Social-Network
Applicazioni e Servizi Web

Marco Fontana - 1125091 {marco.fontana17@studio.unibo.it}

February 2025

1 Introduzione

Questo progetto riguarda la creazione di un social-network che permette di aggiungere e scambiare messaggi tra amici, pubblicare e visualizzare post pubblicati da altri amici e scambiare messaggi effimeri in anonimato sotto ogni post pubblicato.
Essendo questo progetto stato utilizzato anche per altri esami, in questa relazione si prenderà in considerazione solamente la parte sviluppata appositamente per l'esame di applicazioni e servizi web.
Il codice sviluppato appositamente per questo esame è contenuto nella cartella 'frontend' e il codice in comune tra questo esame e gli altri è contenuto nel modulo 'content-service' che contiene uno dei microservizi dell'applicazione, gli altri moduli sono stati utilizzati per altri esami per cui non hanno rilevanza per questo esame.

2 Requisiti

Di seguito i requisiti rilevati in fase di analisi del progetto

2.1 Requisiti funzionali

Sono stati rilevati i seguenti requisiti funzionali:

Login

- Iscrizione al servizio
- Accesso al servizio una volta effettuata l'iscrizione

Amicizie

- Inoltrare richieste di amicizia ad altri utenti iscritti all'applicazione per poterli aggiungere ai propri amici

- Aggiungere altri utenti alla propria lista amici una volta ricevuta una richiesta di amicizia
- Scambiare messaggi effimeri con i propri amici
- Ricevere notifiche in tempo reale sulle richieste di amicizia ricevute
- Ricevere notifiche in tempo reale sulle richieste di amicizia inoltrate, nel momento in cui vengono accettate o rifiutate

Post

- Pubblicare, visualizzare ed eliminare i propri post. (I post devono essere di tipo testuale)
- Visualizzare post che vengono pubblicati da amici (feed) dopo aver accettato la richiesta di amicizia.
- Scambiare e visualizzare messaggi effimeri, in tempo reale ed in anonimato sotto ad ogni post tra tutti gli utenti che possono visualizzarlo

2.2 Requisiti non funzionali

Sono stati rilevati i seguenti requisiti non funzionali:

- **Usabilità**: il sistema deve essere seguire tutte regole che rendono l'applicazione chiara e immediata nell'utilizzo indipendentemente dal tipo di utente che intende usufruirne
- **Accessibilità**: il sistema deve seguire le buone pratiche per rendere semplice l'utilizzo dell'applicazione ad ogni tipo di utente che vuole usufruirne
- **Adattabilità**: il sistema deve potersi adattare ad ogni tipo di schermo e browser utilizzato
- **Scalabilità**: il sistema deve essere facilmente scalabile e manutenibile

3 Design

3.1 Personas e scenarios

Di seguito alcune personas e scenarios sviluppate per progettare l'applicazione in base ad aspetti di design rilevanti riguardo i target users.

- Anna ha 27 anni ed è una content creator e scrittrice freelance. Ha studiato Comunicazione e Media Digitali e lavora a progetti editoriali e creativi. Vive in una grande città con un coinquilino e ama condividere i suoi pensieri e testi con la sua community online. Per lei, i social network sono un modo per esprimersi e connettersi con persone che apprezzano la scrittura. Anna usa il social network per pubblicare brevi poesie, riflessioni e pensieri quotidiani. Segue altri scrittori e partecipa a discussioni creative.

- Giovanni ha 35 anni ed è un consulente di marketing digitale. Ha una laurea in Economia e un master in Digital Marketing. È sposato e ha un figlio piccolo. Per lui, il social network è uno strumento di lavoro: lo usa per condividere aggiornamenti professionali, interagire con il suo network e promuovere eventi e progetti. Giovanni pubblica consigli sul marketing, riflessioni sulle tendenze del settore e aggiornamenti su eventi a cui partecipa o che organizza.

- Marco ha 40 anni ed è un ingegnere informatico. Lavora in una grande azienda tecnologica e segue con interesse i temi legati all'innovazione, all'intelligenza artificiale e alla sostenibilità digitale. È sposato e ha due figli. Usa il social network principalmente per informarsi, leggere opinioni di esperti e partecipare a discussioni su argomenti tecnologici. Non è un utente che pubblica spesso, ma quando lo fa, vuole stimolare un dibattito costruttivo. È scettico sui contenuti virali e preferisce interazioni di qualità a un grande numero di follower.

Di seguito alcuni scenarios identificati:

- Anna vuole mostrare a tutti i suoi amici le poesie che ha scritto, così apre il sito, essendo questa la prima volta che utilizza l'applicazione, inserisce le proprie credenziali e si registra.
  A questo punto visualizza la home page del sito, essendo questa la prima volta che avvia l'applicazione non ha ancora nessun amico nella lista amici, preme il bottone delle amicizie e dal menu manda una richiesta di amicizia ad uno dei suoi amici.
  Quando Anna riceve la notifica che la sua richiesta di amicizia è stata accettata decide di pubblicare la sua prima poesia, in modo che il suo amico Marco possa vederla.
  Anna decide quindi di tornare alla home page, cliccando il bottone per tornare indietro e poi crea un nuovo post inserendo la sua prima poesia.
  Una volta pubblicata, Marco potrà immediatamente visualizzarla dalla propria home page.

- Giovanni accede al social network per informare il suo network di un webinar che sta organizzando. Dopo aver effettuato il login, apre la sezione per creare un nuovo post e scrive un breve testo con la data. Essendosi Giovanni accorto poco dopo aver creato il post che la data inserita è errata, decide di eliminare il post e creare un altro, preme il bottone per visualizzare la propria pagina del profilo ed elimina il post cliccando sul bottone sotto ad esso.
  Una volta eliminato, torna alla home page e crea un nuovo post con le date corrette.

- Marco, mentre beve il caffè, legge un articolo sulla regolamentazione dell'IA e decide di avviare una discussione sul social network. Dopo aver effettuato il login, scrive un post con una domanda provocatoria: "L'intelligenza artificiale dovrebbe essere regolamentata più rigidamente? Quali potrebbero

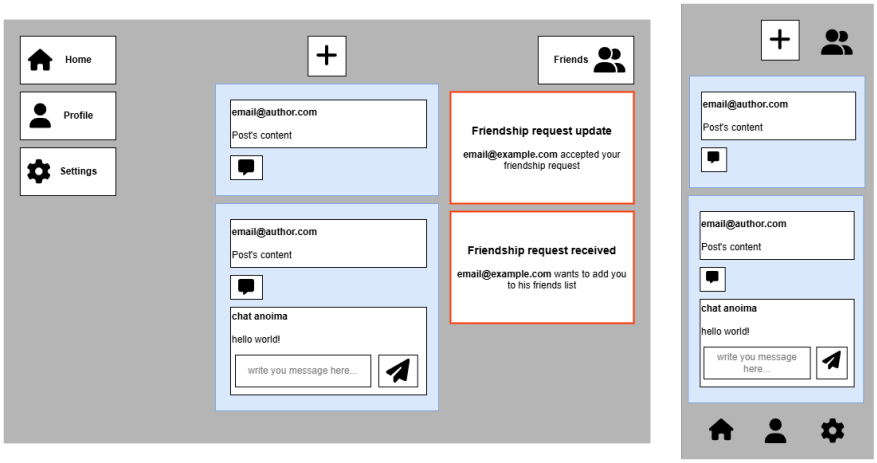

Figure 1: Mockup home

essere le conseguenze?". Dopo aver pubblicato il post, apre la chat sotto ad esso per leggere le opinioni degli utenti, ci sono opinioni contrastanti tra chi sostiene la regolamentazione e chi invece la considera un freno all'innovazione.

3.2 Architettura

L'architettura generale è distribuita, event-driven e a micro-servizi.
Per quanto riguarda solamente la parte per l'esame di ASW, è stato usato lo stack MEVN ed è stata sviluppata l'applicazione come single page application. Utilizza API REST, SSE e WebSocket a seconda del tipo di funzionalità.

3.3 Design delle intefacce

Riguardo al design delle interfacce, particolare attenzione è stata data alla accessibilità, inserendo colori di uso comune e descrizioni alternative alle immagini e icone che sono state utilizzate durante lo sviluppo.
Inoltre si è scelto un design 'mobile first', dando la priorità ai dispositivi mobile rispetto ai dispositivi a schermo orizzontale più grande.
Il layout è di tipo responsive e si adatta ad ogni dispositivo che sta utilizzando l'applicazione.

Durante la fase di design sono stati generati alcuni mockup, di seguito una parte di quelli che sono stati utilizzati (fig. 1, 4, 3, 2), su richiesta possono essere mostrati tutti i mockup prodotti.

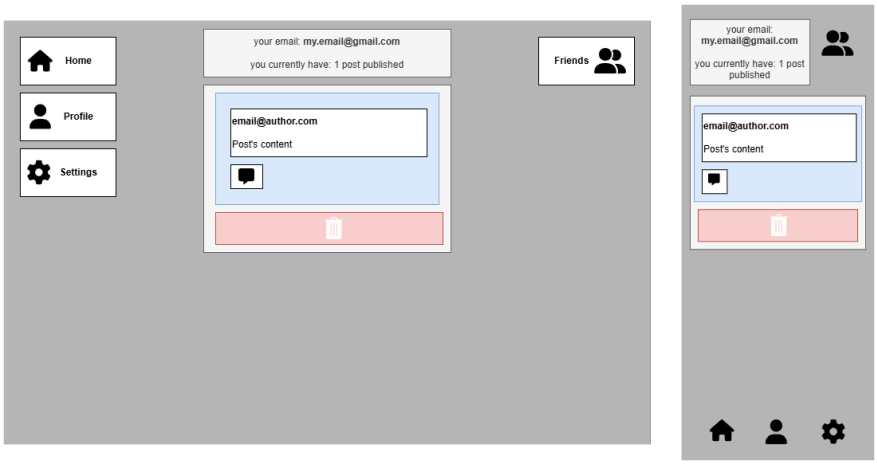

Figure 2: Mockup profile

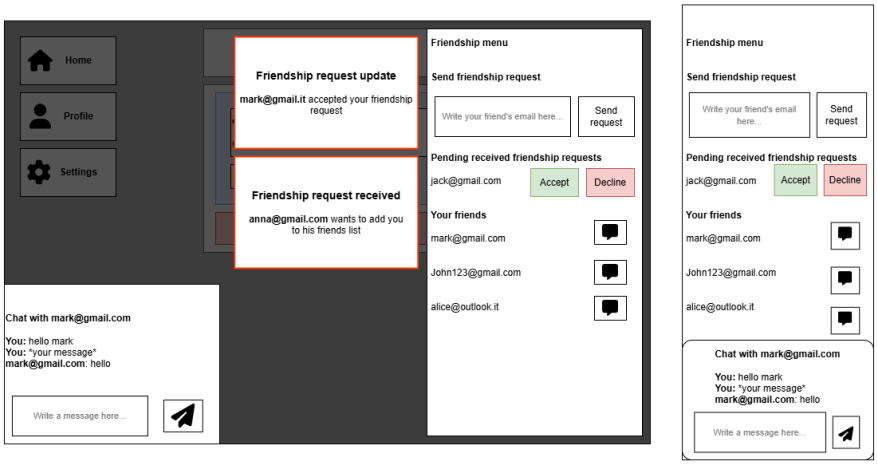

Figure 3: Mockup friendship menu

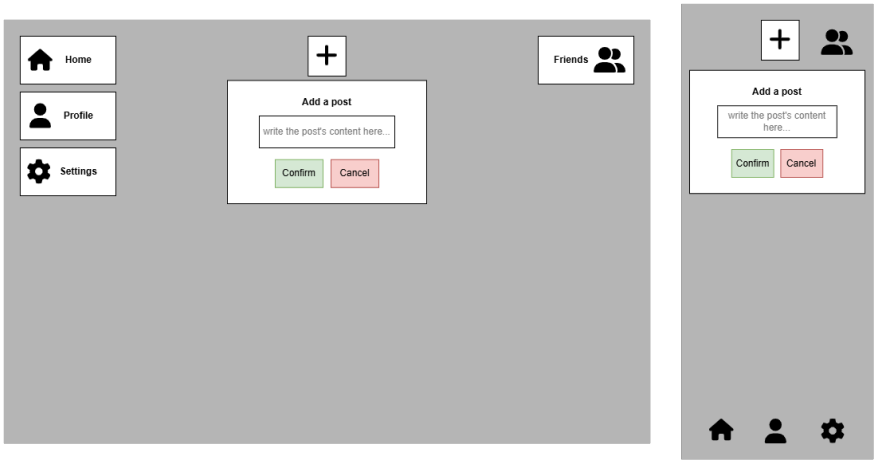

Figure 4: Mockup add post

4 Tecnologie

Di seguito le tecnologie utilizzate per quanto riguarda solamente la parte per l'esame di ASW:

- **Stack MEVN**: MongoDB, Express.js, Vue.js, Node.js, stack che permette di sviluppare applicazioni web full-stack con un approccio moderno
- **Typescript**: sia lato back-end che front-end per poter utilizzare Javascript con la tipizzazione
- **SCSS**: per la semplicità e flessibilità che garantisce rispetto a CSS
- **Mongoose**: per interagire con database Mongo con una API chiara e semplice
- **Socket.io**: per permettere comunicazioni bidirezionali a bassa latenza, utilizzata per le chat
- **Axios**: per effettuare richieste HTTP con una API chiara e semplice
- **Pinia**: per poter scambiare e gestire dati degli utenti che hanno acceduto al servizio
- **Jest**: per effettuare test del sistema
- **KafkaJS**: per scambiare eventi lato back-end tra i microservizi del sistema mediante event-broker kafka
- **Docker**: per effettuare il deploy del sistema

- **Gradle**: come strumento di build automation (utilizza un plugin per effettuare build di codice scritto in typescript lato back-end)
- **Vite**: come strumento di build automation lato front-end

5 Codice

Di seguito alcune porzioni di codice rilevanti che riguardano gli argomenti trattati nel corso.

5.1 Sse

Il progetto fa uso di SSE per notificare ai client alcuni tipi di eventi.
Di seguito un esempio di come vengono ricevuti e mostrati eventi riguardo ai post (dopo che due utenti stringono una amicizia, tutti i post che vengono pubblicati successivamente sono mostrati immediatamente a tutti gli amici nei rispettivi feed):

```
// controller.ts

/**
 * Handler for the SSE endpoint
 */
sseHandler = (req: Request, res: Response) => {
    // Set the necessary headers for SSE
    res.setHeader("Content-Type", "text/event-stream");
    res.setHeader("Cache-Control", "no-cache");
    res.setHeader("Connection", "keep-alive");
    res.flushHeaders();

    const postAddedListener = (post: Post) => {
        console.log("received postAdded event, about to send it to
            client: '{}'", post);
        res.write(`data: ${JSON.stringify(post)}\n\n`);
    };

    this.service.getPostAddedEmitter().on("postAdded",
        postAddedListener);

    req.on("close", () => {
        this.service.getPostAddedEmitter().removeListener("
            postAdded", postAddedListener);
    });
};
```

```
//Home.vue

eventSource = defineSseEventSource(email.value, 'localhost',
    '8082');

if (eventSource) eventSource.onmessage = (event: MessageEvent<any>)
     => {
console.log('Received event:', event.data);
```

```
try {
  const newPost: PostType = JSON.parse(event.data);
  console.log('New post:', newPost);

  if (newPost.author && newPost.author.email) {
    posts.value.push(newPost);
  } else {
    console.error('Post received without valid user information');
  }
} catch (error) {
  console.error('Error parsing SSE data:', error);
}
```

5.2 WebSocket

Il progetto fa uso di WebSocket per quanto riguarda le chat.
Di seguito un esempio di come è stata utilizzata una socket per permettere di scambiare messaggi nelle chat sotto ai post

```
// api.ts

/**
 * Start the server
 * @param onStarted callback to run when the server starts
 */
async start(onStarted: () => void = () => {}) {
    return new Promise<void>((resolve) => {
        const app = express();

        app.use(cors({
            origin: 'http://localhost:5173',
            methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
            allowedHeaders: ['Content-Type', 'Authorization']
        }));

        app.use(...this.middlewares);
        app.use("/", this.router);

        this.server = http.createServer(app);

        this.io = new SocketIOServer(this.server, {
            cors: {
                origin: 'http://localhost:5173',
                methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS
                    '],
            }
        });

        this.io.on('connection', (socket) => {
            console.log('A user connected');

            socket.on('joinRoom', (postId) => {
                socket.join(postId);
                console.log(`User joined room for post: ${postId}`)
                    ;
```

```
            });

            socket.on('sendMessage', (postId, messageData) => {
                console.log(`Received message for post ${postId}:`,
                     messageData);
                this.io?.to(postId).emit('newMessage', messageData)
                    ;
            });

            socket.on('disconnect', () => {
                console.log('A user disconnected');
            });
        });

        this.server.listen(this.port, "0.0.0.0", () => {
            console.log(`Server is running on port ${this.port}`);
            onStarted();
            resolve();
        });
    });
}
```

```
// PostChatDialog.vue

/**
 * Joins the chat room via WebSocket and listens for new messages
 */
onMounted(() => {
  socket = io('http://localhost:8082');

  console.log('Joining room:', props.id);
  socket.emit('joinRoom', props.id);

  socket.on('newMessage', (messageData: { content: string }) => {
    console.log("Received message '" + messageData.content + "'
        from socket on chat room '", props.id);
    messages.value.push(messageData);
  });
});

/**
 * Disconnects from the WebSocket when the dialog is closed
 */
onBeforeUnmount(() => {
  if (socket) {
    socket.disconnect();
    console.log('Disconnected from socket');
  }
});

/**
 * Send a message to the chat room via WebSocket
 */
const sendMessage = () => {
  if (!newMessage.value.trim()) return;

  const messageData = {
```

```
    content: newMessage.value,
  };

  console.log('Sending message:', messageData);

  socket.emit('sendMessage', props.id, messageData);

  newMessage.value = '';
};
```

5.3 SCSS

La funzione di @if ed @else di SCSS è stata utilizzata per diversi scopi tra cui quello di mostrare all'utente i testi di errore con un colore che si adatti in base al colore di sfondo, in modo da rendere sempre leggibile il testo indipendentemente dal colore dello sfondo sopra cui si trova.

```
@mixin error-text-style($bg-color) {
  @if $bg-color == red {
    color: invert($bg-color);
  } @else {
    color: mix(red, $bg-color, 90%);
  }
  white-space: pre-line;
}
```

5.4 Vue

Sono stati utilizzati eventi che vengono generati a partire da click su alcuni bottoni, ad esempio, all'aggiunta di un nuovo amico, accettando la sua richiesta di amicizia, viene aggiornata la lista di amici visualizzabile dal menu degli amici.

```
/**
 * Accept a friendship request
 * @param index - Index of the request to accept
 */
const acceptRequest = async (index: number) => {
  try {
    const request = friendRequests.value[index];

    await axios.put('http://localhost:8081/friends/requests/accept
        ', {
      from: request.senderId,
      to: email.value,
    });

    friendRequests.value.splice(index, 1);

    // emit event to the parent component to call fetchFriendships
        in order to update the UI
    emit('refreshFriendships');
  } catch (error) {
    console.error('Error accepting friendship request:', error);
```

```
  }
};
```

6 Test

Sono stati eseguiti diversi tipi di test, sia riguardo il corretto funzionamento del codice sviluppato mediante test automatici, sia riguardo l'usabilità dell'applicazione mediante experience prototyping, usability testing e la verifica del rispetto delle euristiche di Nielsen.

6.1 Euristiche di Nielsen

Uno dei membri del team di sviluppo, come valutatore esperto, ha certificato il rispetto delle euristiche di Nielsen.

6.2 Experience prototyping

Tre utenti target sono stati fatti interagire con un prototipo dell'applicazione facendo uso di mockups, fornendo a loro alcuni task da eseguire, in modo da ottenere informazioni riguardo la chiarezza e l'usabilità dell'applicazione.
Sono state messe in luce alcune criticità riguardo la creazione di nuovi post, il bottone per la generazione di essi era stato inserito troppo in basso e risultava difficile da rilevare.
Di conseguenza è stato spostato in cima, in modo da essere più semplice da identificare.

6.3 User testing

Tre utenti target sono stati fatti interagire con una versione beta dell'applicazione, a tutti sono stati forniti alcuni obiettivi.
Gli obiettivi proposti erano i seguenti:

- inoltrare una richiesta di amicizia
- creare un post inserendo un contenuto a piacere
- aprire una chat con un amico e inoltrare un messaggio a piacere
- eliminare un proprio post
- aprire la chat di un post e inoltrare un messaggio a piacere

A seguito della fase di testing è stato fornito a ciascuno dei partecipanti un questionario UEQ completo (26 domande) per ottenere una valutazione sulla user experience.
Dai risultati ottenuti è emerso che l'applicazione è estremamente semplice da usare e capire, ma non spicca per originalità e modernità nello stile (fig. 5, 6).

UEQ Scales (Mean and Variance)		
Attrattività	1,833	1,03
Apprendibilità	3,000	0,00
Efficienca	3,000	0,00
Controllabilità	2,500	0,19
Stimolazione	0,417	1,33
Originalità	0,167	5,40

Figure 5: Media e varianza tabella UEQ

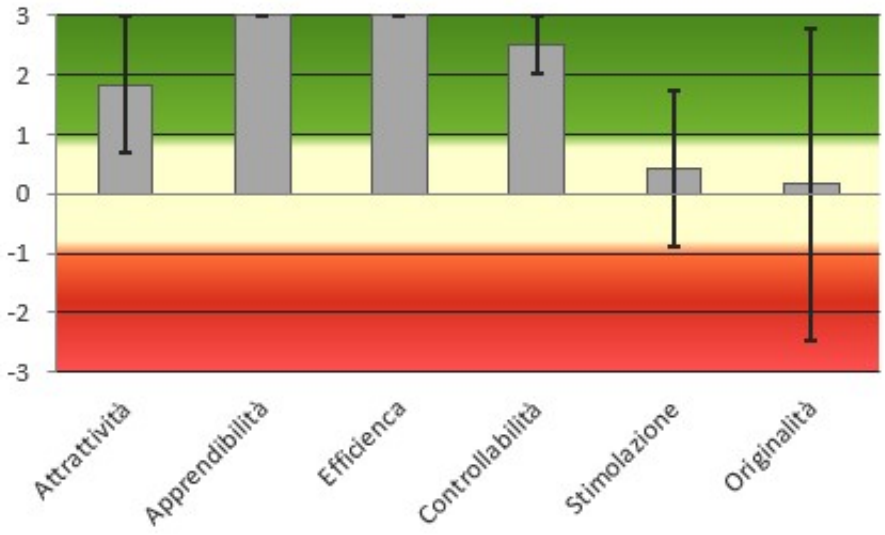

Figure 6: Media e varianza grafico UEQ

7 Deployment

Per effettuare il deploy del progetto è necessario avere Docker e Gradle installati sul sistema e seguire i seguenti passi:

- Aprire il terminale e posizionarsi nella cartella principale del progetto
- Creare due file 'db-root-password.txt' e 'db-password.txt' con dentro una password a piacere, ricordarsi di evitare eventuali spazi finali
- Eseguire il comando 'docker-compose up -d', in automatico verranno scaricare le immagini dei microservizi da docker-hub
- Posizionarsi nella cartella 'frontend'
- Eseguire il comando 'npm run dev' per lanciare l'applicazione
- Una volta terminato, eseguire il comando 'docker-compose down -v' per rimuovere i container e i dati salvati sui volumi

Se si intende effettuare delle modifiche al progetto e poi eseguirlo nuovamente è necessario eseguire anche il task gradle 'compileTypescript' nel modulo 'content-service'.

Si consiglia di accedere (opzionalmente) con l'utente 'test@gmail.com' in quanto vengono generati eventi di test (come richieste di amicizia e pubblicazione di post testuali) ogni 30 secondi che sono indirizzati a questo utente, in modo da simulare un uso realistico dell'applicazione. Questo profilo è stato utilizzato per effettuare test riguardo gli eventi che vengono ricevuti dagli utenti che utilizzano l'applicazione (la generazione automatica di questi eventi di test è gestita nella classe 'main.ts' nel modulo 'content-service').

8 Conclusioni

Il progetto è stato sviluppato secondo tutti requisiti rilevati in fase di analisi con una particolare attenzione all'accessibilità, usabilità e testing dell'applicazione. Essendo solamente un prototipo manca di diverse funzionalità come un login mediante password che lo renderebbe più sicuro, la possibilità di aggiungere amici specificando il nome utente anzichè l'email e l'aggiunta di altre funzionalità tipiche dei social-network (esempio: i 'mi piace' e i commenti sotto ai post), che sono possibili sviluppi futuri dell'applicazione.