Corso di Laurea Magistrale in
Ingegneria e Scienze Informatiche

Applicazioni e Servizi Web

a.a. 2019-2020

RutWorking

Paolo Baldini
paolo.baldini6@studio.unibo.it
matricola: 0000949248

Marco Meluzzi
marco.meluzzi@studio.unibo.it
matricola: 0000936082

Lorenzo Sutera
lorenzo.sutera@studio.unibo.it
matricola: 0000950056

2 settembre 2020

Indice

1 Introduzione

Il progetto consiste nalla realizzazione di un'applicazione client-server, basata su stack MEVN (MongoDB / Express / VueJS / Node.js), per supportare lo sviluppo collaborativo di sistemi software in un contesto aziendale.
La piattaforma si propone come punto di organizzazione e **coordinazione delle attività di sviluppo** software fra componenti di un progetto. Questi avranno la possibilità di comunicare tramite essa e organizzare lo svolgimento delle attività. Come elemento fondamentale dell'applicazione ci saranno i *progetti*, ognuno dei quali potrà essere costituito da più *moduli*. Lo sviluppo di un modulo di progetto verrà assegnato ad un team costituito da un numero variabile di sviluppatori. Ogni modulo sarà quindi un punto di coordinamento nello sviluppo, integrando una propria *chat* in cui gli sviluppatori potranno comunicare e un *Kanban* su cui organizzare le operazioni/compiti da portare a termine.
Gli sviluppatori potranno ovviamente lavorare su più moduli e progetti contemporaneamente e i risultati da essi ottenuti (completamento di task del kanban) concorreranno alla creazione di una *graduatoria dei migliori sviluppatori* della ditta.

2 Requisiti

Descrizione delle caratteristiche e funzionalità che il sistema prevede.

i. registrazione utente
ii. login utenti / admin
iii. creazione progetto
iv. creazione modulo
v. creazione teams
vi. implementazione del Kanban
vii. accettazione compiti (elemento sul kanban)
viii. annullamento compiti
ix. assegnamento forzato di un compito (da parte del referente del modulo)
x. chat di modulo
xi. classifica miglior sviluppatore (gamification)
    - premi in punti per completamento di task
    - penalità per task eseguiti in maniera superficiale
xii. rimozione utente dal servizio
xiii. notifiche

2.1 Note e approfondimenti:

i - registrazione utente: essendosi scelto di sviluppare un'applicazione collaborativa orientata all'utilizzo aziendale, si è ritenuto adeguato assegnare la possibilità di registrazione di un utente ad un amministratore generale.
Concettualmente, all'assunzione dell'utente verrà creato un'apposito indirizzo email aziendale, il quale verrà dunque utilizzato per la registrazione del dipendente nell'applicativo web. La registrazione consisterà dunque nella generazione di una password da parte del sistema che verrà quindi fornita all'utente e che potrà successivamente essere cambiata dallo stesso.
Abbiamo ritenuto che questa pratica possa permette all'azienda di mantenere un ferreo controllo sugli utenti del sistema, aumentandone teoricamente l'efficienza.

iii - creazione progetto: la creazione di progetti è una funzionalità disponibile a qualsiasi utente. Una volta che la creazione verrà effettuata, l'utente creatore diventerà capo del progetto e sarà l'unico col permesso di creare dei moduli di lavoro in esso.

iv - creazione modulo: la creazione di moduli è una funzionalità disponibile solo al capo del progetto. Alla creazione, l'utente creatore potrà scegliere di assegnarne la gestione ad un diverso utente o di assumerne esso stesso la guida.

v - creazione teams: un team è un insieme di utenti assegnati al completamento di un modulo. Il concetto stesso di team viene rappresentato nel sistema dall'insieme di sviluppatori del modulo, che saranno dunque guidati dal capo del team, rappresentato dal dirigente del modulo.

vi, vii, viii, ix - Kanban: questo conterrà i task che gli sviluppatori dovranno svolgere per il completamento del modulo di progetto. Questi saranno inizialmente aggiunti in uno stato *TO-DO*, che potrà poi passare ad *ASSIGNED* quando un utente ne accetterà lo sviluppo (o quando gli sarà assegnato dal capo modulo), *IN-PROGRESS* quando l'utente comincerà effettivamente lo sviluppo e *DONE* quando questo sarà terminato. Sarà eventualmente possibile annullare l'assegnazione di un task (da parte del dirigente del modulo o per rinuncia esplicita da parte dello sviluppatore precedentemente incaricato).

xi - classifica miglior sviluppatore: considerando l'interesse di una ditta ad incentivare il lavoro dei propri dipendenti, si è scelto di introdurre nel servizio la possibilità di creare una classifica dei migliori impiegati. Questa pratica vuole stimolare una sana competitività fra i dipendenti, incentivandoli a competere per raggiungere i risultati migliori e, eventualmente, premi forniti dalla ditta.
La classifica si basa dunque sui compiti completati da ciascun impiegato nei vari progetti. Per ogni task risolto verrà assegnato un punteggio allo sviluppatore, che potrà così scalare la classifica. Per scoraggiare atteggiamenti insalubri,

inoltre, il sistema è stato strutturato in modo che approcci superficiali al completamento dei task, che portino dunque il compito a dover essere successivamente rivisto, portino a variazioni negative del punteggio.

xii - rimozione utente dal servizio: l'amministratore del servizio ha la possibilità di rimuovere un utente dallo stesso, a seguito di eventi quali il licenziamento, mantenendo tuttavia la possibilità di visualizzarne i dati inerenti.

xiii - notifiche: nel sistema è presente la possibilità di visualizzare notifiche. Queste sono relative alla pubblicazione di messaggi in chat, aggiunta di uno sviluppatore ad un modulo di progetto e al completamento di un task (quest'ultima ricevuta solo dal capo del modulo).

3 Design

Il design dell'architettura del sistema è stata concepita cercando di coinvolgere il più possibile gli utenti, in un'ottica di *User Center Design*. Inizialmente l'approccio è stato adottato senza un effettivo coinvolgimento degli utenti reali, creando piuttosto dei modelli di possibili utilizzatori del sistema (*personas*) e avvalendosi di *mockup* dell'interfaccia utente. In fase di testing, invece, il contributo degli utenti è stato decisivo per cercare di risolvere le principali problematiche emerse e migliorare il più possibile la *user experience*.

3.1 Personas

Andrea è uno sviluppatore della ditta. Normalmente, ha un sacco di compiti assegnati da svolgere e spesso si appunta note su post-it, creando non poco disordine. Considerando poi che la ditta sviluppa in maniera Agile, questi compiti richiedono spesso una certa dose di comunicazione con i colleghi, obbligando Andrea a girovagare per l'ufficio perdendo non poco tempo.
Da quando l'azienda ha acquistato una licenza per il software organizzativo *RutWorking*, Andrea non deve più appendere post-it sulla scrivania e neppure girovagare per l'ufficio per comunicare, preferendo invece l'utilizzo di chat e kanban.

Antonio è uno sviluppatore. A differenza di Andrea, non ha problemi a girovagare in giro e avere confusione intorno. Antonio è infatti poco motivato nel lavoro e quindi poco gli importa di svolgere i compiti in maniera oculata. Il lavoro non lo stimola e la ditta sarebbe disposta a investire qualche risorsa per mantenere attivo lo sviluppatore che, invero, conosce molto bene i vari progetti intrapresi nel corso degli anni e quindi risulta difficilmente rimpiazzabile.
Con l'acquisto del software, la ditta trova la soluzione al disinteresse del soggetto. Andrea è infatti molto competitivo e la graduatoria del miglior sviluppatore lo diverte e lo incita a lavorare.

Cosimo è il capo di un progetto e deve gestirne lo sviluppo delle varie sottoparti. Questo è particolarmente impegnativo in quanto il progetto è di notevoli dimensioni. Cosimo vorrebbe quindi poter definire degli assistenti che si occupino di gestire delle sottoparti del progetto, lasciando a lui la possibilità di suprevisionare tutto in maniera più generale.
Con l'adozione del software *RutWorking* Cosimo crea quindi il suo progetto e assegna vari moduli da completare ai suoi assistenti.

Giovanni è un assistente di Cosimo e si occupa di dirigere la realizzazione del modulo a lui assegnato. Dopo varie riunioni identifica i vari task che devono essere completati e li pubblica quindi sul kanban, in attesa che gli sviluppatori se ne facciano carico.
Giovanni è però contrariato dal fatto che nessuno sviluppatore accetti *quel* compito particolarmente difficile che ha pubblicato. Per questo decide di assegnarlo obbligatoriamente ad Antonio per essere sicuro che venga svolto.

Anna è la segretaria che si occupa delle assunzioni. Nella ditta è colei che ha l'account amministratore e che quindi si occupa di registrare i nuovi dipendenti. Prima di tutto registra per ognuno l'email aziendale che verrà utilizzata dai clienti per contattare i vari teams di sviluppo. Succesisvamente, inserisce nel software l'email del dipendente in modo che possa successivamente essere assegnato ai vari progetti della ditta.

3.2 Mockup

Sono stati realizzati i *mockup* delle pagine e delle funzionalità più importanti del sistema, seguendo un approccio *mobile first* e adottando la tecnica del *responsive design*.

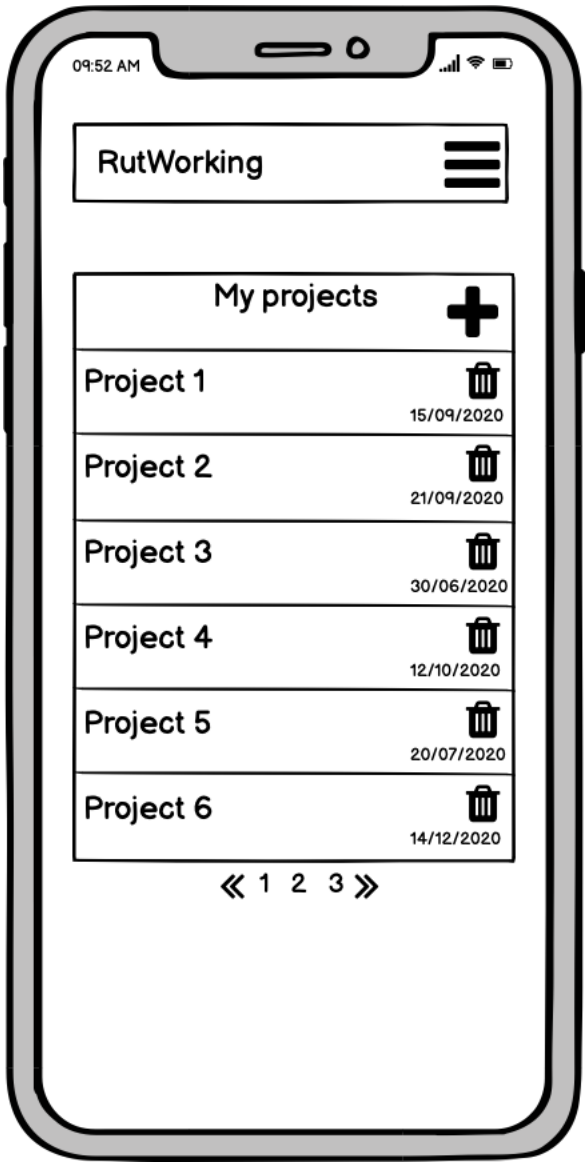

Figura 1: Home page (versione mobile). Lista dei progetti a cui l'utente loggato collabora.

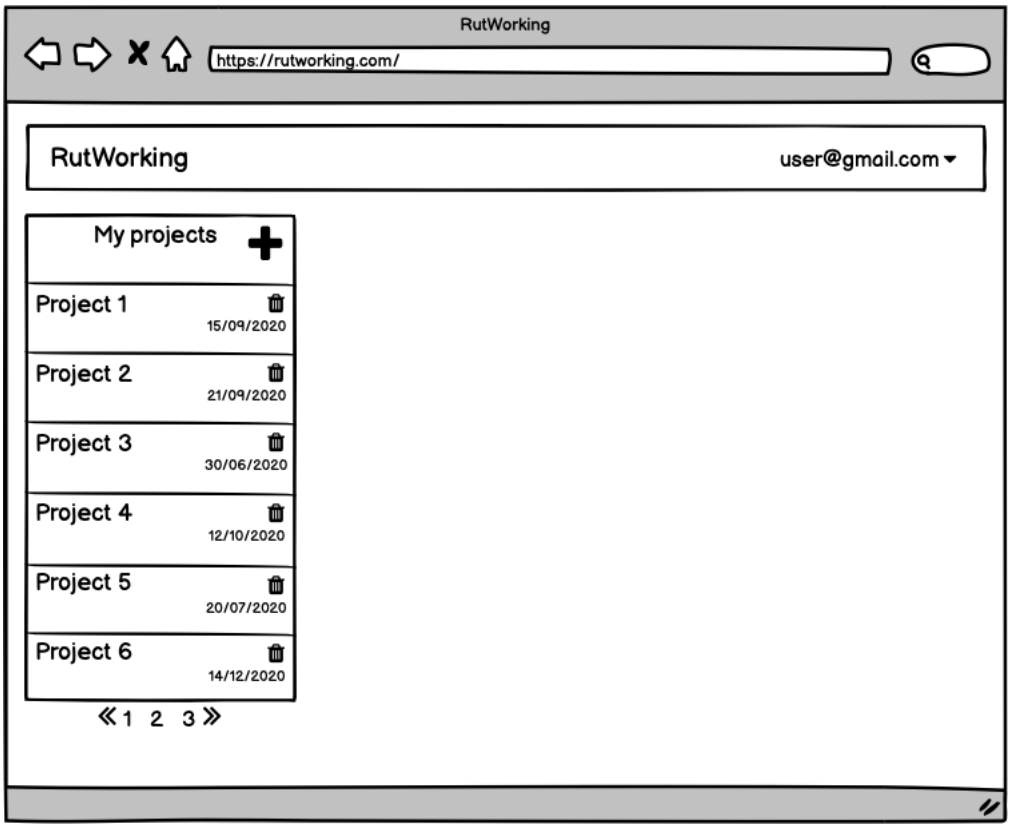

Figura 2: Home page (versione desktop)

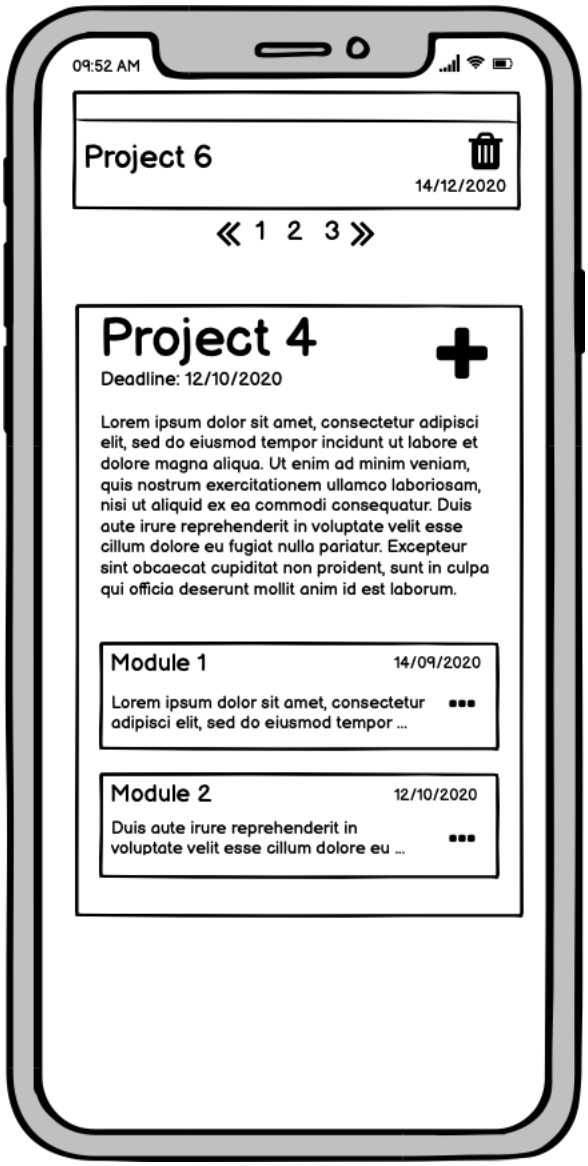

Figura 3: Dettaglio progetto (versione mobile). Informazioni relative al progetto selezionato con la lista di moduli di cui è costituito.

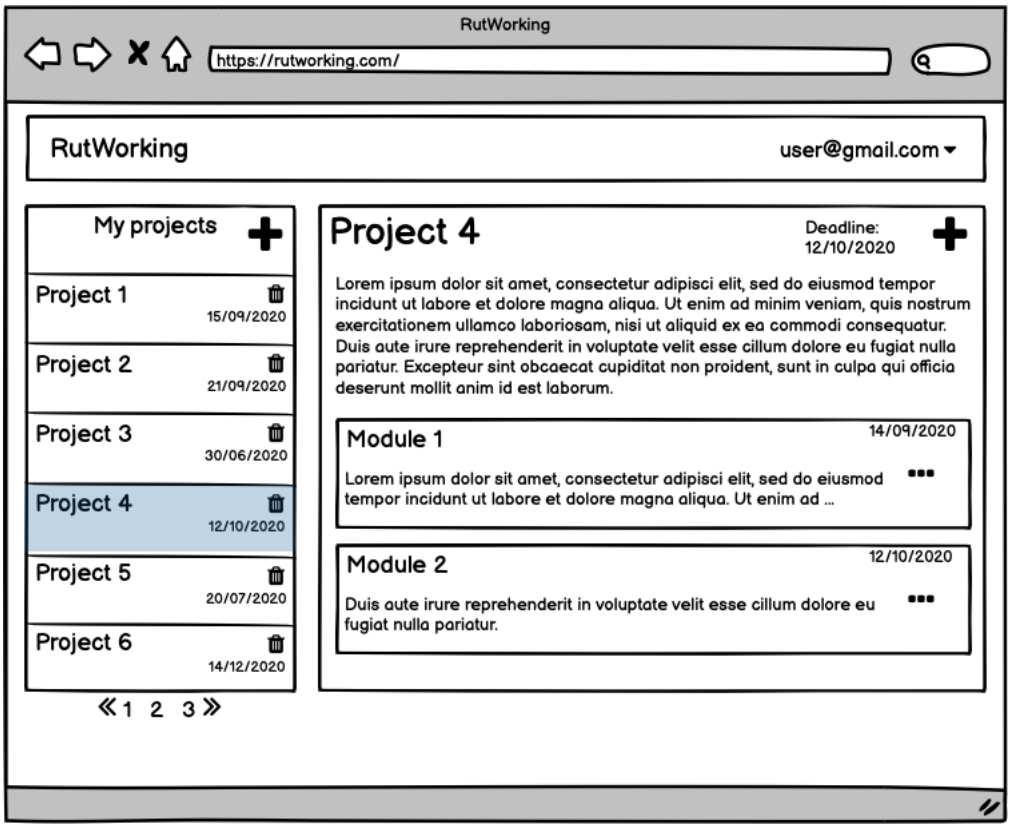

Figura 4: Dettaglio progetto (versione desktop)

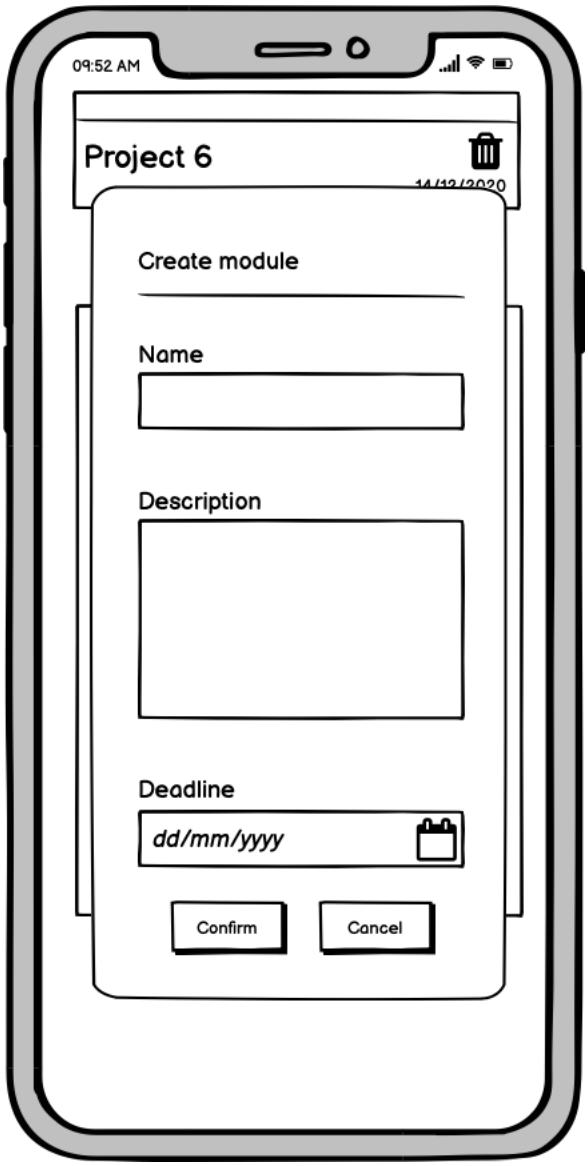

Figura 5: Aggiunta di un modulo al progetto (versione mobile)

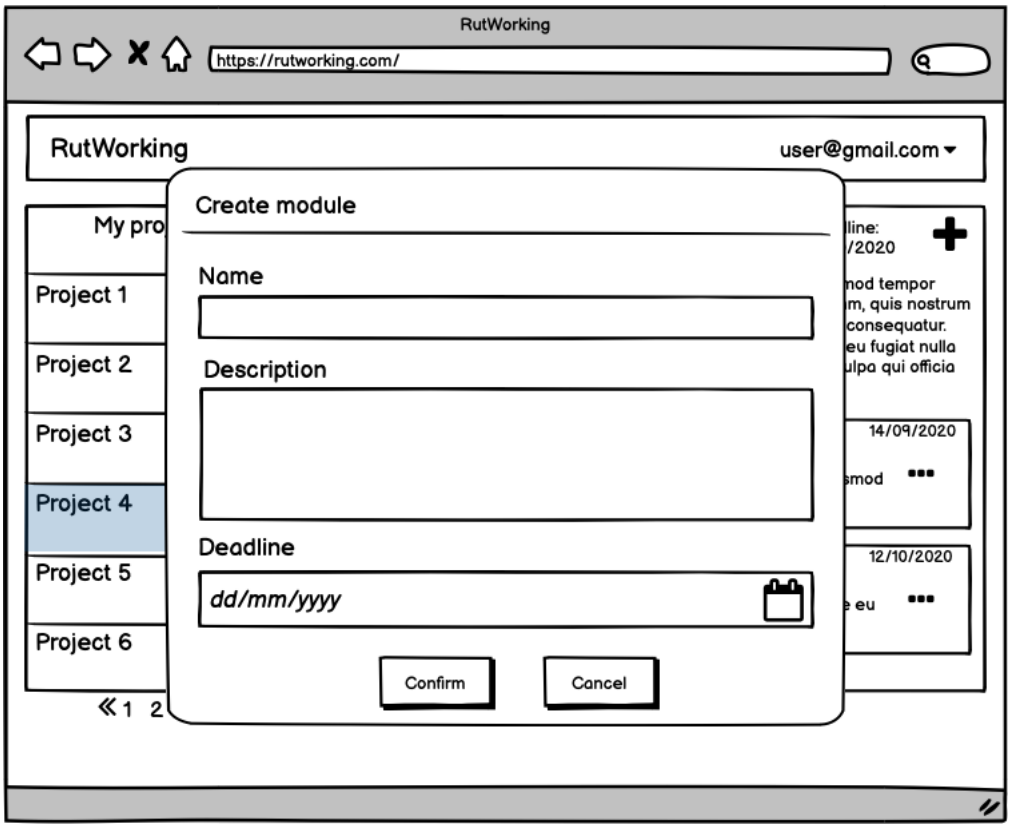

Figura 6: Aggiunta di un modulo al progetto (versione desktop)

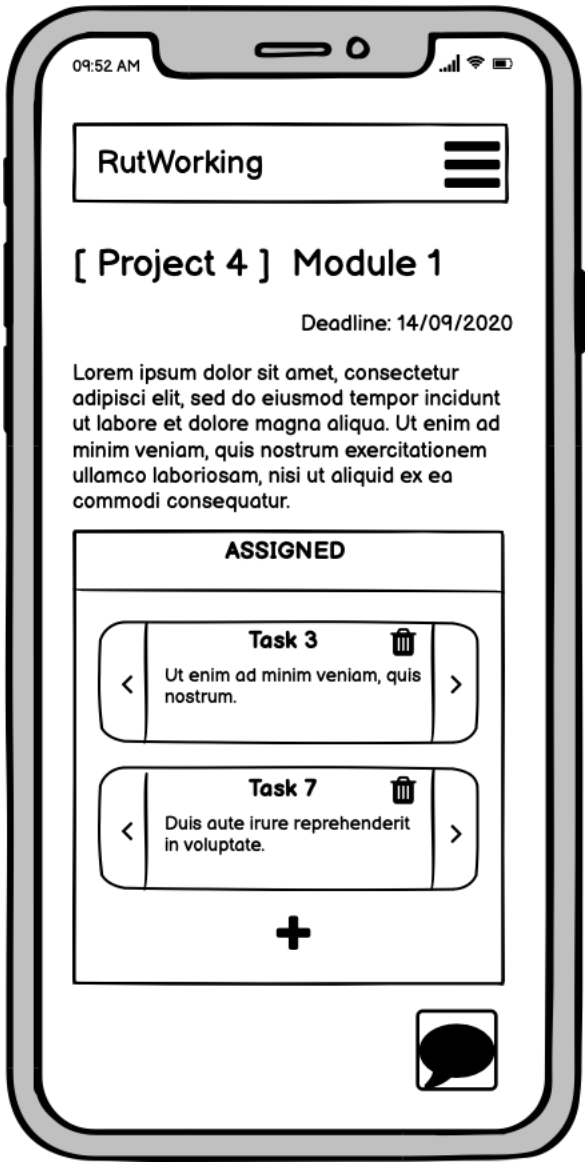

Figura 7: Working area (versione mobile). Informazioni inerenti al modulo e kanban.

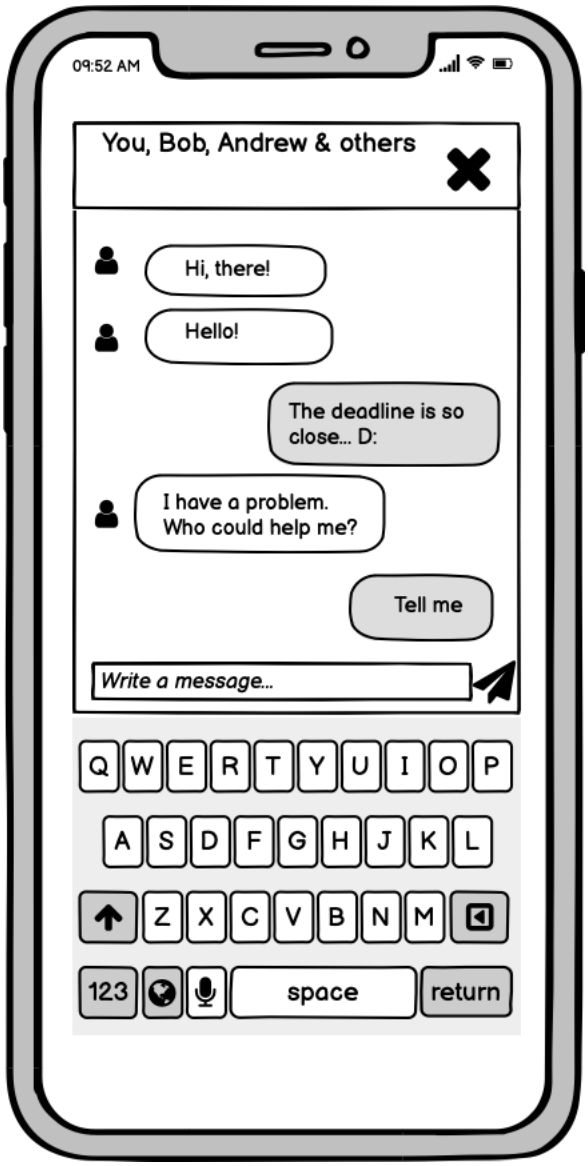

Figura 8: Chat di modulo (versione mobile)

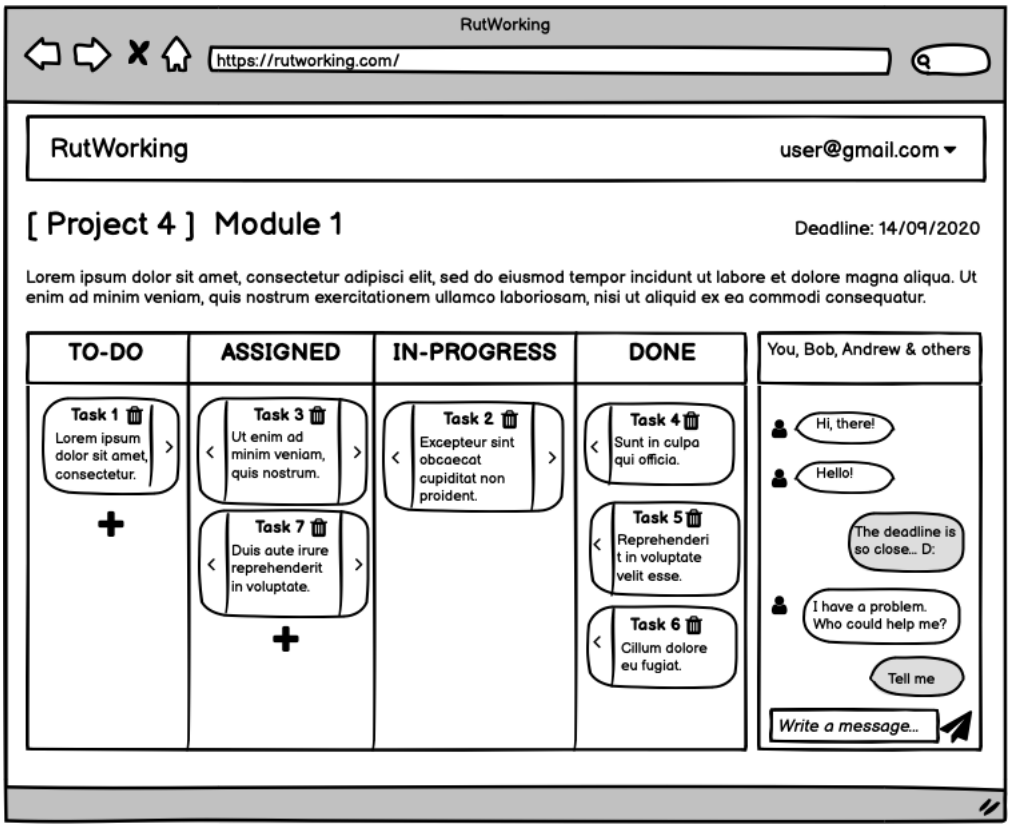

Figura 9: Working area (versione desktop). Informazioni inerenti al modulo, kanban e chat.

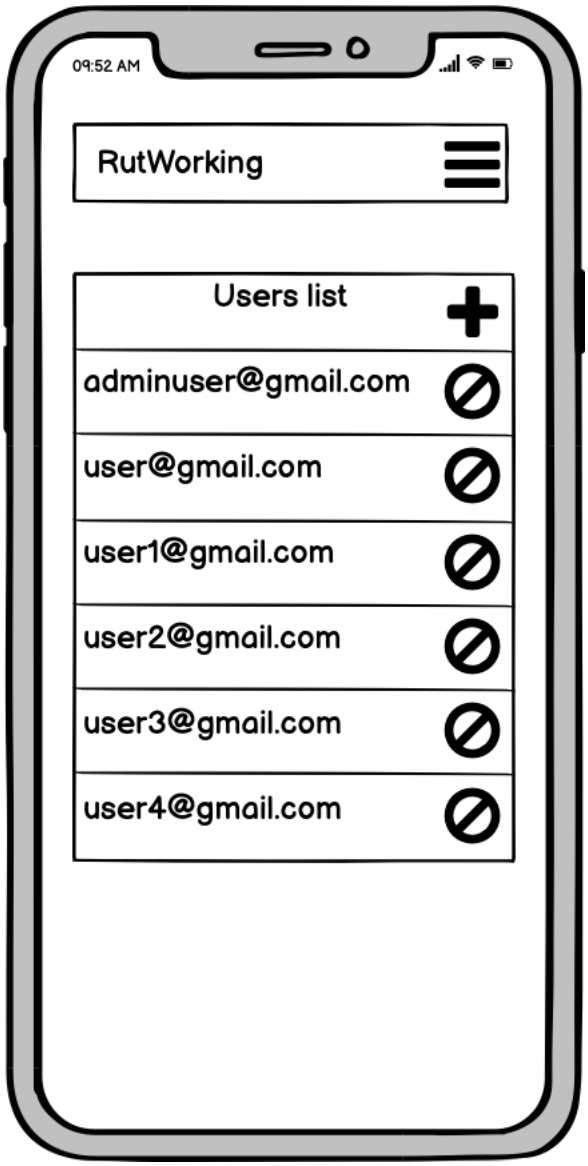

Figura 10: Admin page (versione mobile). Lista utenti registrati al servizio.

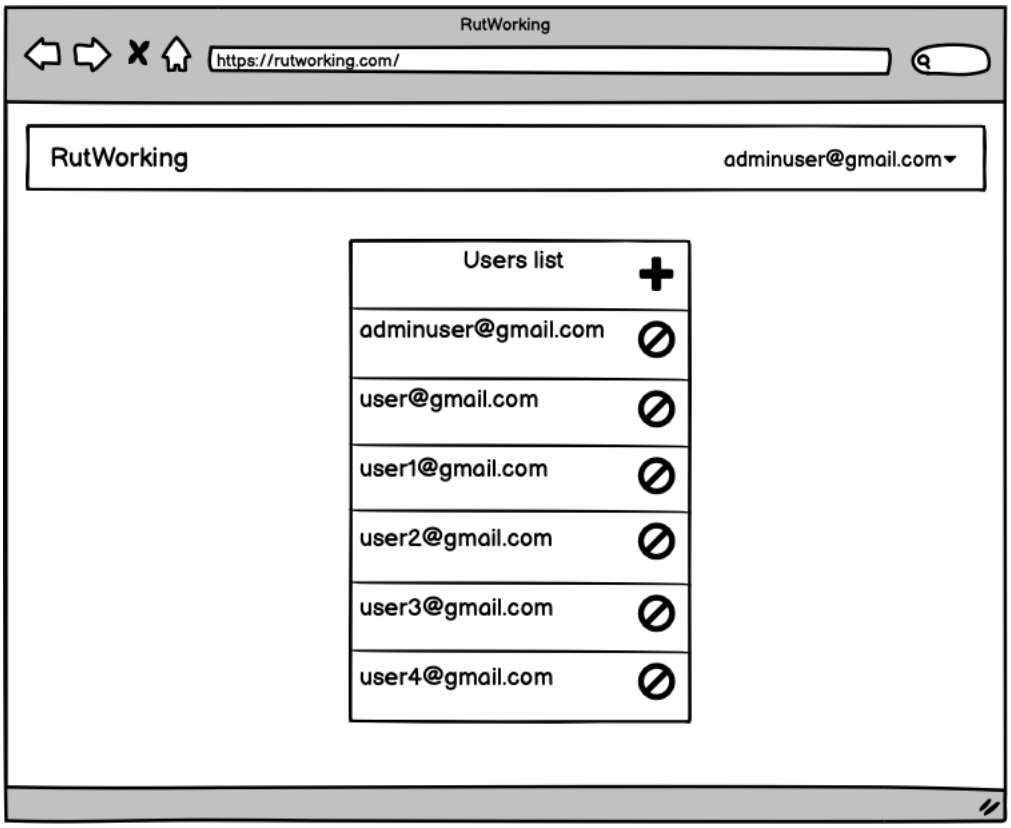

Figura 11: Admin page (versione desktop)

3.3 REST API

L'interazione client-server dell'applicativo si basa su chiamate HTTP. Queste sono effettuate su URL ben definiti e attinenti allo stile REST.
Una volta definiti i *mockup*, il passo successivo dello sviluppo dell'applicativo è stato identificare i tipi di dati necessari al suo funzionamento e, da questi, le API che era necessario creare. Il processo non è ovviamente risultato perfetto fin da subito ed ha richiesto successivi aggiustamenti sia per ottenere dati che non si era immediatamente pensato di rendere accessibili, sia per aggiungere filtri e parametri di utility alle varie chiamate.
A seguito del lavoro effettuato, riteniamo che il processo di design e, soprattutto, utilizzo delle API sia risultato molto più affrontabile grazie alla creazione di una documentazione pubblica delle stesse permessa dal servizio web Swagger. Questo servizio di hosting ha permesso di migliorare molto il processo di interazione fra i membri del team, riducendo i tempi e le comunicazioni necessarie.
La versione finale delle API è disponibile al seguente URL:
app.swaggerhub.com/apis/pb.mandrab/RutWorking/1.0.0.

4 Tecnologie

4.1 Server side

Node.js[15] è una piattaforma cross-platform event-driven per l'esecuzione di codice JavaScript. Permette di creare il proprio web server.
Nonostante fosse espressamente richiesto per lo sviluppo del progetto, la sua velocità e versatilità, in aggiunta alla grande quantità di pacchetti presenti, ci avrebbe comunque spinto ad un suo utilizzo. Inoltre, la sua caratteristica di scalabilità dovuta all'essere una piattaforma event-driven e la tipologia di progetto a cui ci stavamo approcciando (non CPU intensive), lo rendevano il candidato ideale.

Express[16] è lo standard de facto fra i framework per server-applications costruite su Node.js. È stato progettato per facilitare l'uso di Node e per creare API REST in maniera facile.

MongoDB[5] & Mongoose[17] è un database non relazionale basato sull'utilizzo di documenti JSON-like. Rappresenta in maniera molto semplice le informazioni, permettendo una 'gerarchia' delle stesse tramite nesting di documenti. Si è rivelato molto semplice ed intuitivo da utilizzare, grazie anche alla presenza di operatori auto-esplicativi e all'utilizzo di mongoose. Quest'ultimo aiuta nello specifico definendo degli *schemas* e *models*, che creano una struttura standard per un determinato documento, utile ad individuare alcuni tipi di errori.

Firebase Cloud Messaging[3] è una soluzione cross-platform che permette l'invio di messaggi/notifiche senza costi. Tenendo in considerazione anche l'utilizzo di web-socket, la scelta è ricaduta sull'affidarsi a questo servizio in quanto, seppur comunque funzionale all'obiettivo, rappresenta la scelta più utilizzata per sistemi mobile. Il nostro ragionamento ha infatti considerato la possibilità che una ditta opti per la creazione di un app nativa (in parallelo al client web) per la navigazione dei suoi dipendenti su dispositivi mobile. In questo ambito, la soluzione più comunemente adottata è quella dell'utilizzo di FCM.
Data quindi l'equipollenza di risultati (relativamente al contesto in esame) ed avendo già avuto modo di approcciarci all'utilizzo di web-socket in altri corsi, abbiamo deciso di sperimentare l'utilizzo di questo servizio esterno.

Mocha[4] è un framework di test per JavaScript che funziona su Node.js. Permette il testing di funzionalità asincrone in maniera relativamente semplice. Nel nostro progetto è stato utilizzato nel testing delle API risultando spesso veramente comodo nell'individuazione di errori a seguito di modifiche importanti del servizio. Questo framework è stato utilizzato in combinazione con il modulo *supertest[8].*

bcrypt[18] è un modulo per Node.js che permette un'agevole cifratura delle password. Per questa, utilizza un sale ed ha il vantaggio di salvare lo stesso e la password hashata come un'unica *string* composta da due parti, il che ne semplifica lo storing. Inoltre, questo modulo ha il vantaggio di essere una funzione adattiva che permette di aumentarne il tempo di calcolo semplicemente aumentando il numero di iterazioni effettuate dall'algoritmo. Questo modulo si è dunque considerato una valida scelta per la questione *security.*

nodemailer[6] è un modulo per Node.js che permette l'interfacciamento con client mail per l'invio di posta elettronica. Nel progetto è stato utilizzato in quanto necessario in due specifiche casistiche.
Abbiamo già detto come la registrazione di un utente sia effettuata all'assunzione e come la password sia inizialmente generata automaticamente. Quest'ultima non può evidentemente essere data all'amministratore sia per questioni di praticità che di sicurezza. Il trade-off è stato utilizzare l'invio di una mail contenente la password all'indirizzo di posta elettronica dell'utente. Questo potrà dunque utilizzarla per loggarsi e quindi, successivamente, cambiarla.
Il secondo punto è stato invece l'eliminazione di un utente (in realtà, nella pratica è un blocco). In questo caso l'email è inviata per informare l'utente di essere stato bloccato dal sistema.

OpenAPI[7] è una specifica creata per descrivere le REST API fornite da un servizio. È stata utilizzata per la documentazione delle API durante lo sviluppo. La specifica è stata utilizzata attraverso l'editor online Swagger[9].

4.2 Client side

Vue.js[14] è un framework JavaScript open-source in configurazione *Model–View–ViewModel* (MVVM) per la creazione di interfacce utente e *single-page applications*. I componenti Vue estendono gli elementi HTML di base per incapsulare il codice riutilizzabile. Vue presenta un sistema di reattività che utilizza semplici oggetti JavaScript e un re-rendering ottimizzato. Ogni componente tiene traccia delle sue dipendenze reattive durante il rendering, in modo che il sistema sappia esattamente quando eseguire nuovamente il rendering e su quali componenti. Rappresenta la colonna portante del *client-side* di questo progetto, nonché un elemento dello stak MEVN. Tra le opzioni possibili - React, Angular e Vue - la scelta è ricaduta su quest'ultimo in quanto ci è sembrato più intuitivo e modulare, considerata la sua natura a componenti. Proprio questi componenti e moduli installabili in stile plug-in nel progetto permettono di realizzare funzionalità avanzate richieste per le moderne single-page application come il routing.

Bootstrap[1] è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS per le varie componenti dell'interfaccia, come moduli, pulsanti e navigazione, così come alcune estensioni opzionali di JavaScript. Bootstrap ci ha permesso di organizzare e dare uno stile alla *user interface* e quindi ai componenti creati con Vue in maniera agile e veloce. Sopratutto grazie al *grid system* e alle proprietà come *display*. Bootstrap ha avuto un ruolo centrale nella realizzazione della *responsiveness* di questa SPA.

Vue-router[13] è il router ufficiale di Vue.js. Permette a Vue di creare single-page application. L'utilizzo di questo modulo è molto intuitivo e rende possibile la gestione e la personalizzazione anche delle pagine d'errore (*e.g.* 404).

Swiper[10] è un componente Vue presente nella vasta libreria di npm che permette di implementare un sistema di swiping principalmente pensato per utenti mobile touch screen. In questa applicazione è stato utilizzato per migliorare e semplificare la user experience e il layout per la parte mobile.

Vue-Font-Awesome[11] è un componente che si appoggia alla raccolta di icone di Font Awesome. Permette, grazie al suo componente, di importare, inserire e personalizzare icone nella propria applicazione.

Vue-resource[12] è un plugin per Vue.js che fornisce servizi per effettuare richieste web e gestire le risposte utilizzando XMLHttpRequest o JSONP. Il meccanismo delle *promises* è molto comodo e semplice da applicare.

5 Codice

5.1 Server side

Mongo queries

```
await DBProject.update(
  { _id: this.parentID },
  {
    $set: { "modules.$[module].kanbanItems.$[kanbanItem].status":
      States.TODO },
    $unset: { "modules.$[module].kanbanItems.$[kanbanItem].assignee":
      "" }
  }, {
    arrayFilters : [
      { "module._id" : this._id() },
      {
        "kanbanItem.status": { $ne: States.DONE },
        "kanbanItem.assignee": userID
      }
    ]
  }
)
```

Una query di aggiornamento che utilizza gli *arrayFilters*.

```
await DBUser.aggregate([
    { $match: { score: { $ne: null } } },
    { $sort: { score: -1 } },
    { $skip: skipN },
    { $limit: 100 }
])
```

Una semplice query utilizzante la *aggregate* per ottenere valori da più documenti.

Mongoose middleware

```
userSchema.pre('save', async function (next) {
    let user = <IDBUser>this
    let salt = await genSalt(SALT_WORK_FACTOR)
    let hashedPwd = await hash(user.password, salt)
    user.password = hashedPwd
    next()
})
```

Utilizzo del *middleware 'Pre'* di mongoose per l'hashing della password alla creazione.

Firebase Cloud Messaging

```
await _admin.messaging().sendMulticast({
    data: notification,
    tokens: [...new Set(tokens)]
})
```

Procedura di invio di notifiche tramite *firebase*.

5.2 Client side

Vue Router

```
Vue.use(VueRouter);

const routes = [{
    path: "/",
    name: "HomePage",
    component: HomePage,
    meta: { title: 'Home Page | RutWorking', }
  }, {
    path: "/login",
    name: "LoginPage",
    component: LoginPage,
    meta: { title: 'Login | RutWorning', }
  }, ..... {
    path: "/404",
    name: "PageNotFound",
    meta: { title: 'Page Not Found | RutWorking', },
    component: () => import(/* webpackChunkName: "register" */
      "../views/PageNotFound.vue") // lazy loading
  }, { path: '*', redirect: '/404' } // otherwise redirect to error page
];
const router = new VueRouter({mode: "history",
  base: process.env.BASE_URL, routes});

router.beforeEach((to, from, next) => {
  // redirect to login page if not logged in and trying to access a
  // restricted page
  const publicPages = ['/login'];
  const authRequired = !publicPages.includes(to.path);
  const loggedIn = localStorage.getItem('user');

  if (authRequired && !loggedIn) {
    return next('/login');
  }
  document.title = to.meta.title;
  next();
})
```

Caricamento pagine in modalità lazy loading e gestione errore 404.

Componente Vue

```
<template>
  <transition name="modal">
    <div class="modal-mask">
      <div class="modal-wrapper">
        <div class="modal-container">
          <div class="modal-header text-secondary">
            <slot name="header">{{ title }} </slot>
          </div>
          <div class="modal-body text-danger">
            <slot name="body">
              <strong>{{ message }}</strong>
              <br/>
              <div v-if="secondaryMessage" class="text-secondary">
                {{ secondaryMessage }}
              </div>
            </slot>
          </div>
          <div class="modal-footer">
            <slot name="footer">
              <button class="modal-default-button btn btn-primary"
                @click="$emit('closeModal')"> OK </button>
            </slot>
          </div>
        </div>
      </div>
    </div>
  </transition>
</template>

<script>
export default {
  data() {
    return {
      showModal: false
    }
  },
  props: {
    title: {type: String},
    message: {type: String},
    secondaryMessage: {type: String}
  }
};
</script>
```

Semplice esempio di componente Vue (modale).

Notifiche Firebase

```
messaging.onMessage(payload => {
  var notifications = localStorage.getItem('notifications');
  switch (payload.data.topic) {
    case "chat_message":
      var username = JSON.parse(localStorage.getItem('user')).email;
      if (payload.data.senderEmail != username) {
        notifications++;
        localStorage.removeItem('notifications');
        localStorage.setItem('notifications', notifications);
      }
      break;
    case "developer_added":
      notifications++;
      localStorage.removeItem('notifications');
      localStorage.setItem('notifications', notifications);
      break;
    case "task_completed":
      notifications++;
      localStorage.removeItem('notifications');
      localStorage.setItem('notifications', notifications);
      break;
  }
  this.notificationsNumber = notifications;
});
```

Listener per le notifiche provenienti da Firebase.

Vue Resource

```
this.$http.post(localStorage.getItem('path') +
    '/user/' + this.user.email, json, tokenjson
).then(function() {
    this.$emit('userAdded');
    this.closeForm();
  }, (err) => {
    console.log(err.body);
    this.showModal = true;
    this.title = 'Error';
    this.message = err.body;
    this.creating = false;
  });
}
```

Esempio di HTTP Request POST con Vue Resource.

6 Test

Durante il progetto si è cercato di applicare varie metodologie di testing sul sistema. Queste sono state effettuate grazie al supporto di familiari e amici che si sono offerti di effettuare una simulazione di utilizzo. Essendo stato necessario effettuare più sessioni con diversi utenti, si sono potute applicare due tecniche: *think aloud protocol* e la sua variante *co-discovery learning*.
Da questi test sono emerse varie problematiche e idee che sono state tenute in considerazione per il miglioramento del sito e dalla *user experience* in generale. Alcune di queste sono state accolte e implementate mentre per altre non si è riusciti (più che altro per questioni di tempo) a far molto. Di seguito descriveremo brevemente due di queste sessioni e alcune impressioni significative ottenute dai soggetti del test.

6.1 Think aloud protocol

Durante il test il soggetto ha evidenziato stupore quando, tentando di bloccare un utente, il sistema non chiedeva conferma dell'azione che, per scelte logiche, risulta invero una scelta irreversibile. Inoltre, il soggetto ha espresso perplessità riguardo al design dell'icona utilizzata per il pulsante adibito al blocco dell'utente, in quanto lo ha condotto erroneamente a pensare che gli utenti fossero già tutti bloccati. Queste osservazioni sono state accolte richiedendo la conferma per ogni azione irreversibile attuabile al sistema e modificando il layout dei pulsanti sopra citati.
L'utente ha espresso frustrazione quando, dovendo tornare alla schermata precedente per l'aggiunta di un membro del team al modulo, il sistema mostrava la finestra di default anzichè l'ultimo progetto aperto, costringendolo di fatto a ritrovare il progetto e il modulo in questione. Considerato quindi che la dinamica del sistema impone che per assegnare un compito a uno sviluppatore questo venga aggiunto precedentemente come membro del modulo, l'utente ha suggerito di rendere più esplicativa in anticipo questa precondizione utilizzando pulsanti singoli le cui icone suggerissero le azioni compibili sul modulo, anziché nasconderli all'interno del dropdown presente al momento del test. Tali osservazioni hanno quindi portato all'eliminazione del dropdown in favore dei pulsanti singoli, mentre per evitare di tornare indietro per aggiungere utenti al modulo si è scelto di predisporre un pulsante apposito anche all'interno della working area.

6.2 Co-discovery learning

I soggetti hanno affrontato lo svolgimento dei task aiutandosi a vicenda. Durante questa prova, i dubbi dell'uno e dell'altro sono stati annotati e presi in considerazione. Ad ogni modo, tutti i task, seppur con qualche complicazione, sono stati portati a termine dai soggetti.

Nello specifico, i dubbi più interessanti riscontrati sono stati i seguenti.
I soggetti trovano confusionario che cliccando sul box del modulo questo si apra nella successiva schermata mentre cliccando sulla descrizione dello stesso (facente anch'essa parte del box) questa transizione non avvenga. Ciò è causato dal fatto che la descrizione stessa è un elemento cliccabile della box.
I soggetti mostrano avversione alla mancata presenza di un pulsante di aggiunta utenti direttamente nella pagina del modulo. Questo renderebbe l'aggiunta di uno sviluppatore un'operazione più scomoda del previsto. I soggetti notano comunque come questa azione sia, probabilmente, temporalmente sporadica.
Gli utenti mostrano confusione nella versione mobile del kanban in quanto non ritengono intuitivo lo swipe orizzontale per visualizzare le varie colonne. Questi suggeriscono di utilizzare, anzichè le attuali frecce non cliccabili, dei pulsanti veri e propri o dei 'pallini' che indichino la posizione in una *sequenza di schermate orizzontali*. Quest'ultima alternativa è stata poi quella implementata.
I soggetti mostrano dubbi sulla persistenza nella visualizzazione di account bloccati nell'area dell'admin. Non esprimono contrarietà, ma pongono domande sul perché sia necessario che un amministratore continui a vedere account non più utilizzati.

7 Deployment

Il codice sorgente dell'applicazione è disponibile al seguente repository di GitHub: https://github.com/Mandrab/RutWorking
Clonando il progetto è possibile mettere in esecuzione sia la parte client che la parte server previo download dei pacchetti necessari al funzionamento degli stessi, mediante il package manager npm. Per avviare il client è necessario eseguire il comando `npm run serve`. Per il server, nel caso si utilizzi un sistema UNIX, è sufficiente il comando `npm run-script run`, mentre su sistemi Windows è necessario prima compilare i sorgenti *typescript* con `tsc` e poi procedere con `node server.js` (presente nel path `src/main`). Per questioni di riservatezza/security, si è deciso di non pubblicare sul repository alcuni file di configurazione, che sono comunque stati forniti al momento della consegna dell'elaborato.
Per agevolare il rilascio del servizio, si è deciso anche di utilizzare Docker[2]. Per problemi legati all'utilizzo di NGINX, nello specifico alla sua configurazione, non si è riusciti ad effettuare il rilascio della parte client come si sarebbe voluto. Per questo motivo sono state pacchettizzate solo la parte server e il database mongodb, creando per ciascuno di essi un apposito container. Attraverso la definizione di una rete virtuale, che permette altresì di non dover definire degli indirizzi IP ma di sfruttare direttamente i nomi dei container come se fossero dei nodi di host, è quindi possibile fare comunicare la parte client con il resto del sistema. Seguendo questa modalità di installazione, i passi necessari sono i seguenti:

1. scaricare lo zip contenente i sorgenti del client dal drive (quelli su GitHub non funzionerebbero in quanto hanno le chiamate HTTP dirette a `localhost` anzichè al nome del container).

2. predisporre i pacchetti necessari alla parte client, tramite il comando `npm install`

3. eseguire la parte client, con `npm run serve -- --port 8081`

4. creare la rete virtuale mediante la quale i container comunicheranno, usando `docker network create -d bridge interna`

5. estrarre i file dell'immagine del container creato per *mongodb* tramite `tar xvzf mongodb_saved.tgz`, caricare nel registro locale l'immagine con `docker load < mongodb_save.tar` e infine eseguirla con `docker run -itd --network interna -p 27017-27019:27017-27019 --name mongodb mongodb`

6. ripetere gli stessi comandi per il server, eseguendolo questa volta tramite `docker run -itd --rm --network interna --name server -p 8080:8080 server`

8 Conclusioni

Giunti al termine del progetto, possiamo dire di essere tutto sommato compiaciuti del risultato ottenuto. Seppur qualche imperfezione di design e/o di user experience possa esser rimasta, il risultato finale ha convinto sia noi sviluppatori che gli utenti coinvolti nel test.
La parte di design ci ha dato modo di capire quanto effettivamente i mockup e i test con utenti siano importanti per la creazione di una view comprensibile. Inoltre, la creazione delle API REST ci ha fatto comprendere quanto queste siano importanti per separare la parte di logica applicativa e persistenza dalla parte prevalentemente grafica e di presentazione del client.
Parlado di tecnologie invece, possiamo affermare che questo progetto ci ha permesso di approfondire le nostre conoscenze su di un ampio spettro di argomenti. Questi spaziano dai DB NOSQL alle tecnologie server-side e client-side per arrivare addirittura alle tecnologie create prevalentemente per i dispositivi mobile (firebase).
In sintesi possiamo dire che, per i risultati ottenuti e le conoscenze acquisite, siamo contenti del lavoro effettuato durante la realizzazione del progetto.

Riferimenti bibliografici

[1] Bootstrap. https://getbootstrap.com/.

[2] Docker. https://www.docker.com.

[3] Firebase cloud messaging. https://firebase.google.com/docs/cloud-messaging.

[4] Mocha. https://mochajs.org.

[5] Mongodb. http://www.mongodb.com.

[6] nodemailer. https://nodemailer.com.

[7] Openapi. https://www.openapis.org.

[8] supertest. https://github.com/visionmedia/supertest.

[9] Swagger. https://swagger.io.

[10] Swiper. https://www.npmjs.com/package/vue-awesome-swiper.

[11] Vue-font-awesome. https://fontawesome.com/how-to-use/on-the-web/using-with/vuejs.

[12] Vue-resource. https://www.npmjs.com/package/vue-resource.

[13] Vue-router. https://router.vuejs.org/.

[14] Vue.js. https://vuejs.org/.

[15] Ryan Dahl. Node.js. http://www.nodejs.org.

[16] TJ Holowaychuk. Express.js. http://www.expressjs.com.

[17] Sergey Lyubka. Mongoose. https://mongoosejs.com.

[18] David Mazières Niels Provos. bcrypt. https://en.wikipedia.org/wiki/Bcrypt.