RELAZIONE

PRIMA PARTE

Nella prima parte del percorso, ho avuto modo di studiare e apprendere tutte le conoscenze utili ai fini del progetto. Qui di seguito riporto la prima relazione, consegnata dopo l'apprendimento di tali conoscenze.

----------**DASHBOARD**

Nella Dashboard troviamo le informazioni generali del profilo worker.
Il workerData lo si estrae tramite un Custom Hook denominato useWorkerData che utilizza getRequest in /workers

```
const fetchWorkerData = async () => {
      try {
        const res = await getRequest("/workers", {
          headers: {
            Authorization: `Bearer ${jwt}`,
          },
        });
```

Va quindi a prendere i dati restituiti dalla richiesta al server e setta il workerData tramite setWorkerData. Successivamente chiama la funzione getBoostStatus ed altre funzioni per andare a fare un controllo dell'iscrizione a Boost.

```
if (res.status === 200) {
          setWorkerData(res.data);
          await getBoostStatus(res.data.user_id);
        }
```

In Dashboard troviamo una serie di getRequest, che richiedono l'asset e il research status. In questo modo occupiamo la prima parte del sito in cui si mostra il nome dell'utente, se ha il profilo boost e il suo status.

Scorrendo troviamo la sezione relativa alle offerte disponibili per l'utente, utilizzando una Callback e una richiesta getRequest per vedere se è disponibile almeno un'offerta di lavoro. Nel caso questa richiesta restituisca almeno un'offerta:

```
 if (res.status === 200 && res.data.length > 0) setNewJobOffersPresence(true);
```

ci ritornerà la sezione con un button che porta alle candidature.

In fondo alla pagina troviamo invece l'anteprima del CV nativo dell'utente, gestito da da una CtaBlock per l'intestazione e da una componente chiamata CVBlock che riceve come props i dati dell'user, dei worker e gli assets.

----------**CANDIDATURE**

All'interno della pagina /candidature troviamo la sezione relativa alle nuove offerte di lavoro per cui l'utente è idoneo, le candidature attive e le offerte archiviate.

Tramite una callback, utilizziamo una getRequest per fetchare le nuove offerte:

```
const res = await getRequest(
        `/worker_job_positions/job_positions?limit=10&offset=0&status_key=worker_jp_propos
ed`
```

Un'altra per fetchare tutte le offerte archiviate:

```
const res = await
getRequest("/worker_job_positions/job_positions?limit=10&offset=0&sentiment=positive"
```

*Una chiamata patchRequest per aggiornare lo status del worker con la posizione lavorativa

```
const updateWorkerJobPositionStatus = async (interested: boolean, job_position_id: string)
=> {
    setLoading(true);
    const formData = new FormData();
    formData.append("job_position_id", job_position_id);
```

```
    formData.append("status_key", interested ? "worker_jp_interested" :
"worker_jp_not_interested");
    formData.append("worker_id", workerData?.user_id ?? "");

    try {
      const { data } = await patchRequest(`/worker_job_positions/update_status`, formData,
{
```

Vengono gestite poi tutte le interazioni con i bottoni presenti all'interno delle card delle offerte/candidature di lavoro.
Ogni card presenta 1-2 bottoni che richiamano la funzione handleModalSelect a cui viene passato come parametro l'azione eseguita che ricade in queste tipologie di azioni:

```
export type Actions =
  | "worker_jp_proposed"
  | "set_interest"
  | "set_no_interest"
  | "interested_again"
  | "not_interested_anymore"
  | "offer_not_available"
  | "interest_success"
  | "no_interest_success"
  | "set_videos";
```

C'è una funzione chiamata handleCallback che riceve come parametro l'azione eseguita e il lavoro selezionato, con uno switch delle varie Actions richiamiamo la funzione di set dello stato relativa al case di nostro interesse, che apre il modal relativo all'azione selezionata.

```
case "set_no_interest":
        setModalSelected("set_no_interest");
        break;
```

il case relativo all'Action "interested_again" è l'unico che chiama invece la funzione che cambia lo Status del worker con la posione di lavoro.*

```
case "interested_again":
        updateWorkerJobPositionStatus(true, jobPosition.id);
        break;
```

Non mi è chiara al 100% questa funzione:

```
useEffect(() => {
    const { job_position_id } = router.query;
    if (!job_position_id) return;

    setTimeout(() => {
      const section = document.querySelector("#candidature-attive");
      scrollIntoViewOffset(section, 0);
    }, 300);

    const allJobOffers = [...activeOffers, ...archivedOffers];

    setTimeout(() => {
      handleCallback("set_videos", allJobOffers.find((job) => job.job_position.id ===
job_position_id)?.job_position);
    }, 800);

    setJobSelected(allJobOffers.find((job) => job.job_position.id ===
job_position_id)?.job_position);

    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [router.query, activeOffers, archivedOffers]);
```

----------**OFFERTE DI LAVORO**

La sezione “offerte di lavoro” comprende una lista di tutte le offerte di lavoro presenti nel sistema. Qui si può filtrare la ricerca in base al settore, al periodo stagionale e non, in Italia o all’estero e un input text che cerca per città.

La funzione principale è “fetchJobOffer” che modifica la query base:

```
let query = `/employers/job_positions/workers?offset=${offset}&limit=9`;
```

aggiungendo, per ogni filtro selezionato, un “pezzo” in più alla query. Esempio, se il filtro “Italia” o “Estero” sono selezionati avremo:

```
if (activeFilters.italy !== activeFilters.abroad) {
        query += `&country=${activeFilters.italy ? "Italy" : "Abroad"}`;
      }
```

A query compilata avremo la getRequest:

```
try {
        const res = await getRequest(query, {
          headers: {
            Authorization: `Bearer ${jwt}`,
          },
        });
```

La selezione di ogni filtro farà partire la funzione handleFiltering che aggiunge allo stato activeFilters il nuovo filtro.

```
const handleFiltering = (variable: FilterVariables, value: string | boolean) => {
    setOffset(0);
    setActiveFilters((prev) => ({ ...prev, [variable]: value }));
  };
```

Nel caso sia stata effettuata una candidatura per un certo lavoro, partirà un countdown di 24 ore nel quale non sarà possibile effettuare ulteriori candidature. La funzione che regola se bloccare o no le nuove candidature è la funzione checkApplicationStatus che richiama una funzione get per controllare lo status corrente:

```
getRequest(`/workers/check_application_status?worker_id=${worker_id}`
```

Al termine del coundown, si ritorna all’impostazione base con:

```
if (counter < 0) {
      clearInterval(timer);
      setCountdownTimer(null);
      return;
    }
```

SECONDA PARTE

Nella seconda parte del percorso, sono stato incaricato della mia prima task.
Il mio obiettivo era quello di migliorare/aggiornare una sezione della piattaforma, nella quale venivano mostrate delle card con le descrizioni di ogni offerta lavorativa.

- Ho avuto modo di imparare tanto per quanto concerne lo styling di un'applicazione web e i relativi componenti.
- Attraverso l'utilizzo della piattaforma Figma, avevo sempre a portata di mano un esempio grafico di ciò che avrei dovuto ricreare
- Attraverso le piattaforme BitBucket e Jira, vedevo il compito a me assegnato e caricavo le bozze per farle controllare
-