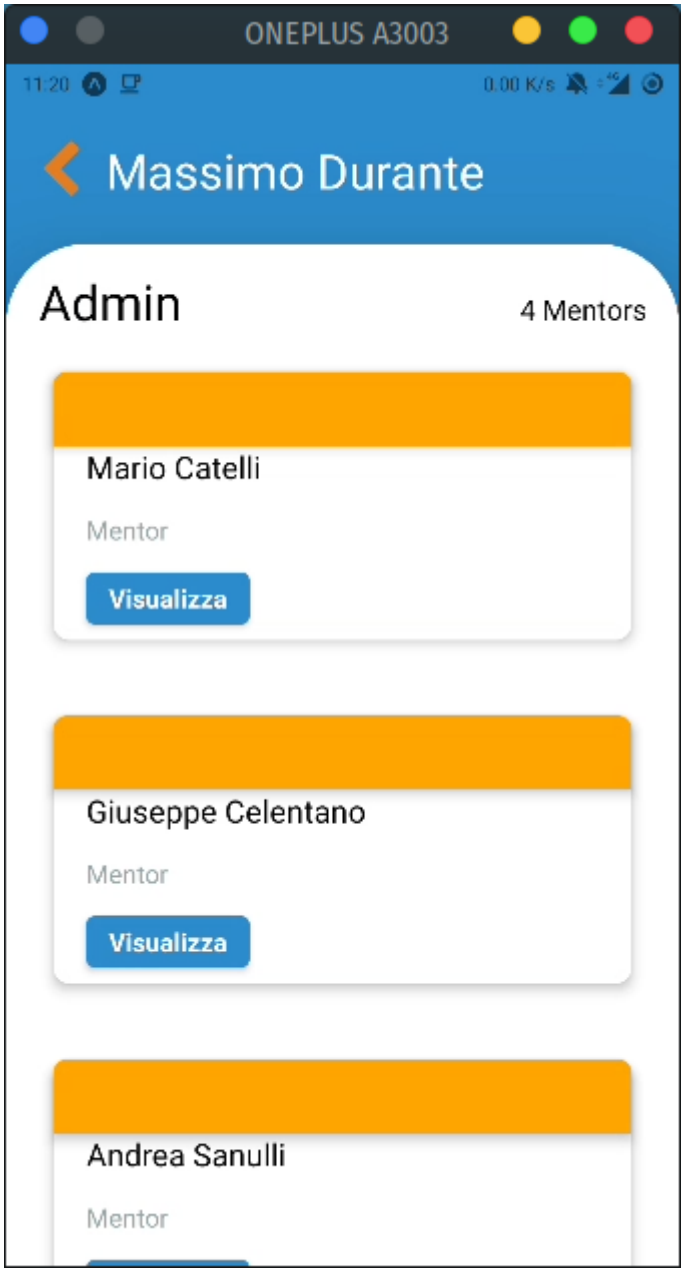
Mentre la variabile `mentors` è, invece, un array di oggetti in cui ogni oggetto rappresenta un mentor, tale array è ottenuto inviando una diversa *request*.

> I dati in esso contenuti sono gli stessi per qualsiasi admin acceda.

La variabile, infine, `isLoaded` viene utilizzata per ritardare il rendering della UI quando i dati da dover mostrare non sono ancora disponibili, ciò è dovuto all'asincronicità dello scambio dati tra client e server . Ovviamente nel mentre l'utente vedrà una schermata di caricamento.

Il rendering vero e proprio è realizzato dal componente `<RenderAdmin>`. Questi, essendo un componente separato riceve anche la props `navigation` che contiene metodi per poter cambiare pagina.

Si è voluto realizzare un componente separato per renderizzare la UI perché in tal modo è possibile separare la logica di funzionamento dall'effettivo processo di rendering dei dati.



Pagina `Admin.jsx`

```
import React from "react";
import Caricamento from "../components/caricamento";
import RenderAdmin from "../renders/renderAdmin";
```

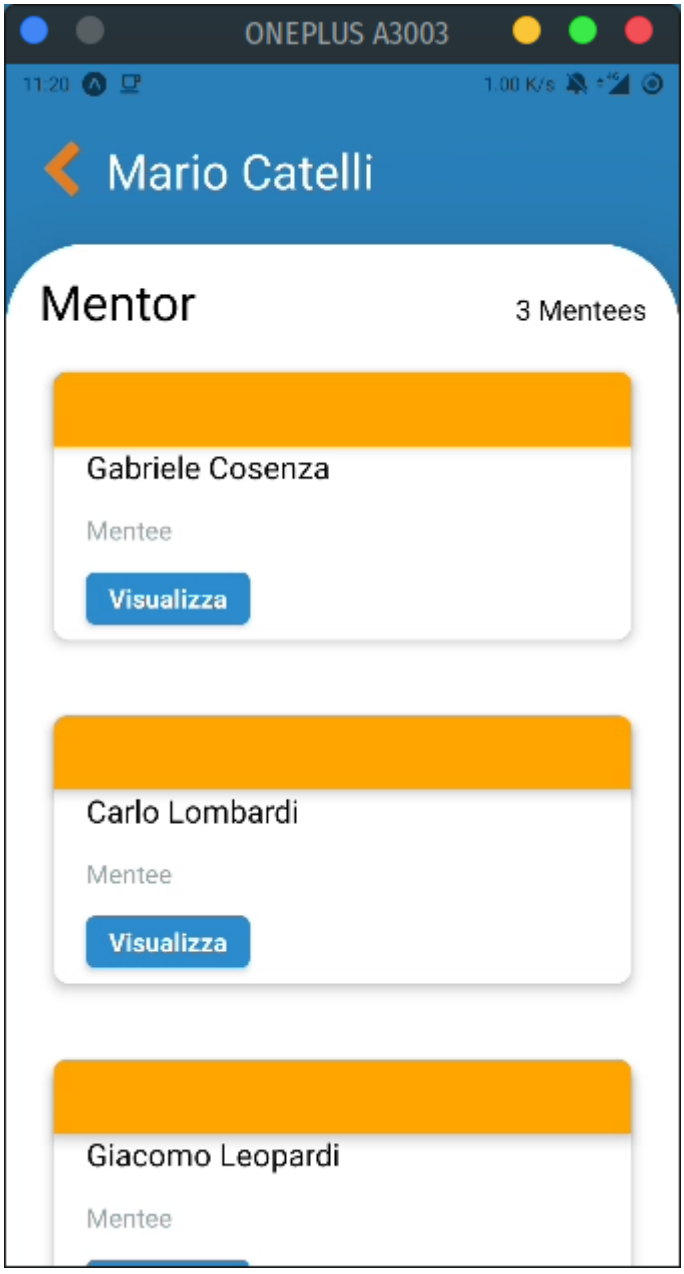
```
import HOST from "../costants/host";

const Admin = ({ route, navigation }) => {
  const [mentors, setMentors] = React.useState([]);
  const [admin, setAdmin] = React.useState({});
  const [isLoaded, setIsLoaded] = React.useState(false);

  const { adminId } = route.params;

  const URLADMIN = `${HOST}/admins`;
  const URLMENTORS = `${HOST}/mentors`;
  React.useEffect(() => {
    async function fetchData() {
      const res = await fetch(`${URLADMIN}/${adminId}`);
      const data = await res.json();
      setAdmin(data);
      const res2 = await fetch(URLMENTORS);
      const data2 = await res2.json();
      setMentors(data2);
      setIsLoaded(true);
    }
    setTimeout(() => {
      fetchData();
    }, 1000);
  }, []);

  return (
    <>
      {isLoaded ? (
        <RenderAdmin mentors={mentors} admin={admin} navigation={navigation} />
      ) : (
        <Caricamento />
      )}
    </>
  );
};

export default Admin;
```

Home Page mentor

L'home page per i mentor viene mostrata quando un admin naviga verso questa pagina, oppure quando l'utente collegato è un mentor.

Essa riceve dal componete padre una props `mentorId`, il quale è un numero che identifica il mentor all'interno dell'azienda. Questo identificativo verrà usato nella *request* per ottenere i dati del mentor in questione, ed i suoi relativi mentee.

Come nella pagina precedente, tali dati vengo utilizzati per aggiornare lo stato, in questo caso un unico oggetto. Una volta ottenuti correttamente, lo stato passa al componente `<RenderMentor>` che lo renderizzerà, viene inoltre passata la props `navigation` che contiene un metodo per navigare nella prossima pagina o tornare indietro.



Pagina `Mentor.jsx`

```
import React from "react";
import Caricamento from "../components/caricamento";
import RenderMentor from "../renders/renderMentor";
import HOST from "../costants/host";

const Mentor = ({ route, navigation }) => {
  const { userId } = route.params;
  const [utente, setUtente] = React.useState(null);
  const [loaded, setLoaded] = React.useState(false);
  const URL = `${HOST}/mentors/${userId}?_embed=mentees`;
  React.useEffect(() => {
    async function fetchData() {
      const res = await fetch(URL);
      const data = await res.json();
      setUtente(data);
      setLoaded(true);
    }

    setTimeout(() => {
      fetchData();
    }, 1000);
  }, []);

  return (
    <>
      {loaded ? (
        <RenderMentor utente={utente} navigation={navigation} />
      ) : (
        <Caricamento />
      )}
    </>
  );
};

export default Mentor;
```

Home page Mentee

Questa pagina quando viene visualizzata, riceve dal componente padre una props che ho chiamato `menteeId`, tale valore viene utilizzato in due diverse *request*:

- recuperare i dati del mentee in questione, salvando nell'oggetto `mentee`
- recuperare i dati relativi agli obiettivi a carico di tale mentee `tasks`

Tale pagina è accessibile da tutti gli admin, dal mentee corrente e dal suo mentor.

Successivamente i valori ottenuti dalle *request* vengono passati al componente `<RendeMentee>` per essere mostrati a schermo.

In seguito a tale pagina ne è prevista infine un altra che mostrerà nel dettaglio le specifiche dell'obiettivo selezionato.

```jsx
import React from "react";
import Caricamento from "../components/caricamento";
import RenderMentee from "../renders/renderMentee";
import HOST from "../costants/host";

const Mentee = ({ route, navigation }) => {
  const { menteeId } = route.params;
  const [mentee, setMentee] = React.useState({});
  const [tasks, setTasks] = React.useState([]);
  const [loaded, setLoaded] = React.useState(false);

  const URLMENTEE = `${HOST}/mentees/${menteeId}?_expand=mentor`;
  const URLTASK = `${HOST}/svolto?menteeId=${menteeId}&_expand=task`;

  React.useEffect(() => {
    async function fetchData() {
      const res = await fetch(URLMENTEE);
      const data = await res.json();
      setMentee(data);

      const res2 = await fetch(URLTASK);
      const data2 = await res2.json();
      setTasks(data2);
      setLoaded(true);
    }

    setTimeout(() => {
      fetchData();
    }, 10);
  }, []);

  return (
    <>
      {loaded ? (
        <RenderMentee mentee={mentee} tasks={tasks} navigation={navigation} />
      ) : (
        <Caricamento />
      )}
    </>
  );
};
export default Mentee;
```

Pagina Obiettivi

Tale pagina, come le altre, riceve dal componente padre una props che ho chiamato `taskId`. Tale valore identifica univocamente un obiettivo aziendale e viene adoperato in due diverse *request*:

- recuperare con più dettaglio i dati relativi al singolo obiettivo
- ottenere una lista di colleghi per tale obiettivo

Come noto un obiettivo può essere svolto da più mentee, e un mentee può intraprendere più obiettivi, per tanto questa risulta un'associazione **n -> n**, dunque è doveroso implementare una tabella di transizione accessibile all'end point `/svolto`.

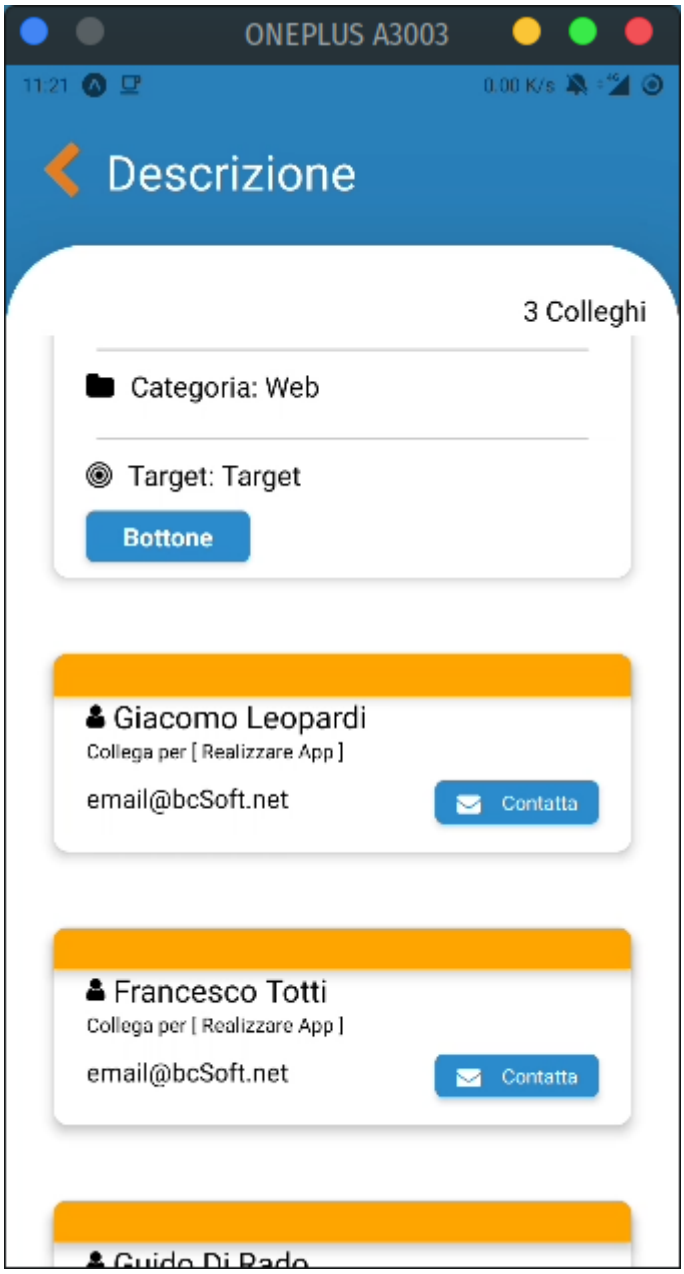
Tramite **Json-Server**, software con il quale ho reso disponibile i dati online, è possibile estendere i dati ottenuti dalle query con le keyword `_expand` e `_embed` vorrò utilizzare questa tecnica per ottenere più dati relativi, sia agli obbiettivi che hai mentee.

Gli oggetti `colleghi` e `task` vengono passati al componente `<RenderTask>` per essere mostrati all'utente, da notare che anche in questo caso viene passato la props `navigation`, che verrà usata solamente per tornare alla pagina precedente.

Si è voluto arricchire questa pagina, mostrando nome, cognome e mail di tutti i mentee che collaborano sullo stesso obiettivo, anche se rendono conto del proprio operato a diversi mentor, per favorire la comunicazione "orizzontale" tra mentee.

> Sono del parere che lo scambio di conoscenze ed esperienze acquisite sul campo, sia un momento di crescita formativa e personale estremamente importante.


ONEPLUS A3003
11:20
0.00 K/s
Descrizione
3 Colleghi
Tecnologie
Senior
Realizzare App. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Categoria: Web
Target: Target
Bottone
Giacomo Leopardi



Pagina `task.jsx`

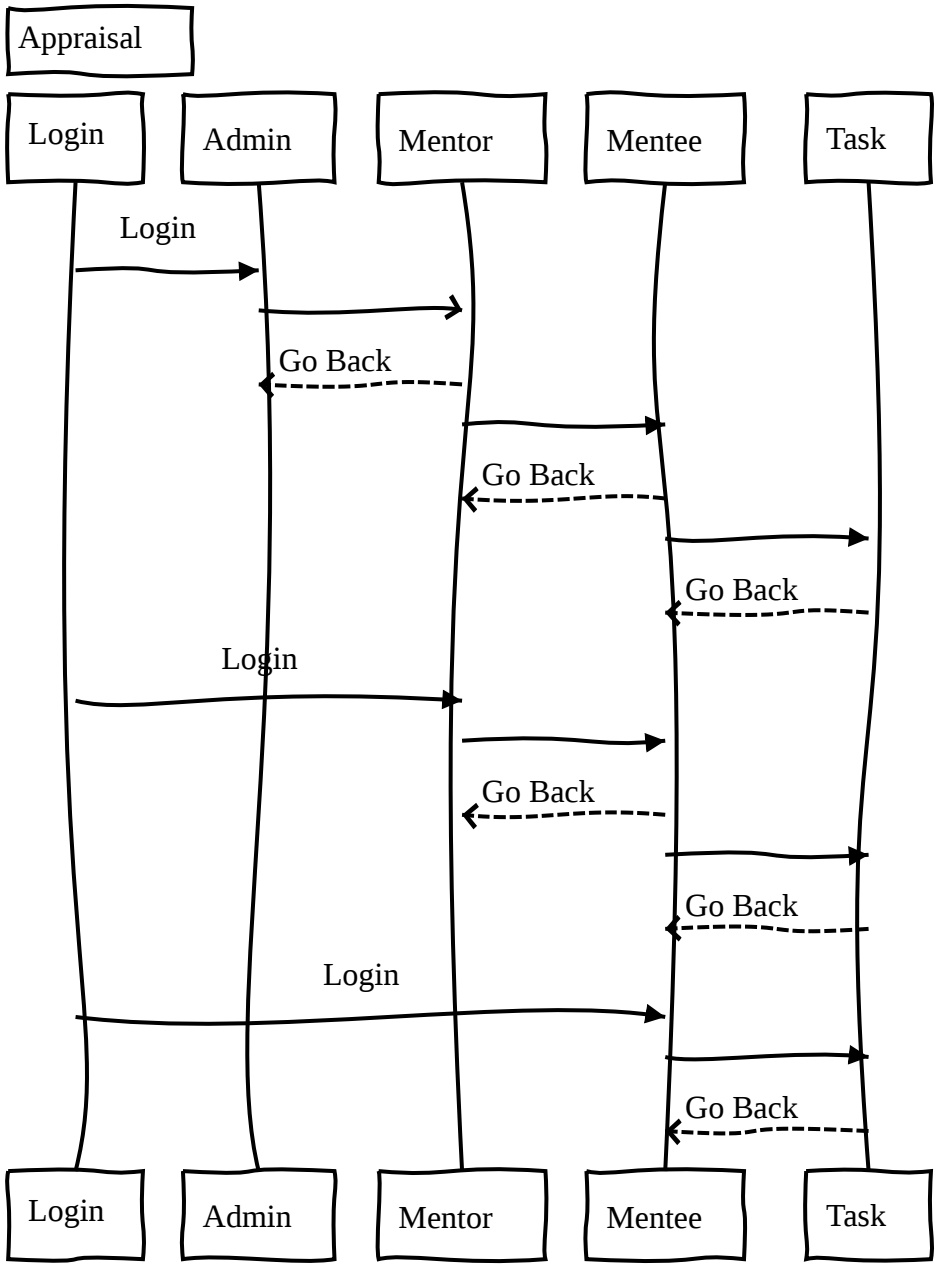
```
import React from "react";
import Caricamento from "../components/caricamento";
import RenderTask from "../renders/renderTask";
import HOST from "../costants/host";

export default Task = ({ route, navigation }) => {
  const { taskId } = route.params;
  const [colleghi, setColleghi] = React.useState([]);
  const [task, setTasks] = React.useState({});
  const [loaded, setloaded] = React.useState(false);

  const URLCOLLEGHI = `${HOST}/svolto?taskId=${taskId}&_expand=mentee&_expand=task`;
  const URLTASKS = `${HOST}/tasks/${taskId}`;

  React.useEffect(() => {
    async function fetchData() {
      const res = await fetch(URLTASKS);
      const data = await res.json();
```

```
      setTasks(data);

      const res2 = await fetch(URLCOLLEGHI);
      const data2 = await res2.json();
      setColleghi(data2);
      setloaded(true);
    }
      fetchData();
    }, 1000);
  }, []);

  return (
    <>
      {loaded ? (
        <RenderTask colleghi={colleghi} task={task} navigation={navigation} />
      ) : (
        <Caricamento />
      )}

    </>
  );
};
```

Navigazione

La navigazione all'interno dell'app è gestita con uno schema a `Stack` quindi in qualsiasi pagina ci si trovi premendo **indietro** si tornerà alla View precedente.

Questo sistema mi permette di "far iniziare" la navigazione da una qualsiasi pagina purché si riescano a fornire a tale pagina i dati necessari affinché quest'ultima possa iniziare il processo di recupero dei dati e successiva renderizzazione.



API

Al momento della stesura di questo documento l'api è disponibile online, per consultazione e testing di qwery tutti i dati sono liberamente accessibili all'end point `https://my-json-server.typicode.com/Aldodo91/Json-Server`.

L'API proposta è realizzata, come detto, con **Json-Server**, per tanto supporta i più comuni metodi HTTP `GET` `PUT` `POST` `DELETE`, farò riferimento a tale URL con il nome `HOST`.

Login come ADMIN

Quando un admin logga correttamente il server dovrà restituire, un oggetto json

```
{
  "adminId": 1
}
```

Il valore associato ad **adminId** verrà usato in chiamata `fetch` per conoscere le informazioni di tale admin (nome, cognome , ecc..)

`HOST/admins/${adminId}`

```
{
  "nome": "Massimo",
  "cognome": "Durante",
  "email": "email@bcSoft.net",
  "tipoUtente": "Admin",
  "id": 1
}
```

Inoltre gli admin vedono la lista dei **mentors** accessibile all'url

`HOST/mentors` tale url riposta un array di tutti i mentor.

```
[
    {
      "nome": "Martin",
      "cognome": "Pescatore",
      "email": "email@bcSoft.net",
      "tipoUtente": "Mentor",
      "id": 4
    }
]
```

Scegliendo su un particolare mentor si può vedere le info di tale mentor ad esempio:

`HOST/mentors/1`


```
{
  "nome": "Mario",
  "cognome": "Catelli",
  "email": "email@bcSoft.net",
  "tipoUtente": "Mentor",
  "id": 1
}
```


Un admin, inoltre, può arricchire la risposta dal server usando l'end point seguente per vedere tutti i mentor con annessi i mentee. Di seguito mostro il mentor (id 4) con mentee `HOST/mentors/4?_embed=mentees`


```
{
    "nome": "Martin",
    "cognome": "Pescatore",
    "email": "email@bcSoft.net",
    "tipoUtente": "Mentor",
    "id": 4,
    "mentees": [
      {
        "nome": "Camilla",
        "cognome": "Conte",
        "email": "email@bcSoft.net",
        "tipoUtente": "Mentee",
        "mentorId": 4,
        "id": 12
      },
      {
        "nome": "Fillippa",
        "cognome": "Lienz",
        "email": "email@bcSoft.net",
        "tipoUtente": "Mentee",
        "mentorId": 4,
        "id": 13
      }
    ]
  }
```

Login come MENTOR

Quando accede un Mentor o un Admin naviga in tale pagina viene passato un oggetto json

```
{
  "mentorID": 1
}
```

Questo oggetto verrà utilizzato nell'unica chiamata Api prevista, ad oggi, per tale pagina.

`HOST/mentors/${mentorID}?_embed=mentees`

```
{
    "nome": "Martin",
    "cognome": "Pescatore",
    "email": "email@bcSoft.net",
    "tipoUtente": "Mentor",
    "id": 4,
    "mentees": [
      {
        "nome": "Camilla",
        "cognome": "Conte",
        "email": "email@bcSoft.net",
        "tipoUtente": "Mentee",
        "mentorId": 4,
        "id": 12
      },
      {
        "nome": "Fillippa",
        "cognome": "Lienz",
        "email": "email@bcSoft.net",
        "tipoUtente": "Mentee",
        "mentorId": 4,
        "id": 13
      }
    ]
  }
```

Seguendo tale logica i mentor quando effettuano l'accesso, vedono il loro `nome` `cognome` ed altri dati, inoltre essi vedono anche una lista di mentees associati.

Login come MENTEE

L'home page dei Mentee, che viene visualizzata quando il mentor o un admin vi naviga oppure quando ad accedere è un mentee, riceve come parametro un oggetto json

```
{
  "menteeId": 1
}
```

Questo valore verrà usato in due chiamate Api:

- Dati utente

`HOST/mentees/1?_expand=mentor`

```
{
  "nome": "Gabriele",
  "cognome": "Cosenza",
  "email": "email@bcSoft.net",
  "tipoUtente": "Mentee",
  "mentorId": 1,
  "id": 1,
  "mentor": {
    "nome": "Mario",
    "cognome": "Catelli",
    "email": "email@bcSoft.net",
    "tipoUtente": "Mentor",
    "id": 1
  }
}
```

Così da sapere le informazioni circa il mentee loggato e quelle relative al suo mentor.

- Obiettivi a carico

`HOST/svolto?menteeId=${menteeId}&_expand=task`

```
[
  {
    "menteeId": 1,
    "taskId": 1,
    "id": 1,
    "task": {
      "descrizione": "Fornire API. Lorem ipsum..",
      "gruppo": "Junior",
```

```
        "ambito": "Tecnologie",
        "area": "Informatica",
        "categoria": "Web",
        "target": "Target",
        "id": 1
         }
    }
]
```

Query utili

L'Api inoltre fornisce altri end point utili:

- Tutti i mentee che lavorano su uno stesso obbiettivo comune

`${HOST}/svolto?taskId=${taskId}&_expand=mentee&_expand=task`


```
[
  {
    "menteeId": 2,
    "taskId": 3,
    "id": 8,
    "mentee": {
      "nome": "Carlo",
      "cognome": "Lombardi",
      "email": "email@bcSoft.net",
      "tipoUtente": "Mentee",
      "mentorId": 1,
      "id": 2
    }
  },
  {
    "menteeId": 7,
    "taskId": 3,
    "id": 17,
    "mentee": {
      "nome": "Jessica",
      "cognome": "Fletcher",
      "email": "email@bcSoft.net",
      "tipoUtente": "Mentee",
      "mentorId": 3,
      "id": 7
    }
  },
  {
    "menteeId": 8,
    "taskId": 3,
    "id": 18,
    "mentee": {
      "nome": "Francesco",
      "cognome": "Totti",
      "email": "email@bcSoft.net",
      "tipoUtente": "Mentee",
      "mentorId": 3,
      "id": 8
    }
  }
]
```

Popolazione database

Il file json che ho utilizzato ha una dimensione di *12 KB*, ed è composto da 5 endpoint:

- admins
- mentors
- mentees
- tasks
- svolto

Ciascuno dei quali è un array di oggetti json.

`/admins`

```
{
  "id": 1, // id dell'admin
  "nome": "Nome",
  "cognome": "Cognome",
  "email": "email@...",
  "tipoUtente": "Admin"
}
```

`/mentors`

```
{
  "id": 1, // id dell'mentor
  "nome": "Nome",
  "cognome": "Cognome",
  "email": "email@...",
  "tipoUtente": "Mentor"
}
```

`/mentees`

```
{
  "id": 1, // id dell'mentee
  "mentorId": 2, // importante -> id del mentor
  "nome": "Nome",
  "cognome": "Cognome",
  "email": "email@...",
  "tipoUtente": "Mentee"
}
```

`/tasks` (la tabella obbiettivi)

```
{
  "id": 1,
  "descrizione": "fornire API",
  "gruppo": "Gruppo",
  "ambito": "Ambito",
  "Area": "Area",
  "categoria": "Categoria",
  "target": "Target"
}
```

`/svolto` (tabella di transizione)

```
{
    "id":1  //semplice id in autoincremento,
    "menteeId": 2,
    "taskId": 3,
}
// In questo caso il mentee con id 2 sta facendo l'obbiettivo 3
```