

Rapporto del TEAM 3

Gruppo

Siamo il gruppo T3 Chess Cake formato dai membri:

- Petru Marcel Marincas - 0001021679 - Product Owner
- Davide Donati - 0001019487 - Scrum Master
- Giuseppe Forciniti - 0001019464 - Developer
- Saad Farqad Medhat - 0001027683 - Developer
- Rafid Fardeen Hoque - 0001027503 - Developer
- Vlad Alexandru Stefanuca - 0001027092 - Developer

Prodotto da costruire

Il prodotto ideato è un'applicazione web all'interno della quale si può giocare a scacchi non eterodossi, in particolare alle varianti: Really Bad Chess¹ e Kriegspiel².

Entrambe le varianti hanno la modalità "Giocatore contro giocatore online", mentre per Really Bad Chess abbiamo sviluppato anche altre modalità da giocare in locale oppure contro il computer.

L'applicazione contiene anche un profilo personale di ogni utente, dove poter visualizzare i propri progressi, e una classifica, dove potersi confrontare con altri utenti.

Vi è infine un'integrazione con i Social Media, in particolare Facebook, X e Reddit: è possibile condividere i propri traguardi o le proprie vittorie all'interno di questi social.

Scope e Backlog

Il prodotto è stato suddiviso in 7 epiche principali: Login e signup, Pagina del profilo e utente, Really Bad Chess, Navbar e interfaccia di tutto il sito, Leaderboard, Social Media integration e Kriegspiel.

In seguito, per ogni epica, trascriviamo quali sono le User Stories che abbiamo creato (non necessariamente in ordine di importanza / creazione, e con tra parentesi i punti relativi alla quantità di sforzo che avevamo supposto per ogni US)

Login e signup

Realizzazione di interfacce dedicate, metodi per gestire le richieste e gestione dei token di login

- Come guest, voglio poter entrare come anonimo (Punti: 2)
- Come guest, voglio registrarmi al sito con un nome utente e una password, per tenere traccia dei miei progressi ed accedere a più feature (Punti: 9)
- Come utente (registrato), voglio poter inserire un nome utente e una password, per poter accedere al mio profilo (Punti: 4)

¹ <http://www.reallybadchess.com/>

² [https://en.wikipedia.org/wiki/Kriegspiel_\(chess\)?useskin=vector](https://en.wikipedia.org/wiki/Kriegspiel_(chess)?useskin=vector)

- Come utente (registrato), voglio che la mia sessione di login sia salvata nel tempo, in modo da non dover eseguire continuamente il login (Punti: 5)

Pagina del profilo e utente

Comprendente dei dettagli di un utente registrato

- Come utente (registrato), voglio poter accedere all'area personale, dove posso vedere il mio profilo, ovvero il mio nome e i dati relativi alle partite che ho giocato (Punti: 2.5)
- Come utente (registrato), voglio vedere la lista delle partite fatte (Punti: 10)
- Come utente (registrato), voglio essere valutato da una funzione di elo, per tenere traccia dei miei progressi, mettermi a confronto con altri giocatori e risultare nelle leaderboard (Punti: 5)
- Come utente, voglio poter guardare il replay di una delle partite che ho fatto di Really Bad Chess all'interno dell'elenco partite sul mio profilo o sul profilo di altri (Punti: 9)

Really Bad Chess

Creazione di interfaccia, backend e frontend per il gioco di Really Bad Chess

- Come utente, voglio vedere la chessboard, i suoi pezzi e che io ci possa interagire (Punti: 6)
- Come utente, voglio poter generare una chessboard in base a un valore che identifica il rank, in modo da avere una partita sbilanciata (così come succede su Really Bad Chess) (Punti: 4)
- Come utente, voglio scegliere nell'interfaccia se giocare:
 - o Player vs Computer (Ranked)
 - o Daily Challenge
 - o Player vs Player (locale)
 - o Player vs Player (online)
 - o Freeplay

Inoltre voglio poter scegliere il tempo se sto giocando contro un giocatore, il rank se gioco in Freeplay o in giocatore contro giocatore locale, e, in quest'ultimo caso, anche i nomi dei due utenti. (Punti: 3.5)

- Come guest o utente, voglio poter giocare in locale contro un altro utente (Punti: 3)
- Come utente, voglio che le mie partite di Really Bad Chess vengano salvate sul mio profilo utente (Punti: 4)
- Come utente, voglio poter giocare contro un computer che è molto facile da battere ma che ha anche lui pezzi a caso e con un vantaggio dato dal rank (Punti: 7)
- Come utente, voglio poter giocare alla daily challenge contro il computer e che il numero di mosse per fare scacco venga salvato (Punti: 6)
- Come utente, voglio poter creare lobby per giocare contro altri giocatori, e voglio vedere le lobby disponibili e sfidare i giocatori (Punti: 19)
- Come utente, voglio che io e il mio avversario abbiamo un timer che mostra il tempo a disposizione e che, se finisce il tempo a disposizione ad uno dei giocatori lui perda in automatico la partita (Punti: 9)
- Come utente, voglio che i risultati che esistono nella leaderboard non siano sbagliati e che nel giocare le partite vi siano dei controlli sull'esecuzione delle mosse, su chi sta eseguendo le mosse, etc. (Voglio che ci sia uno strato di protezione e certificazione nel backend sul giocare) (Punti: 15)

Navbar e interfaccia di tutto il sito

Strumento intuitivo per navigare da una pagina all'altra

- Come guest o utente voglio poter navigare tra le varie interfacce, ovvero leaderboard, registrazione / login, pagina del profilo, gioco ed eventualmente altre (Punti: 5)

Leaderboard

Visualizzazione di classifiche

- Come utente o guest, voglio vedere le leaderboard di chi ha battuto la daily challenge di Really Bad Chess nel minor numero di mosse (Punti: 8)
- Come utente o guest, voglio vedere i nomi dei giocatori che hanno vinto con il rank più alto in Really Bad Chess (Punti: 4.5)
- Come utente o guest, voglio poter vedere gli elo più alti in Really Bad Chess (Punti: 4)

Social Media integration

Integrazione di funzionalità legate ai Social Media

- Come utente o guest, voglio poter condividere i risultati delle mie partite sui social network attraverso un post (Punti: 6)
- Come utente, voglio poter condividere la mia posizione nelle leaderboards sui social media (Punti: 6)
- Come utente, voglio poter creare un link di invito ad una partita da postare sui social media, in modo che altri giocatori possano entrare nella mia lobby e giocare contro di me (Punti: 4)

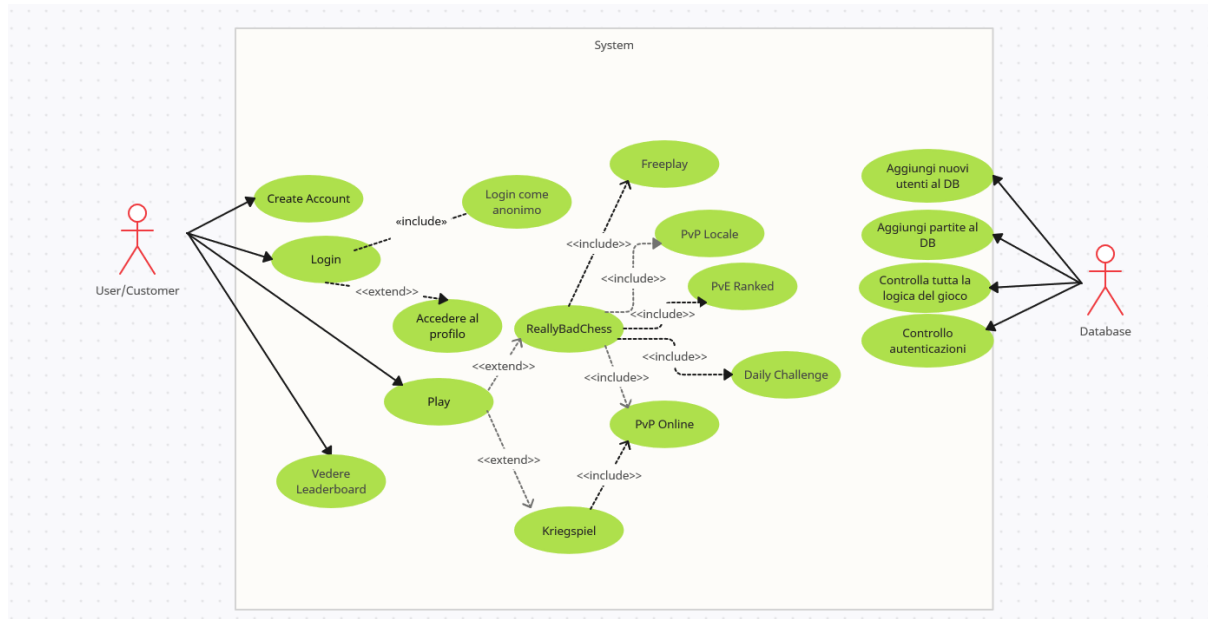
Kriegspiel

Creazione di interfaccia, backend e frontend per il gioco di Kriegspiel

- Come utente, voglio avere un'interfaccia dove posso vedere l'elenco di lobby nelle quali io posso entrare (Punti: 2.5)
- Come utente, voglio poter ottenere l'elenco di partite nelle quali posso entrare facendo una richiesta al server (Punti: 2)
- Come utente, voglio poter creare una nuova partita online, specificando la durata, e che questa partita venga aggiunta all'elenco di partite "aperte" (dove si può entrare) (Punti: 3)
- Come utente, voglio poter entrare in una partita di Kriegspiel contro un altro utente (Punti: 2)
- Come utente, voglio poter ottenere i dettagli della partita della quale sono dentro (Punti: 3)
- Come utente, voglio poter muovere i pezzi nella scacchiera e far sì che il cambiamento si veda anche per il giocatore avversario (Punti: 3)
- Come utente, voglio vedere quello che l'umpire mi comunica (Punti: 6.5)
- Come utente o guest, voglio poter vedere una leaderboard degli ELO più alti su Kriegspiel (Punti: 4.5)
- Come utente, voglio che siano visibili nell'elenco di partite anche le partite di Kriegspiel e che ci sia il mio ELO e winrate su Kriegspiel (Punti: 4.5)

Casi d'uso

Il diagramma di caso d'uso è dedicato alla descrizione delle funzioni o servizi offerti dal sistema, in questo caso il gioco di scacchi, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso, quindi l'utente e il database.



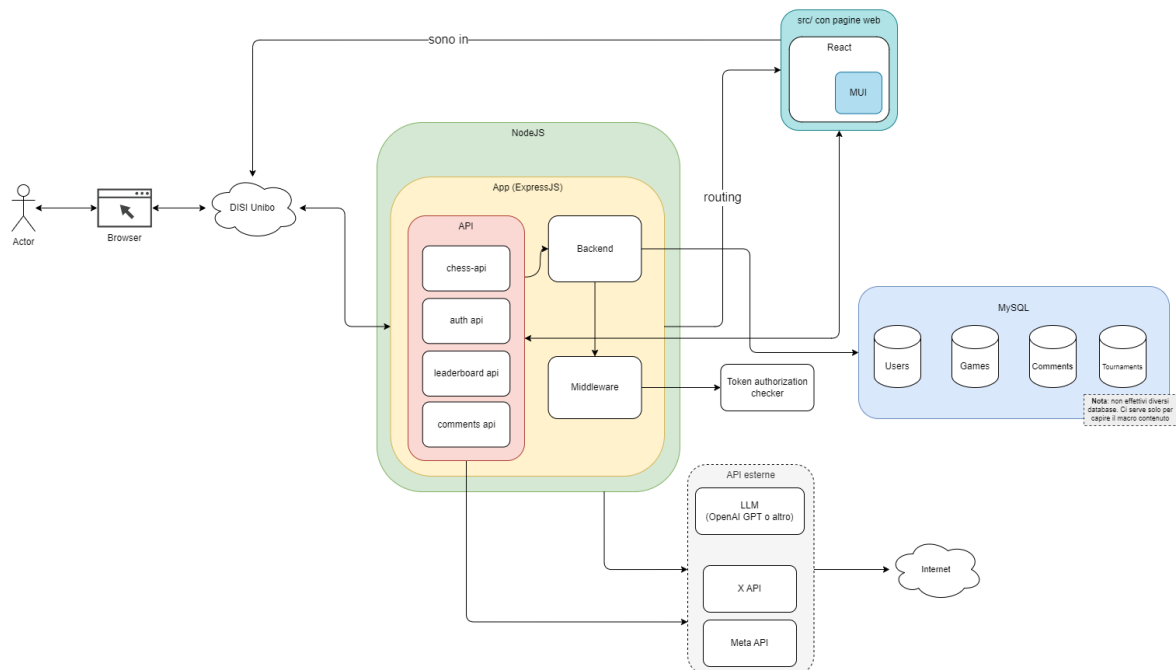
Architettura

L'architettura iniziale era stata decisa in base all'esperienza dei singoli membri del team di sviluppo: sono state scelte, tramite votazione democratica, architetture familiari.

Il server abbiamo deciso di strutturarli utilizzando NodeJs, in particolare utilizzando la libreria ExpressJs pensata utile per lo sviluppo di RESTful APIs.

Come prima idea di un database eravamo in accordo sull'utilizzo di MySQL, in seguito questa scelta ha portato a problemi, infatti il team si è trovato obbligato a cambiare rotta.

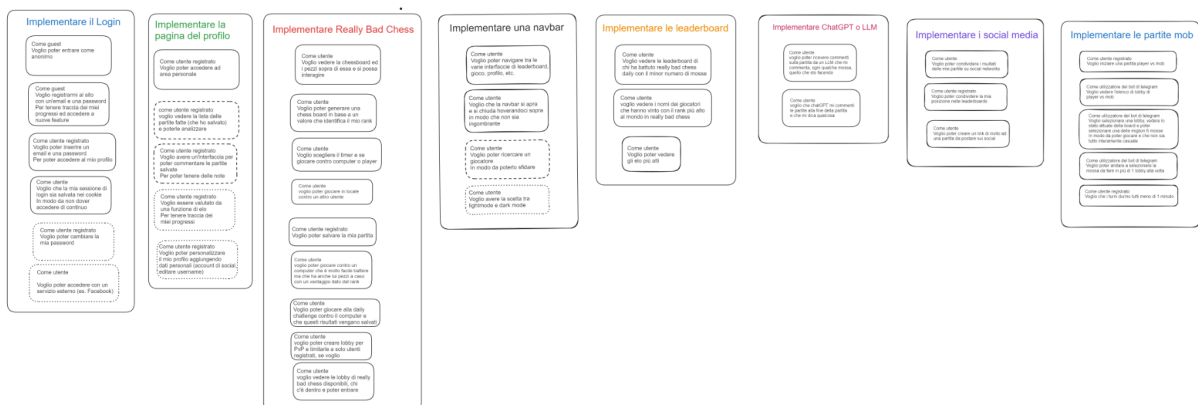
L'applicazione è pensata per essere renderizzata con l'aiuto di React, dove in particolare lo stile dei componenti è stato dato utilizzando MUI per mantenere, facilmente, uno standard per tutta la pagina web.



Sprint 0

Lo sprint iniziale prevedeva unicamente la creazione di un backlog e di consolidare il team building. Il team building viene descritto meglio nella sezione successiva.

Questo era come si presentava il backlog iniziale, nato dalle idee combinate del PO e del team di sviluppo:



La retrospettiva di questo sprint è stata fatta direttamente con i professori, quindi non c'è documentazione riguardante essa.

Sprint 1

Sprint Goals

Gli obiettivi posti dal team per il primo sprint lavorativo erano:

1. Implementazione di Really Bad Chess:
 1. Gestione della scacchiera.
 2. Generazione casuale delle partite basata sul ranking dei giocatori.
 3. Creazione di una "daily challenge" con generazione casuale.
2. Sistema di Accesso e Registrazione:
 1. Pagina di login e registrazione.
 2. Opzione per accedere come utente anonimo, con limitazioni funzionali (es. impossibilità di salvare partite).
 3. Salvataggio di username, password e token di sessione nel database.
3. Gestione della Sessione di Login:
 1. Mantenimento e gestione della sessione utente durante la navigazione nell'app.
4. Interfaccia Utente e Navigazione:
 1. Implementazione di una barra di navigazione verticale per spostarsi tra le diverse sezioni dell'applicazione.

Sprint Backlog

Le User Stories relative agli Sprint Goals ideati erano:

Implementazione di Really Bad Chess:

- Come utente, voglio vedere la chessboard, i suoi pezzi e che io ci possa interagire (Punti: 6)
 - Come utente, voglio poter generare una chessboard in base a un valore che identifica il rank, in modo da avere una partita sbilanciata (così come succede su Really Bad Chess) (Punti: 4)
 - Come utente, voglio scegliere nell'interfaccia se giocare:
 - o Player vs Computer (Ranked)
 - o Daily Challenge
 - o Player vs Player (locale)
 - o Player vs Player (online)
 - o Freeplay
- Inoltre voglio poter scegliere il tempo se sto giocando contro un giocatore, il rank se gioco in Freeplay o in giocatore contro giocatore locale, e, in quest'ultimo caso, anche i nomi dei due utenti. (Punti: 3.5)
- Come guest o utente, voglio poter giocare in locale contro un altro utente (Punti: 3)

- Come utente, voglio che le mie partite di Really Bad Chess vengano salvate sul mio profilo utente (Punti: 4)
- Come utente, voglio poter giocare contro un computer che è molto facile da battere ma che ha anche lui pezzi a caso e con un vantaggio dato dal rank (Punti: 7)

Sistema di Accesso e registrazione:

- Come guest, voglio poter entrare come anonimo (Punti: 2)
- Come guest, voglio registrarmi al sito con un nome utente e una password, per tenere traccia dei miei progressi ed accedere a più feature (Punti: 9)
- Come utente (registrato), voglio poter inserire un nome utente e una password, per poter accedere al mio profilo (Punti: 4)

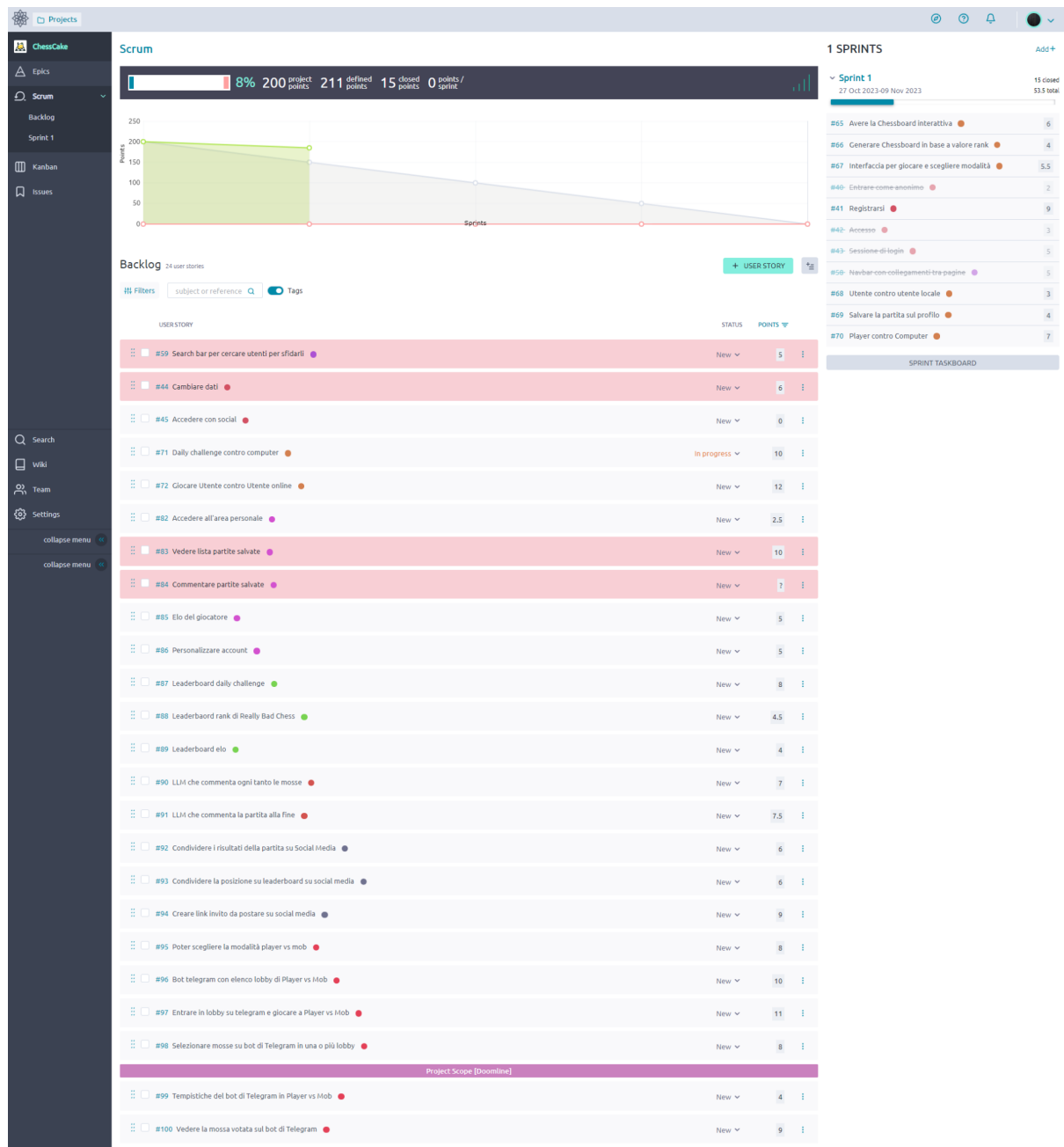
Gestione della Sessione di Login:

- Come utente (registrato), voglio che la mia sessione di login sia salvata nel tempo, in modo da non dover eseguire continuamente il login (Punti: 5)

Interfaccia Utente e Navigazione:

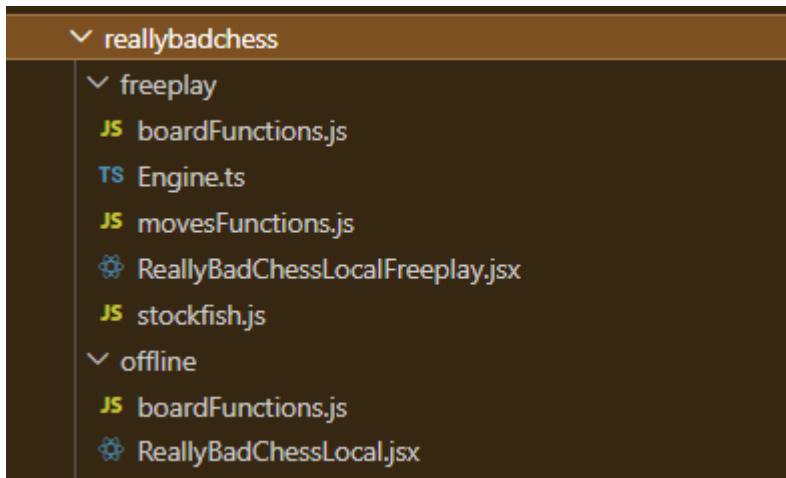
- Come guest o utente voglio poter navigare tra le varie interfacce, ovvero leaderboard, registrazione / login, pagina del profilo, gioco ed eventualmente altre (Punti: 5)

Con rispettiva immagine del backlog:



Descrizione del codice

Di seguito mostro le cartelle relative a Really Bad Chess, suddivise in due modalità: Freeplay, che mette l'utente contro il computer, con però la possibilità di decidere un ranking, e Offline, dove l'utente può giocare contro amici in locale, sulla stessa macchina.



In particolare mostro una delle prime funzioni create dal team di sviluppo, ossia la generazione della Board:

```
function generateBoard(mode, rank) {
  const newChess = new Chess();

  newChess.clear();

  // Caricamento pezzi
  let seed = 0;
  if (mode === 'dailyChallenge') {
    const today = new Date();
    seed = ((((((today.getFullYear() * 100 + today.getMonth() * 10 + today.getDate()) * 214013 + 2531011) >> 16) & 0x7fff) * 214013 + 2531011) >> 16) & 0x7fff);
  }

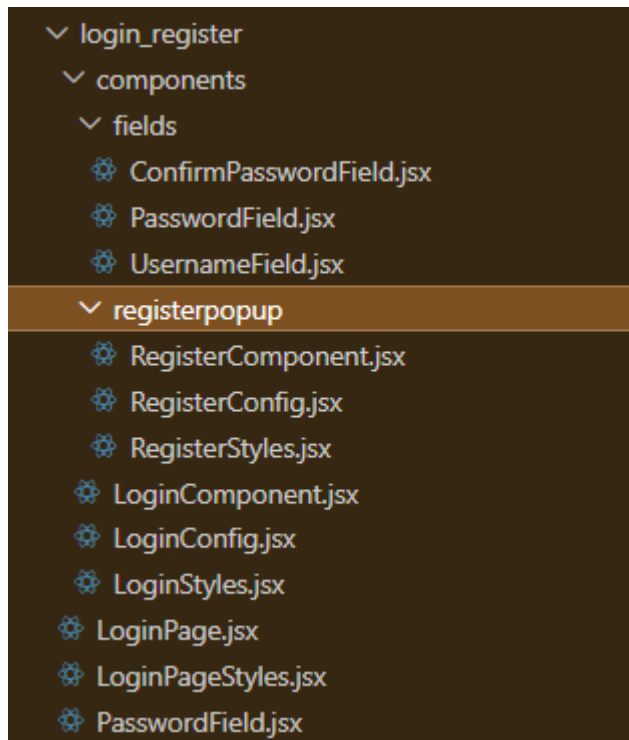
  const [playerRank, opponentRank] = calculateRanks(rank, seed);
  const whitePieces = findChessPiecesWithRank(playerRank, seed).sort((a, b) => a.value - b.value);
  const blackPieces = findChessPiecesWithRank(opponentRank, seed).sort((a, b) => a.value - b.value);

  const whiteSquares = ['a1', 'b1', 'c1', 'd1', 'f1', 'g1', 'h1', 'a2', 'b2', 'c2', 'd2', 'e2', 'f2', 'g2', 'h2'];
  const blackSquares = ['a8', 'b8', 'c8', 'd8', 'f8', 'g8', 'h8', 'a7', 'b7', 'c7', 'd7', 'e7', 'f7', 'g7', 'h7'];

  // Metti i pezzi disponibili in modo casuale nella fila 2
  whitePieces.filter(piece => piece.name === "q").forEach((piece) => {
    const validSquares = whiteSquares.filter(square => square[1] === "1");
    const randomIndex = Math.floor(Math.random() * validSquares.length);
    const square = validSquares.splice(randomIndex, 1)[0];
    newChess.put({ type: piece.name, color: 'w' }, square);
    whitePieces.splice(whitePieces.findIndex(p => p === piece), 1);
    whiteSquares.splice(whiteSquares.indexOf(square), 1);
  });

  whitePieces.filter(piece => piece.name === "p").forEach((piece) => {
    const validSquares = whiteSquares.filter(square => square[1] === "2");
    if (validSquares.length > 0) {
      const randomIndex = Math.floor(Math.random() * validSquares.length);
      const square = validSquares.splice(randomIndex, 1)[0];
      newChess.put({ type: piece.name, color: 'w' }, square);
      whitePieces.splice(whitePieces.findIndex(p => p === piece), 1);
      whiteSquares.splice(whiteSquares.indexOf(square), 1);
    }
  });
}
```

Questo è il frontend relativo al login e alla registrazione, comprendente di singoli componenti per i TextField e delle Paper relative.



In particolare questo è il componente relativo alla pagina di Login

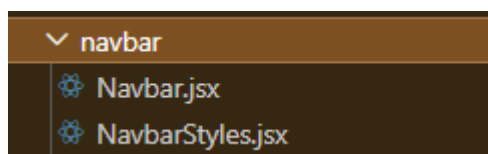
```
export default function LoginPage() {
  const { loginStatus, isTokenLoading } = useTokenChecker();
  const navigate = useNavigate();

  if (isTokenLoading) {
    return (
      <Box display="flex" justifyContent="center" alignItems="center" minHeight="100vh">
        <CircularProgress />
      </Box>
    );
  }

  if (loginStatus) {
    navigate("/play");
  }

  return (
    <Container
      maxWidth = 'false'
      sx = {styles.container}
    >
      <Grid
        container
        sx={styles.grid}
      >
        <Grid
          item
          xs={6}
          sx={styles.imageGrid}
        >
        </Grid>
        <Grid
          item
          xs={6}
          >
            <LoginComponent trigger="true"/>
          </Grid>
        </Grid>
      </Container>
    );
  }
```

Cartelle delle navbar:



Definition of Done

Durante questo sprint una User Story sarà definita come Done, o completata, se tutti i task che la riguardano sono stati completati, supera i test relativi nei vari casi ed è stata approvata dal PO.

Test

```
//Controlla che ci siano solo pezzi dello stesso colore su ogni lato
test('should have pieces only of the same color on each side', () => {
  const mode = 'dailyChallenge';
  const rank = 1;
  const chessBoard = generateBoard(mode, rank);
  const fen = chessBoard.board.fen();
  const whitePieces = fen.match(/[PNBRQK]/g);
  const blackPieces = fen.match(/[pnbrqk]/g);
  expect(whitePieces).toHaveLength(16);
  expect(blackPieces).toHaveLength(16);
});
```

Come utente, voglio vedere la chessboard, i suoi pezzi e che io ci possa interagire

```
test('should return an array of selected chess pieces', () => {
  const rank = 15;
  const selectedPieces = findChessPiecesWithRank(rank, 0);
  const expectedValue = (rank)/15;
  const sumOfValues = selectedPieces.reduce((sum, piece) => sum + piece.value, 0);
  const averageValue = sumOfValues / 15;

  expect(averageValue).toBeCloseTo(expectedValue,-1);
});
```

Come utente, voglio poter generare una chessboard in base a un valore che identifica il rank, in modo da avere una partita sbilanciata

```
describe('LoginPage', () => {
  it('should log in anonymously successfully', () => {
    cy.visit('/login');
    cy.get('input[name="username"]').first().type('testUserNew');
    cy.get('input[name="password"]').first().type('testPassword');
    cy.get('button[type="submit"]').contains('Login').click();

    cy.url().should('include', '/play');
    cy.get('button:contains("Gioca Really Bad Chess")').click();
    cy.url().should('include', '/play');

    // Sceglie la modalità di gioco PvPLocale
    cy.get('[aria-labelledby="mode-label"]').click();
    cy.contains('Player vs Player (locale)').click();
    // Verifica che la modalità sia stata selezionata correttamente

    // Seleziona una partita di 1 minuto
    cy.get('[aria-labelledby="duration-label"]').click();
    cy.contains('1 minute').click();

    // Inseriamo i nomi dei giocatori
    cy.get('[labelId="player1"]').type('BiancoUser');
    cy.get('[labelId="player2"]').type('NeroUser');

    // Scegliamo un svantaggio per l'avversario
    cy.get('.MuiSlider-root').click({ force: true, position: 'left'});

    //Iniziamo il gioco
    cy.get('button:contains("Start Game")').click();

    cy.url().should('include', '/play/reallybadchess/').then(() => {
      // Facciamo le mosse
      performChessMoves('w');
    });
  });
});
```

Come guest o utente, voglio poter giocare in locale contro un altro utente

```
test("should save the match", async () => {
  const response = await request(app)
    .post("/api/reallybadchess/saveGame/" + gameId)
    .set("Authorization", `Bearer ${token}`);

  expect(response.statusCode).toBe(200);
  expect(response.body.success).toBe(true);
});
```

Come utente, voglio che le mie partite di Really Bad Chess vengano salvate sul mio profilo utente

```
it('should log in anonymously successfully', () => {
  cy.visit('/login');

  // Facciamo il login
  cy.get('button[type="submit"]').contains('Continua come anonimo').click();

  // Verifica che il login sia avvenuto con successo
  cy.url().should('include', '/play');
});
```

Come guest, voglio poter entrare come anonimo

```
// Test for valid credentials
test("should return 200 or 400 for POST request with valid credentials", async () => {
  const response = await request(app)
    .post("/api/register")
    .send({ username: "testerUsername", password: "test" });
  expect([200, 400]).toContain(response.statusCode);
  if (response.statusCode === 200) {
    expect(response.body).toEqual({ success: true });
  } else {
    expect(response.body).toEqual({ success: false, reason: "Username already exists" });
  }
});
```

Come guest, voglio registrarmi al sito con un nome utente e una password, per tenere traccia dei miei progressi ed accedere a più feature

```
test("should return 200 and token for POST request with correct credentials", async () => {
  const response = await request(app)
    .post("/api/login")
    .send({ username: "testerUsername", password: "test" });

  expect(response.statusCode).toBe(200);
  expect(response.body).toHaveProperty("token");
  token = response.body.token; // Save the token for later use
});
```

Come utente (registrato), voglio poter inserire un nome utente e una password, per poter accedere al mio profilo

```
test("should return 200 for request with correct token", async () => {
  const response = await request(app)
    .post("/api/tokenTest")
    .set("Authorization", `Bearer ${token}`);

  expect(response.statusCode).toBe(200);
});
```

Come utente (registrato), voglio che la mia sessione di login sia salvata nel tempo, in modo da non dover eseguire continuamente il login

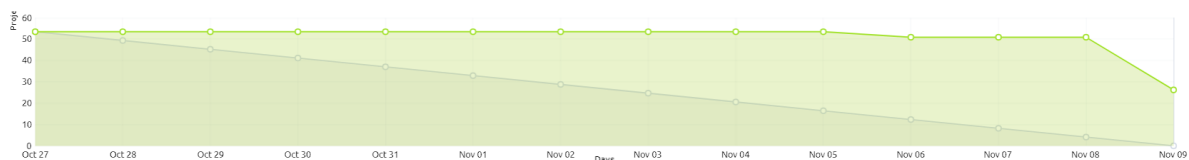
```
it('should see the leaderboards', () => {  
  cy.visit('/login');  
  cy.get('button[type="submit"]').contains('Continua come anonimo').click();  
  
  cy.url().should('include', '/play');  
  
  // Clicchiamo sul pulsante "Leaderboard"  
  cy.get('button:contains("Leaderboard")').click();  
  
  cy.url().should('include', '/leaderboard');  
  // Verifica che ci siano i pulsanti per cambiare la modalità  
  cy.get('button:contains("ELO")').click();  
  cy.wait(1000);  
  cy.get('button:contains("Rank")').click();  
  cy.wait(1000);  
  cy.get('button:contains("Daily")').click();  
  cy.wait(1000);  
});
```

Come guest o utente voglio poter navigare tra le varie interfacce, ovvero leaderboard, registrazione / login, pagina del profilo, gioco ed eventualmente altre

Burndown dello Sprint

Di seguito allego il grafico del Burndown relativo a questo sprint: esso sottolinea come il team inizialmente faceva poco utilizzo di Taiga, e come in generale in team si è ritrovato a lavorare maggiormente a ridosso della scadenza.

Entrambe problema sottolineate nella retrospettiva relativa allo sprint.



Retrospettiva

La retrospettiva del primo sprint non è stata fatta con Essence, bensì avevamo usato l'approccio Start, Stop and Continue. Che andava ad etichettare determinati modi di agire utilizzati durante lo sprint. Esso ha evidenziato aree cruciali di miglioramento: enfatizzare la priorità di certe user stories (in quanto erano state completate per prime user stories con priorità minore a discapito di quelle più importanti), aggiornare regolarmente il progresso su Taiga e mantenere pulito il branch principale. Le pratiche efficaci, come la creazione di gruppi di lavoro e le riunioni Scrum regolari, sono state mantenute durante il resto del progetto.

Sprint 2

Sprint Goals

Gli obiettivi posti dal team per il secondo sprint lavorativo erano:

1. Finire l'implementazione di Really Bad Chess:
 1. Fare le partite contro il computer
 2. Fare le partite contro un altro giocatore in locale
 3. Fare la daily challenge
2. Implementare MongoDB per gli account utente
 1. Gestire i token correttamente e usare il database per verificare le credenziali
 2. Generare nuovi account nel database
3. Implementare il timer per le partite
 1. In particolare per le partite giocatore contro giocatore locale
4. Implementare le leaderboard per gli utenti
 1. Leaderboard sul rank
 2. Leaderboard sulla quantità di mosse fatte prima di uno scacco matto in daily challenge

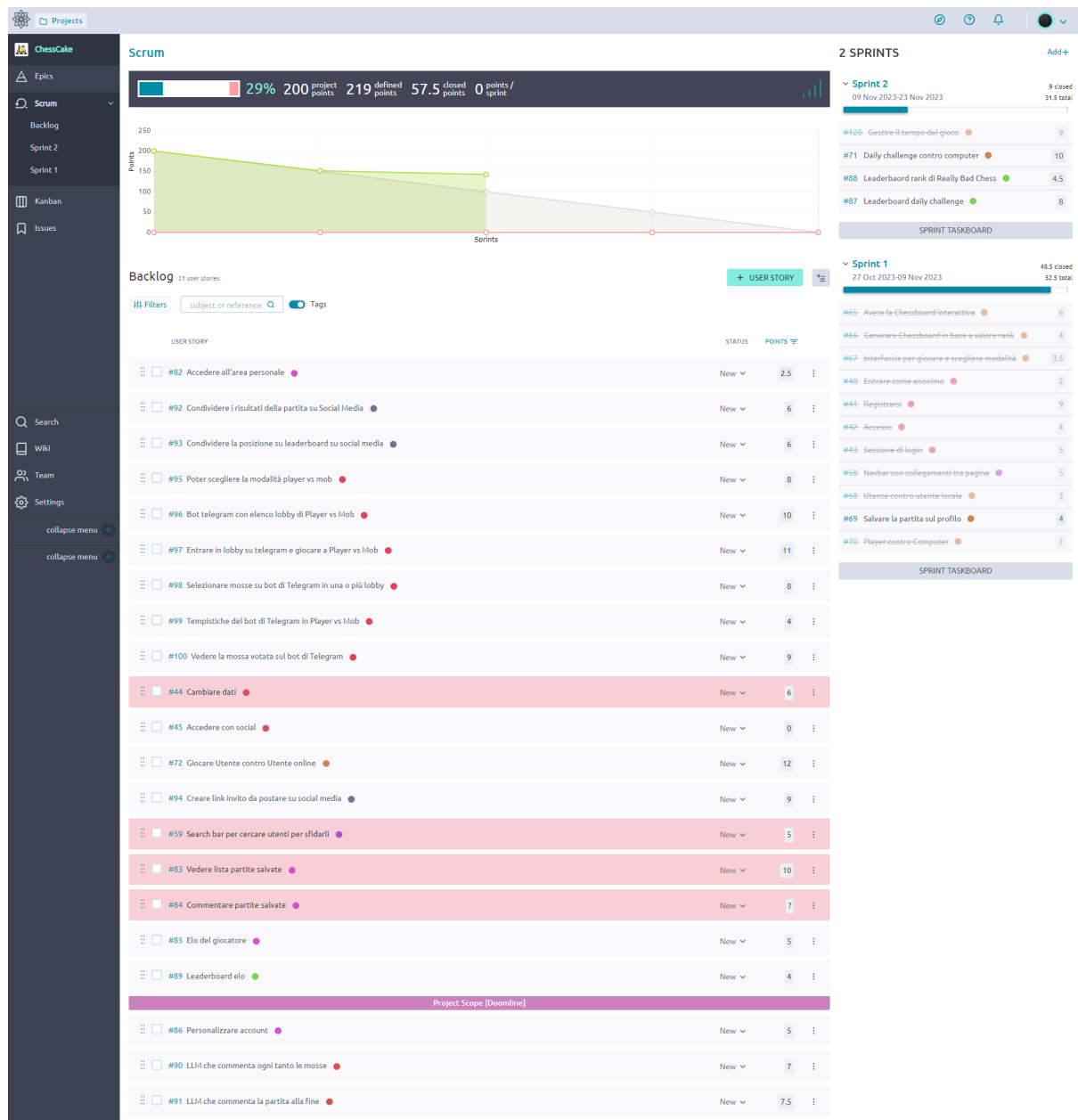
Sprint Backlog

Le User Stories relative agli Sprint Goals ideati erano:

Gestire il tempo del gioco:

- Come utente voglio che io e il mio avversario abbiamo un timer che mostra il tempo a disposizione e che, se finisce il tempo a disposizione ad uno dei giocatori lui perda in automatico la partita (Punti: 9)

Con rispettiva immagine del backlog:



Descrizione del codice

Dopo aver setuppato il server, abbiamo iniziato a scrivere le prime funzioni relative al login

```
router.post("/login", function (req, res) {  
  // Cerchiamo nel req body se vi sono tutti i parametri non nulli, ovvero username e password  
  if (req.body.username && req.body.password) {  
    // Se sono presenti, li salviamo in due variabili  
    let username = req.body.username;  
    let password = req.body.password;  
  
    // Eseguiamo il login  
    logUser(username, password).then((result) => {  
      // Se non ci sono stati errori, ritorniamo un 200  
      res.status(result.status);  
  
      // Ritorniamo un json con il token e un flag di successo  
      // e impostiamo gli header in modo corretto  
      let resBody = result.returnBody;  
  
      res.header("Content-Type", "application/json");  
      res.header("Content-Length", resBody.length);  
  
      res.send(resBody);  
    }).catch((error) => {  
      // Se si è verificato un errore, lo stampiamo in console  
      console.log(error.message);  
      // e ritorniamo un errore 500  
      res.status(error.status).send(error.returnBody);  
    });  
  } else {  
    console.log("Non sono presenti tutti i parametri");  
    // Se non sono presenti tutti i parametri, ritorniamo un errore 400  
    res.status(400).send({ success: false });  
  }  
});
```

Abbiamo successivamente implementato un sistema di token utilizzando la libreria JWT, che ci permette di mantenere tokens di sessione.

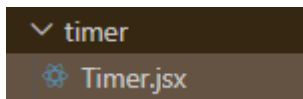
```
const authenticateJWT = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (authHeader) {
    const token = authHeader.split(" ")[1];

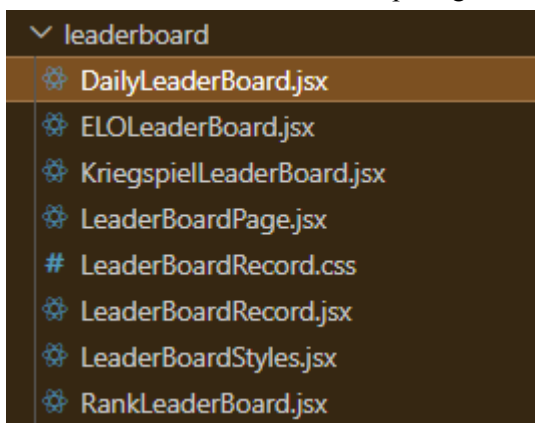
    jwt.verify(token, config.SECRET_KEY, (err, user) => {
      if (err) {
        return res.status(403).send({
          success: false,
          message: "Unauthorized",
        });
      }

      req.user = user;
      next();
    });
  } else {
    return res.status(401).send({
      success: false,
      message: "Unauthorized",
    });
  }
};
```

Cartella relativa al timer:



Cartelle relative alle leaderboard per ogni modalità di gioco:



Definition of Done

La definizione di Done è rimasta invariata rispetto allo sprint precedente.

Test

```
test("should return 200 for post to /api/reallybadchess/newGame", async () => {
  const response = await request(app)
    .post("/api/reallybadchess/newGame")
    .send({ username: "testerUsername", settings: { mode: "playerVsComputer" } })
    .set("Authorization", `Bearer ${token}`);
  expect(response.statusCode).toBe(200);
  gameID = response.body.gameId;
  expect(gameID).toBeDefined();
});

test("should make moves" , async () => {
  const pick = await request(app)
    .get("/api/reallybadchess/getGame/" + gameID)
    .set("Authorization", `Bearer ${token}`);

  expect(pick.statusCode).toBe(200);
  expect(pick.gameID).not.toBe(null);
  const chess = new Chess(pick.body.game.chess._header.FEN);

  //Facciamo una mossa casuale dalla square d2
  const randomMove = chess.moves({verbose: true})[0];
  const response = await request(app)
    .post("/api/reallybadchess/movePiece/" + gameID)
    .send({
      from: randomMove.from,
      to: randomMove.to,
      promotion: "q",
    })
    .set("Authorization", `Bearer ${token}`);
  expect(response.statusCode).toBe(200);
  expect(response.body.success).toBe(true);
  expect(response.statusCode).toBe(200);
  expect(response.body.success).toBe(true);

  // Aspetta 3 secondi
  await new Promise((resolve) => setTimeout(resolve, 5000));

  // Prendi il gioco
  const pick2 = await request(app)
    .get("/api/reallybadchess/getGame/" + gameID)
    .set("Authorization", `Bearer ${token}`);

  expect(pick2.statusCode).toBe(200);
  expect(pick2.body.game.chess._turn).toBe("w"); // Controlla se è di nuovo il tuo turno
}, 10000);
```

Fare le partite contro il computer

```
describe('Simulazione di due utenti', () => {
  it('Utente 1 - Login e Inizio Partita', () => {
    // Login per l'Utente 1
    cy.visit('/login');
    cy.get('[labelId=usernameRegisterField]').should('exist').type('testUser');
    cy.get('[labelId=passwordRegisterField]').should('exist').type('testPassword');
    cy.get('button[type="submit"]').contains('Login').click();

    // Controlliamo che il login sia andato a buon fine
    cy.url().should('include', '/play');
    cy.get('button:contains("Gioca Really Bad Chess")').click();
    cy.url().should('include', '/play');

    // Scegliamo PvPOnline
    cy.get('[aria-labelledby="mode-label"]').click();
    cy.contains('Player vs Player (online)').click();

    // Scegliamo il tempo
    cy.get('[aria-labelledby="duration-label"]').click();
    cy.contains('1 minute').click();

    // Partiamo e aspettiamo che entri il secondo test
    cy.get('button:contains("Create Game")').click();
    cy.wait(13000);
    cy.url().should('include', '/play/reallybadchess/').then(() => {
      // Giochiamo
      performChessMoves('b');
    });
  });
});
```

Fare le partite contro un altro giocatore in locale

```
//Creiamo un nuovo gioco e salviamo l'id
let gameId = null;
test("should return 200 for post to /api/reallybadchess/newGame", async () => {
  const response = await request(app)
    .post("/api/reallybadchess/newGame")
    .send({ username: "testerUsername", settings: { mode: "dailyChallenge" } })
    .set("Authorization", `Bearer ${token}`);

  expect(response.statusCode).toBe(200);
  gameId = response.body.gameId;
  expect(gameId).toBeDefined();
});

//Controlliamo se il gioco è stato creato
test("should return 200 for post to /api/reallybadchess/getGame/:gameId", async () => {
  const response = await request(app)
    .get("/api/reallybadchess/getGame/" + gameId)
    .set("Authorization", `Bearer ${token}`);

  expect(response.statusCode).toBe(200);
  expect(response.gameId).not.toBe(null);
});

//Facciamo una mossa casuale all'interno del gioco creato
test("should return 200 for post to /api/reallybadchess/movePiece/:gameId", async () => {
  const pick = await request(app)
    .get("/api/reallybadchess/getGame/" + gameId)
    .set("Authorization", `Bearer ${token}`);

  expect(pick.statusCode).toBe(200);
  expect(pick.gameId).not.toBe(null);
  const chess = new Chess(pick.body.game.chess._header.FEN);

  //Facciamo una mossa casuale dalla square d2
  const randomMove = chess.moves({verbose: true})[0];
  const response = await request(app)
    .post("/api/reallybadchess/movePiece/" + gameId)
    .send({
      from: randomMove.from,
      to: randomMove.to,
      promotion: "q",
    })
    .set("Authorization", `Bearer ${token}`);

  expect(response.statusCode).toBe(200);
  expect(response.body.success).toBe(true);
});
```

Fare la daily challenge

```
function performChessMoves(color) {
  cy.get(`[data-square="${generateStartSquare(color)}"] > div`).click({ force: true });
  cy.get(`[data-square="${generateMoveSquare()}"] > div`).click({ force: true }).then(() => {
    // Controlla la presenza di due pulsanti "Esci"
    cy.get('button:contains("Esci)').then(($buttons) => {
      if ($buttons.length === 2) {
        // Se ci sono due pulsanti "Esci", la partita è finita
        cy.log('La partita è finita.');
```

```
        cy.get('button:contains("Esci)').eq(1).click(); // Clicca sul secondo pulsante "Esci"
      } else {
        // Se non ci sono due pulsanti "Esci", la partita non è finita, continua a giocare
        cy.wait(500);
        performChessMoves(color);
      }
    });
  });
}
```

In particolare per le partite giocatore contro giocatore locale

```
test("should return leaderboard data for rank", async () => {
  const response = await request(app)
    .get("/api/leaderboard/rank")
    .set("Authorization", `Bearer ${token}`);

  expect(response.statusCode).toBe(200);
  expect(response.body).toBeDefined();
  expect(response.body.success).toBe(true);
  expect(response.body.leaderboard).toBeDefined();
  expect(response.body.leaderboard.length).toBeGreaterThan(0);
  expect(response.body.userPlace).toBeDefined();
});
```

Leaderboard sul rank

```
test("should return leaderboard data for daily", async () => {
  const response = await request(app)
    .get("/api/leaderboard/daily")
    .set("Authorization", `Bearer ${token}`);

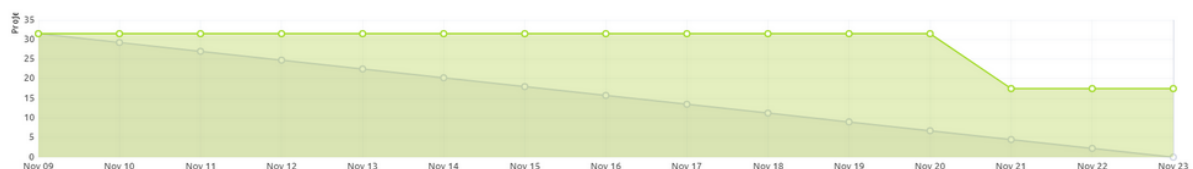
  expect([200, 201]).toContain(response.statusCode);
  expect(response.body.success).toBe(true);
  expect(response.body.leaderboard).toBeDefined();
  expect(response.body).toBeDefined();
  if (response.statusCode === 200)
    expect(response.body.leaderboard.length).toBeGreaterThan(0);
  expect(response.body.userPlace).toBeDefined();
});
```

Leaderboard sulla quantità di mosse fatte prima di uno scacco matto in daily challenge

Burndown dello Sprint

Si nota un piccolo miglioramento nel grafico, però comunque il lavoro rimane per la maggior parte affrontato alla fine dello sprint.

Di seguito allego il grafico del Burndown relativo a questo sprint:

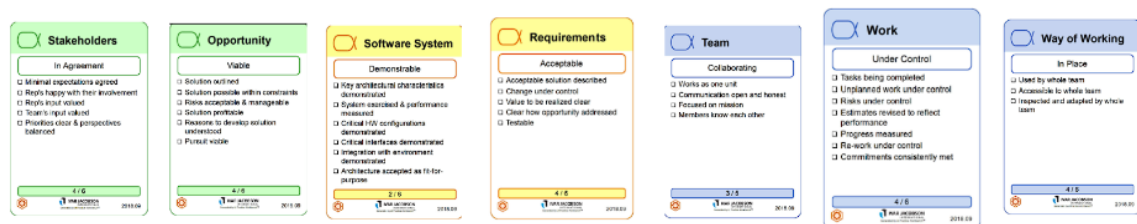


Retrospettiva

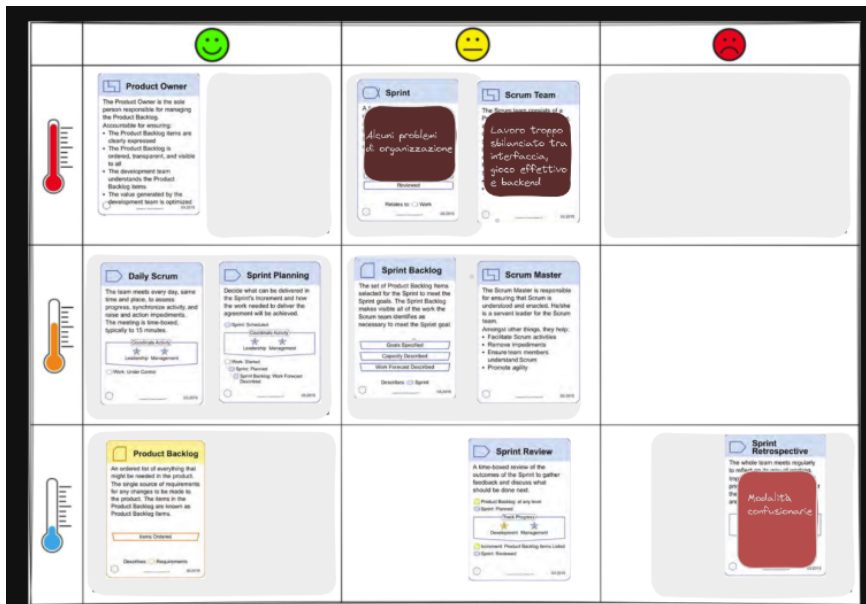
Da questo sprint abbiamo iniziato ad utilizzare la metodologia Essence per guidare la retrospettiva.

Risultati del Progress Poker:

Quindi le corrispondenti carte di progress poker sono:



Risultati di Mad, Sad, Glad:



Con l'utilizzo delle carte ci siamo potuti focalizzare sui macro-argomenti relativi alla nostra esperienza di lavoro e abbiamo potuto discutere su come migliorare il nostro lavoro.

Sono sorte alcune migliorie riguardanti l'approccio alla programmazione, la distribuzione dei compiti, la comunicazione e i placeholder del codice.

Retrospettiva

I goals sono stati completati parzialmente, in quanto:

- Abbiamo completato il gioco di scachi, le interfacce e ciò che abbiamo lasciato indietro da parte del primo sprint
- Abbiamo completato il timer, funzionante
- Abbiamo creato dei test di unità per determinate funzionalità, anche se non tutte, soprattutto quelle del frontend

Tuttavia non abbiamo completato le leaderboard, in quanto avevano priorità inferiori rispetto al resto.

Come migliorare

- Approccio con test:
 - Approcciare la programmazione dapprima con un'idea su come eseguire dei test di unità e provare l'applicazione man mano.
- Migliorare la suddivisione dei membri:
 - Avere una giusta quantità di membri dedicata ad ognuno dei vari compiti, in modo da non avere un sovraccarico di lavoro per alcuni membri e un sottocarico per altri.
 - Questo deve derivare da un'analisi più dettagliata e una valutazione migliore del costo di sviluppo di ogni singola funzionalità.
- Migliorare la comunicazione:
 - Avere una comunicazione più costante tra i membri del team, in modo da avere una visione più chiara del lavoro svolto e di quello da svolgere.
- Evitare i placeholder:
 - Evitare di lasciare codice non completo o comunque che dovrà essere sostituito così spesso, in modo da non accumulare debito tecnico

Sprint 3

Sprint Goals

In questo sprint avevamo deciso di cambiare gli obiettivi del terzo sprint a seguito delle richieste degli stakeholder, in modo da soddisfarle. I nuovi obiettivi erano:

1. Implementare really bad chess nel backend è rimasto una priorità, in quanto è alla base di ogni nuovo passo tecnologico
2. Implementare l'online
3. Recuperare il debito tecnico è comunque essenziale, per rimanere coerenti con le nostre idee, quindi è rimasto un obiettivo, pertanto ci siamo concentrati anche sul fare le leaderboard e gli account (serve anche per poter meglio vedere i risultati ottenuti, quindi è effettivamente un obiettivo importante)
4. Se possibile, implementare la condivisione dei risultati su facebook / X

Sprint Backlog

Le User Stories relative agli Sprint Goals ideati erano:

Accedere all'area personale:

- Come utente registrato voglio poter accedere all'area personale, dove posso vedere il mio profilo, ovvero il mio nome e i dati relativi alle partite che ho giocato (Punti: 2.5)

Vedere lista partite salvate:

- Come utente registrato voglio vedere la lista delle partite fatte (Punti: 10)

Elo del giocatore:

- Come utente registrato voglio essere valutato da una funzione di elo per tenere traccia dei miei progressi, mettermi a confronto con altri giocatori e risultare nelle leaderboard (Punti: 5)

Poter vedere il replay delle partite:

- Come utente voglio poter guardare il replay di una delle partite che ho fatto di Really Bad Chess all'interno dell'elenco partite sul mio profilo o sul profilo di altri (Punti: 9)

Daily challenge contro computer:

- Come utente voglio poter giocare alla daily challenge contro il computer e che il numero di mosse per fare scacco venga salvato (Punti: 6)

Giocare Utente contro Utente online:

- Come utente voglio poter creare lobby per giocare contro altri giocatori, e voglio vedere le lobby disponibili e sfidare i giocatori (Punti: 19)

Evitare esecuzioni false:

- Come utente voglio che i risultati che esistono nella leaderboard non siano sbagliati e che nel giocare le partite vi siano dei controlli sull'esecuzione delle mosse, su chi sta eseguendo le mosse, etc. (Punti: 15)

Leaderboard daily challenge:

- Come guest o utente voglio vedere le leaderboard di chi ha battuto la daily challenge di Really Bad Chess nel minor numero di mosse (Punti: 8)

Leaderboard rank di Really Bad Chess:

- Come utente o guest voglio vedere i nomi dei giocatori che hanno vinto con il rank più alto in Really Bad Chess (Punti: 4.5)

Leaderboard elo:

- Come utente o guest voglio poter vedere gli elo più alti in Really Bad Chess (Punti: 4)

Condividere i risultati della partita su Social Media:

- Come utente o guest voglio poter condividere i risultati delle mie partite su social network attraverso un post (Punti: 6)

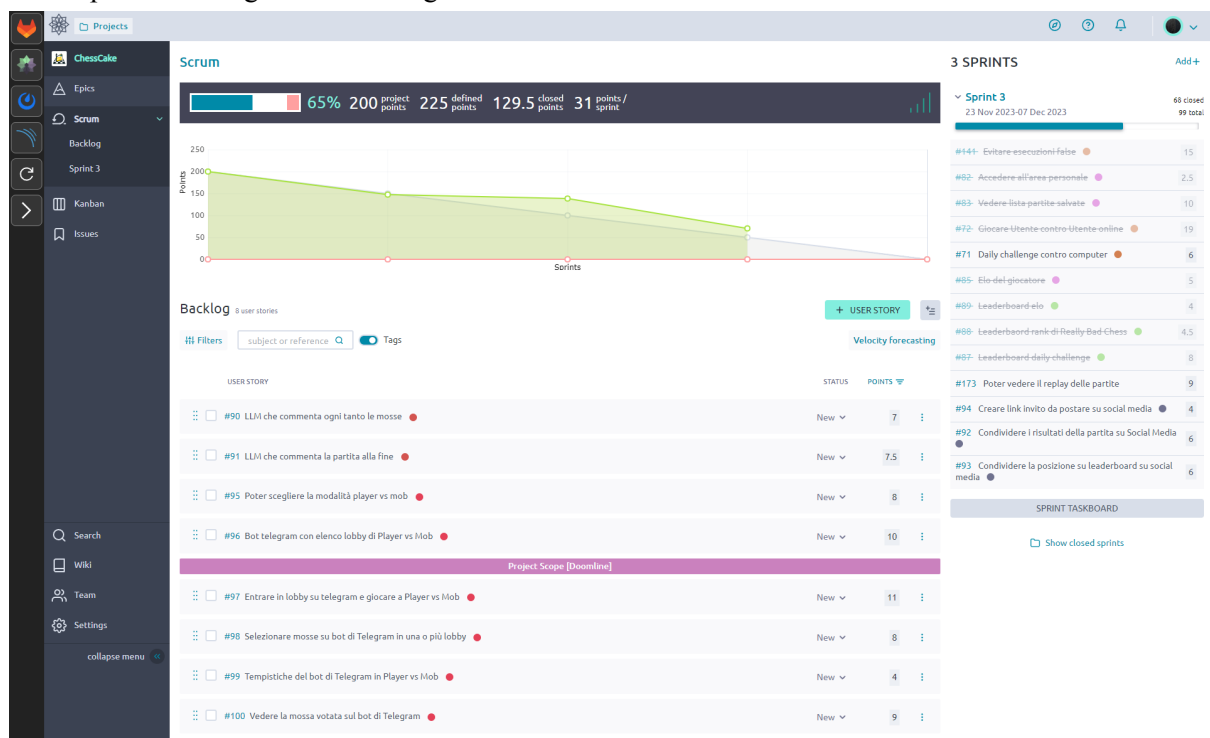
Condividere la posizione su leaderboard su social media:

- Come utente registrato voglio poter condividere la mia posizione nelle leaderboards sui social media (Punti: 6)

Creare link invito da postare su social media:

- Come utente voglio poter creare un link di invito ad una partita da postare sui social media, in modo che altri giocatori possano entrare nella mia lobby e giocare contro di me (Punti: 4)

Con rispettiva immagine del backlog:



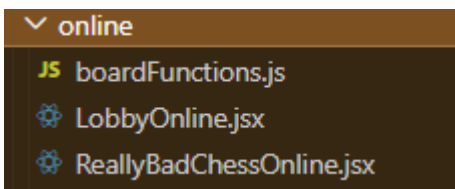
Descrizione del codice

Una volta implementato il database e salvatici dei games sopra abbiamo implementato funzioni che ci permettessero di accedervi.

Il gioco qui è stato finalmente spostato nel backend, dopo le problematiche dei primi due sprints.

```
router.get("/getGame/:gameId", authenticateJWT, async (req, res) => {  
  // Prendiamo il gameId  
  const { gameId } = req.params;  
  // Prendiamo il game dal database  
  const game = chessGames.getGame(gameId);  
  
  // Se il game non esiste, allora ritorniamo un errore  
  if (!game) {  
    return res.status(404).send({  
      success: false,  
      message: "Game not found",  
    });  
  }  
  
  resizeGame(game)  
  
  // Se il game esiste, allora ritorniamo il game  
  res.send({  
    success: true,  
    game: resizeGame(game),  
  });  
});
```

Una volta spostato il gioco nel backend è stato possibile creare la modalità Player vs Player di Really Bad Chess:



Esempio di richiesta di eseguire una mossa da frontend al server

```
const makeMove = async (sourceSquare, targetSquare, username) => {  
  // Gestisco il fatto che non si possa muovere se non è il proprio turno  
  // Questo evita richieste inutili  
  await fetch(`/api/reallybadchess/movePiece/${gameId}`, {  
    method: "POST",  
    headers: {  
      "Content-Type": "application/json",  
      Authorization: `Bearer ${Cookies.get("token")}`,  
    },  
    body: JSON.stringify({  
      username: username,  
      from: sourceSquare,  
      to: targetSquare,  
      promotion: "q",  
    }),  
  })  
  .then((response) => response.json())  
  .then((data) => {  
    if (data.success) {  
      handleGetGameResponse(data);  
    }  
  })  
  .catch((err) => {  
    console.log(err);  
  });  
};
```

Successivamente è stata implementata la connettività con i Social Media, con l'utilizzo della libreria "react-share", che gestisce autonomamente la sessione di login sulle piattaforme e che importa metodi utili alla creazione di un post.

- ShareAppList.jsx
- ShareButton.jsx
- SharePaper.jsx

```
import {  
  FacebookShareButton,  
  TwitterShareButton,  
  XIcon,  
  RedditShareButton,  
} from "react-share";
```

```
<FacebookShareButton
  url={url}
  hashtag="#chesscake"
  style={{
    display: "flex",
    alignItems: "center",
    justifyContent: "space-evenly",
    padding: "8px",
    width: "130px",
    backgroundColor: "#3b5998",
    color: "#fff",
    fontFamily: "Roboto, Helvetica, Arial, sans-serif",
    borderRadius: "10px",
    fontSize: "0.875rem",
  }}
>
  <Facebook />
  Condividi
</FacebookShareButton>
```

Definition of Done

La definizione di Done è rimasta invariata rispetto allo sprint precedente.

Test

```
test("should return account data for existing user", async () => {
  const response = await request(app)
    .get("/api/account/getAccountData/testerUsername")
    .set("Authorization", `Bearer ${token}`);
  expect(response.statusCode).toBe(200);
  expect(response.body).toBeDefined();
});
```

Come utente registrato voglio poter accedere all'area personale, dove posso vedere il mio profilo, ovvero il mio nome e i dati relativi alle partite che ho giocato

```
//Creiamo un nuovo gioco e salviamo l'id
let gameId = null;
test("should return 200 for post to /api/reallybadchess/newGame", async () => {
  const response = await request(app)
    .post("/api/reallybadchess/newGame")
    .send({ username: "testerUsername", settings: { mode: "dailyChallenge" } })
    .set("Authorization", `Bearer ${token}`);

  expect(response.statusCode).toBe(200);
  gameId = response.body.gameId;
  expect(gameId).toBeDefined();
});

//Controlliamo se il gioco è stato creato
test("should return 200 for post to /api/reallybadchess/getGame/:gameId", async () => {
  const response = await request(app)
    .get("/api/reallybadchess/getGame/" + gameId)
    .set("Authorization", `Bearer ${token}`);

  expect(response.statusCode).toBe(200);
  expect(response.gameId).not.toBe(null);
});

//Facciamo una mossa casuale all'interno del gioco creato
test("should return 200 for post to /api/reallybadchess/movePiece/:gameId", async () => {
  const pick = await request(app)
    .get("/api/reallybadchess/getGame/" + gameId)
    .set("Authorization", `Bearer ${token}`);

  expect(pick.statusCode).toBe(200);
  expect(pick.gameId).not.toBe(null);
  const chess = new Chess(pick.body.game.chess._header.FEN);

  //Facciamo una mossa casuale dalla square d2
  const randomMove = chess.moves({verbose: true})[0];
  const response = await request(app)
    .post("/api/reallybadchess/movePiece/" + gameId)
    .send({
      from: randomMove.from,
      to: randomMove.to,
      promotion: "q",
    })
    .set("Authorization", `Bearer ${token}`);

  expect(response.statusCode).toBe(200);
  expect(response.body.success).toBe(true);
});
```

Come utente voglio poter giocare alla daily challenge contro il computer e che il numero di mosse per fare scacco venga salvato

```
describe('Simulazione di due utenti', () => {
  it('Utente 1 - Login e Inizio Partita', () => {
    // Login per l'Utente 1
    cy.visit('/login');
    cy.get('[labelId="usernameRegisterField"]').should('exist').type('testUser');
    cy.get('[labelId="passwordRegisterField"]').should('exist').type('testPassword');
    cy.get('button[type="submit"]').contains('Login').click();

    // Controlliamo che il login sia andato a buon fine
    cy.url().should('include', '/play');
    cy.get('button:contains("Gioca Really Bad Chess")').click();
    cy.url().should('include', '/play');

    // Scegliamo PvPOnline
    cy.get('[aria-labelledby="mode-label"]').click();
    cy.contains('Player vs Player (online)').click();

    // Scegliamo il tempo
    cy.get('[aria-labelledby="duration-label"]').click();
    cy.contains('1 minute').click();

    // Partiamo e aspettiamo che entri il secondo test
    cy.get('button:contains("Create Game")').click();
    cy.wait(13000);
    cy.url().should('include', '/play/reallybadchess/').then(() => {
      // Giochiamo
      performChessMoves('b');
    });
  });
});
```

Come utente voglio poter creare lobby per giocare contro altri giocatori, e voglio vedere le lobby disponibili e sfidare i giocatori

```
test("should return leaderboard data for daily", async () => {
  const response = await request(app)
    .get("/api/leaderboard/daily")
    .set("Authorization", `Bearer ${token}`);

  expect([200, 201]).toContain(response.statusCode);
  expect(response.body.success).toBe(true);
  expect(response.body.leaderboard).toBeDefined();
  expect(response.body).toBeDefined();
  if (response.statusCode === 200)
    expect(response.body.leaderboard.length).toBeGreaterThan(0);
  expect(response.body.userPlace).toBeDefined();
});
```


Come guest o utente voglio vedere le leaderboard di chi ha battuto la daily challenge di Really Bad Chess nel minor numero di mosse

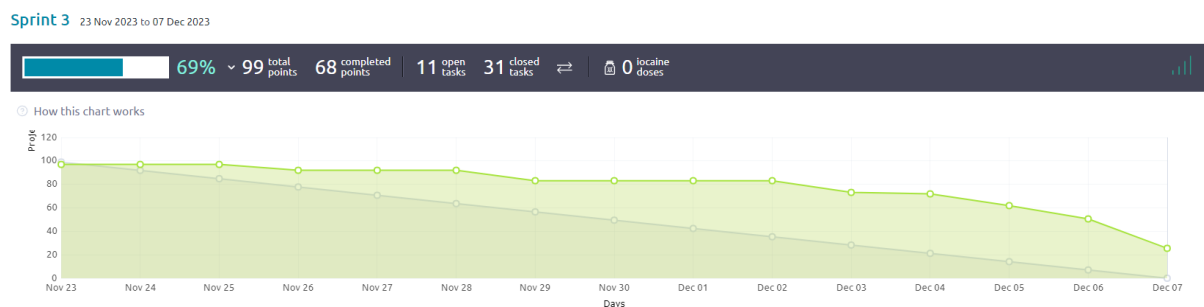
```
test("should return leaderboard data for rank", async () => {  
  const response = await request(app)  
    .get("/api/leaderboard/rank")  
    .set("Authorization", `Bearer ${token}`);  
  
  expect(response.statusCode).toBe(200);  
  expect(response.body).toBeDefined();  
  expect(response.body.success).toBe(true);  
  expect(response.body.leaderboard).toBeDefined();  
  expect(response.body.leaderboard.length).toBeGreaterThan(0);  
  expect(response.body.userPlace).toBeDefined();  
});
```

Come utente o guest voglio poter vedere gli elo più alti in Really Bad Chess

Burndown dello Sprint

Di seguito allego il grafico del Burndown relativo a questo sprint:

Per la prima volta il chart inizia ad avere una forma decente, di graduale discesa nel tempo. Il team infatti è entrato in pieno regime di lavoro, tutti hanno adottato una modalità di lavorare comune e la produttività è incrementata.






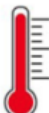





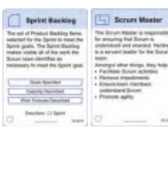
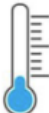

Retrospettiva

Risultati del Progress Poker e Mad, Sad, Glad:

Progress Poker

Abbiamo utilizzato le carte Essence¹ relative a questi punti per dare le valutazioni. Pertanto nella tabella viene trascritto il valore numerico tra parentesi (genericamente compreso tra 1 e 6, ma a volte i valori cambiano. Ogni valore ha una check list relativa ed un significato importante, fare riferimento alle carte Essence²).

Carta \ Membro	Petru	Davide	Saad	Alex	Rafid	Giuseppe	Consenso Finale
Stakeholders	4/6	5/6	4/6	4/6	4/6	5/6	5/6
Opportunity	//	//	//	//	//	//	(non è un prodotto con soluzioni che hanno vincoli economici)
Software System	3/6	4/6	3/6	3/6	3/6	4/6	3/6
Requirements	5/6	4/6	4/6	5/6	5/6	5/6	4/6
Team	3/5	4/5	3/5	3/5	3/5	4/5	3/5
Work	4/6	4/6	4/6	4/6	4/6	4/6	4/6
Way of Working	5/6	5/6	4/6	4/6	5/6	4/6	4/6

Rispetto allo sprint precedente la retrospettiva non è andata molto diversamente. I problemi riguardanti l'organizzazione del lavoro sono sussistiti, facendo questa volta evidenza però sulle skills

dei membri. In quanto membri inesperti si trovavano ad avere problemi con la quantità di lavoro a loro attribuita.

Un ulteriore problema sorto era relativo alle tempistiche di consegna, dove il team si è imposto di mantenere consegne più frequenti, ma ridotte in quantità di lavoro.

Sprint 4

Sprint Goals

I nostri obiettivi per il quarto sprint sono stati i seguenti:

- Finire la condivisione sui social (della posizione nella leaderboard, del risultato della partita e del link di invito)
- Finire i replay (di Really Bad Chess)
- Creare Kriegspiel online in giocatore contro giocatore
- Gestire i dati degli utenti relativi a Kriegspiel (mostrare sull'account l'ELO, il numero di partite giocate e vinte, ecc.)

Questi goals erano per creare un prodotto che combaciasse con i requisiti richiesti dagli stakeholder, ovvero integrare:

- Più modalità di scacchi eterodossi
- Integrazione con i social (nel nostro caso un'integrazione basilare ma comunque funzionante)
- Timer condiviso per le partite
- Gioco multiplayer online, con gestione dell'ELO

Abbiamo anche deciso di rimandare la parte di copertura e di testing all'ultimo sprint, quello conclusivo. Non sappiamo necessariamente se questo è stato saggio, ma sicuramente ci ha permesso di produrre un MVP più soddisfacente e senza molto overhead di lavoro.

Sprint Backlog

Le User Stories relative agli Sprint Goals ideati erano:

Interfaccia per vedere le lobby:

- Come utente, voglio avere un'interfaccia dove posso vedere l'elenco di lobby aperte nelle quali io posso entrare (Punti: 2.5)

API per richiedere le partite:

- Come utente, voglio poter ottenere l'elenco di partite nelle quali posso entrare facendo una richiesta al server (Punti: 2)

Creare nuova partita con durata:

- Come utente, voglio poter creare una nuova partita online, specificando la durata, e che questa partita venga aggiunta all'elenco di partite "aperte" (dove si può entrare) (Punti: 3)

Poter entrare in una partita:

- Come utente registrato, voglio poter entrare in una partita di Kriegspiel contro un altro utente (Punti: 2)

Fare un "Getter" della partita:

- Come utente registrato voglio poter ottenere i dettagli della partita della quale sono dentro (Punti: 3)

Fare un "makeMove" della partita:

- Come utente voglio poter muovere i pezzi nella scacchiera e far sì che il cambiamento si veda anche per il giocatore avversario (Punti: 3)

Fare una console che mostra quello che viene comunicato dall'umpire:

- Come utente, voglio vedere quello che l'umpire mi comunica (Punti: 6.5)

Leaderboard Kriegspiel:

- Come utente o guest, voglio poter vedere una leaderboard degli ELO più alti su Kriegspiel (Punti: 4.5)

Dati nell'account con Kriegspiel:

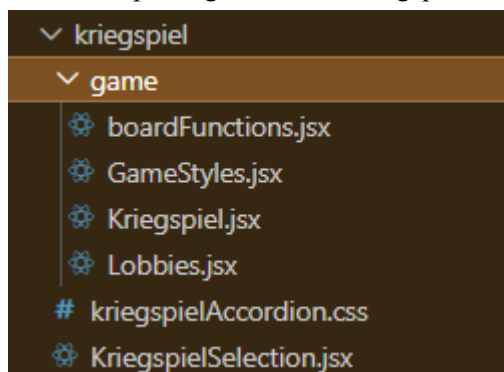
- Come utente, voglio che siano visibili nell'elenco di partite anche le partite di Kriegspiel e che ci sia il mio ELO e winrate su Kriegspiel (Punti: 4.5)

Con rispettiva immagine del backlog:



Descrizione del codice

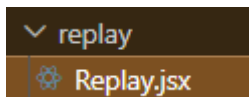
Nell'ultimo sprint, secondo le richieste degli Stakeholders il team ha implementato un secondo gioco, basato sempre sugli scacchi: Kriegspiel



Quello che rende Kriegspiel diverso è l'impossibilità di vedere i pezzi del nemico, per questo ho deciso di mostrare la seguente funzione:

```
const renderCustomPiece = () => {  
  if (playerSide == "b") {  
    return {  
      "wK": "({visibility: hidden})",  
      "wQ": "({visibility: hidden})",  
      "wR": "({visibility: hidden})",  
      "wB": "({visibility: hidden})",  
      "wN": "({visibility: hidden})",  
      "wP": "({visibility: hidden})",  
    };  
  } else {  
    return {  
      "bK": "({visibility: hidden})",  
      "bQ": "({visibility: hidden})",  
      "bR": "({visibility: hidden})",  
      "bB": "({visibility: hidden})",  
      "bN": "({visibility: hidden})",  
      "bP": "({visibility: hidden})",  
    };  
  }  
};
```

Come ultima cosa sono stati resi disponibili i replay per ogni modalità:



Definition of Done

La definizione di Done è rimasta invariata rispetto allo sprint precedente.

Test

```
test('Create a kriegspiel game, and check for the Threefold repetition rule', async () => {
  gameSettingsKriegspiel = {
    settings:{
      mode: 'kriegspiel', // Imposta la modalità desiderata per il test
      rank: 20,
      duration: 15,
    }
  };
  const newGame = await createNewGameWithSettings(player1, player2, gameSettingsKriegspiel);
  const chess = new Chess();
  chess.move('e2e4');
  checkThreefoldRepetition(newGame.gameId, chess);
});
```

Come utente, voglio poter creare una nuova partita online, specificando la durata, e che questa partita venga aggiunta all'elenco di partite “aperte” (dove si può entrare)

```
test("should join a game and move a Piece", async () => {
  // Facciamo il login prima di fare i test, poiché serve un token valido
  const response = await request(app)
    .post("/api/kriegspiel/joinGame/" + gameId)
    .set("Authorization", `Bearer ${token2}`);

  expect(response.statusCode).toBe(200);
  expect(response.body.success).toBe(true);
});
```

Come utente registrato, voglio poter entrare in una partita di Kriegspiel contro un altro utente

```
//Controlliamo se il gioco è stato creato
test("should return 200 for post to /api/kriegspiel/getGame/:gameId/user", async () => {
  const response = await request(app)
    .get("/api/kriegspiel/getGame/" + gameId + "/user")
    .set("Authorization", `Bearer ${token}`);

  expect(response.statusCode).toBe(200);
  expect(response.gameId).not.toBe(null);
  expect(response.body.success).toBe(true);
});
```

Come utente registrato voglio poter ottenere i dettagli della partita della quale sono dentro

```
test("should join a game and move a Piece", async () => {  
  // Facciamo il login prima di fare i test, poiché serve un token valido  
  const response = await request(app)  
    .post("/api/kriegspiel/joinGame/" + gameId)  
    .set("Authorization", `Bearer ${token2}`);  
  
  expect(response.statusCode).toBe(200);  
  expect(response.body.success).toBe(true);  
  const move = await request(app)  
    .post("/api/kriegspiel/movePiece/" + gameId)  
    .send({  
      from: "c2",  
      to: "c3",  
      promotion: "q",  
    })  
    .set("Authorization", `Bearer ${token2}`);  
  expect([200, 400]).toContain(move.statusCode);  
  expect([true, false]).toContain(move.body.success);  
  if (move.statusCode === 400) {  
    expect(move.body.message).toBe("It's not your turn");  
  }  
});
```

Come utente voglio poter muovere i pezzi nella scacchiera e far sì che il cambiamento si veda anche per il giocatore avversario


```
test('should check if the game isDrawable', async () => {
  const response = await request(app)
    .post('/api/kriegspiel/isDrawable/' + gameId)
    .set('Authorization', `Bearer ${token}`);

  expect(response.status).toBe(200);
});

test('should not tell the game is drawable because it doesnt exist', async () => {
  const response = await request(app)
    .post('/api/kriegspiel/isDrawable/' + gameId + "error")
    .set('Authorization', `Bearer ${token}`);
  expect(response.status).toBe(404);
  expect(response.body.success).toBe(false);
  expect(response.body.message).toBe("Game not found");
});

test('should not tell the game is drawable because you are not part of the game', async () => {
  const response = await request(app)
    .post('/api/kriegspiel/isDrawable/' + gameId)
    .set("Authorization", `Bearer ${token3}`);
  expect(response.status).toBe(403);
  expect(response.body.success).toBe(false);
  expect(response.body.message).toBe("Unauthorized");
});

test("should draw the game" , async () => {
  const response = await request(app)
    .post("/api/kriegspiel/draw/" + gameId)
    .set("Authorization", `Bearer ${token}`);

  expect(response.statusCode).toBe(200);
  expect(response.body.success).toBe(true);
});

test("should not draw the game because it doesn't exist" , async () => {
  const response = await request(app)
    .post("/api/kriegspiel/draw/" + gameId + "error")
    .set("Authorization", `Bearer ${token}`);

  expect(response.statusCode).toBe(404);
  expect(response.body.success).toBe(false);
  expect(response.body.message).toBe("Game not found");
});
```

Come utente, voglio vedere quello che l'umpire mi comunica

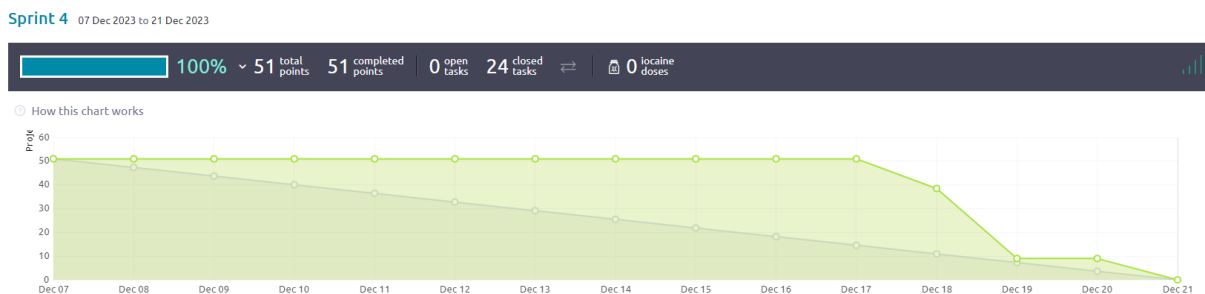
```
test("should return leaderboard data for kriegspiel", async () => {  
  const response = await request(app)  
    .get("/api/leaderboard/eloKriegspiel")  
    .set("Authorization", `Bearer ${token}`);  
  
  expect([200, 201]).toContain(response.statusCode);  
  expect(response.body.success).toBe(true);  
  expect(response.body.leaderboard).toBeDefined();  
  expect(response.body).toBeDefined();  
  if (response.statusCode === 200)  
    expect(response.body.leaderboard.length).toBeGreaterThan(0);  
  expect(response.body.userPlace).toBeDefined();  
});
```

Come utente o guest, voglio poter vedere una leaderboard degli ELO più alti su Kriegspiel

Burndown dello Sprint

Con la presenza degli esami il team ha deciso di lavorare meno durante l'inizio dello sprint di proposito, in quanto le User Stories da finire erano rimaste poche.

Questo viene descritto dal grafico di Burndown sottostante:



Retrospettiva

Risultati del Progress Poker:

Progress Poker

Come per la scorsa retrospective, abbiamo utilizzato le carte Essence¹ relative a questi punti per dare le valutazioni. Pertanto nella tabella viene trascritto il valore numerico tra parentesi (genericamente compreso tra 1 e 6, ma a volte i valori cambiano. Ogni valore ha una check list relativa ed un significato importante, fare riferimento alle carte Essence²).

Carta \ Membro	Petru	Davide	Saad	Alex	Rafid	Giuseppe	Consenso Finale
Stakeholders	5/6	5/6	5/6	5/6	5/6	5/6	5/6
Opportunity	//	//	//	//	//	//	(non è un prodotto con soluzioni che hanno vincoli economici)
Software System	5/6	5/6	4/6	4/6	5/6	4/6	5/6
Requirements	5/6	6/6	5/6	5/6	5/6	5/6	5/6
Team	4/5	5/5	4/5	4/5	4/5	5/5	4/5
Work	5/6	5/6	5/6	5/6	5/6	5/6	5/6
Way of Working	5/6	5/6	5/6	5/6	5/6	5/6	5/6

Glad, Sad, Mad sembrava ridondante al team di sviluppo, in quanto le problematiche erano già state discusse durante il Progress Poker quindi è stato saltato,

A questo punto il team aveva pressoché terminato il progetto, quindi non si sono trovate cose da migliorare se non di terminare il progetto al 100%.

Release Sprint

Durante questo ultimo sprint il team di sviluppo si è impegnato a poter presentare un prodotto finito e funzionante.

Questo comprendeva:

- esecuzione di test mancanti o che davano risultati indesiderati
- bugfixing delle interfacce
- aumentare la copertura di sonarqube
- risoluzione di codesmells

Il progetto finite queste miglirie si poteva dire concluso ufficialmente
La retrospectiva di questo sprint viene presentata nella sezione successiva.

Descrizione del processo seguito

Per l'organizzazione del lavoro abbiamo seguito le guide di Scrum³. Il progetto è stato suddiviso in sprints, risultanti in 6 sprints totali: uno sprint iniziale (chiamato sprint 0), 4 sprint della durata di due settimane l'uno di lavoro e quello finale, dedicato al rilascio e alla levigazione, durato una settimana.

³ <https://www.scrum.org/>

Autodescrizione del Team

Il team è formato da 6 membri, ci conoscevamo già quasi tutti dal vivo e pertanto abbiamo utilizzato Trello rapidamente per inserirci come gruppo e ricercare i membri mancanti.

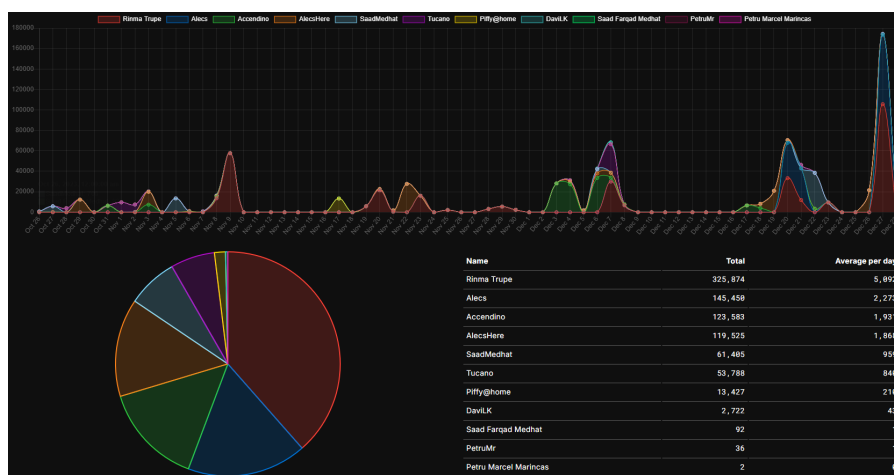
Risultato del team building

Per l'attività di team building abbiamo giocato ad una sessione piuttosto lunga di Scrumble, dal vivo, ed abbiamo provato Escape the boom, il quale tuttavia non funzionava. Scrumble ci è stato molto utile per capire le basi dei processi basati su Scrum, concetti come lo Sprint, i suoi vari step, il debito tecnico, le US. In generale è stata un'esperienza piuttosto formativa, anche se ci ha richiesto parecchio tempo prima di entrare nel meccanismo.⁴

Sintesi dei dati del logger e GitStats

Il codice fatto è visibile su GitLab, in particolare dentro la cartella "code". All'interno di questa cartella vi sono due sezioni: quella dedicata al "backend", ovvero che contiene il nostro web server scritto con Node.js e che usa Express, e quella dedicata al "frontend", ovvero che contiene le pagine web della nostra web application: in particolare, si tratta di codice React che, una volta compilato, finisce tutto dentro la cartella "dist", la quale conterrà l'effettiva pagina web che verrà servita.⁵ Abbiamo inoltre altro codice, quello presente su "test", all'interno del quale usiamo Jest e Cypress per eseguire rispettivamente test del backend e testing E2E.

Abbiamo utilizzato GitStats per controllare le statistiche relative al lavoro del gruppo. Il risultato però esce non pienamente veritiero poichè i membri hanno lavorato in sottogruppi utilizzando sessioni condivise di Visual Studio Code:



⁴ Più dettagli sulle relazioni di teambuilding si possono trovare sulla repository di GitLab, in particolare in "doc/teambuilding", dove vi sono le relazioni relative.

⁵ Più dettagli sul codice presente in "code", ovvero per esempio come si può runnare, dettagli sul suo contenuto, etc. si possono trovare in "code/README.md"

Strumenti di comunicazione

Gli strumenti utilizzati per la comunicazione sono stati WhatsApp e Discord. Tutti i membri del team di sviluppo sono abituati ad un utilizzo quotidiano di queste applicazioni, pertanto abbiamo optato per questo modo di comunicazione piuttosto che Mattermost in modo da non dover aggiungere dell'impegno supplementare, che non riscuote alcuna utilità pratica, per comunicare tra di noi.

In modo asincrono su una chat di gruppo su WhatsApp si comunicavano problemi reputati più semplici e più veloci da risolvere. Lo Scrum Master inoltre, seguendo una tabella Excel ogni settimana riempita dai membri con i propri impegni e delle votazioni avvenute con l'utilizzo dello strumento Poll di WhatsApp, indicava data e orario di riunioni e Daily Scrums sull'ultima piattaforma descritta.

Discord veniva utilizzato per incontri sincroni, come le Daily Scrums. Inoltre in seguito alla retrospettiva del secondo sprint il team ha deciso di utilizzarlo più frequentemente anche per fare sessioni di lavoro che comprendevano più membri, in seguito ad aver notato una scarsa efficienza di lavoro nel suddetto sprint.

ChatGPT

ChatGPT è stato molto utile durante il progetto soprattutto per aiutarci con i giochi proposti: ad esempio durante lo Sprint 0 ha aiutato a simulare il gioco Scrumble creando delle User Stories adeguate.

Durante la creazione delle interfacce il team di sviluppo ha deciso di utilizzare ChatGPT per aiutarsi con la creazione di componenti logicamente semplici ma tediosi da scrivere, per potersi focalizzare sulla scrittura di codice più elaborato.

In particolare molti elementi di stile per bottoni o forms sono stati creati utilizzando ChatGPT.

1. Create a container for your popup and the "X" button:

```
jsx
import React from 'react';

const Popup = () => {
  return (
    <div style={{ position: 'relative', width: '300px', height: '200px' }}
      /* Popup content */
      <div style={{ backgroundColor: 'white', padding: '16px', position:
        <h2>Popup Content</h2>
        <p>This is the content of the popup.</p>
      </div>

      /* X button */
      <button
        style={{
          position: 'absolute',
          top: '8px',
          right: '8px',
          padding: '4px',
          backgroundColor: 'red',
          color: 'white',
          border: 'none',
          cursor: 'pointer',
        }}
        onClick={() => {
          // Handle close button click
        }}
      >
        X
      </button>
    </div>
  );
};

export default Popup;
```

Retrospektiva finale

La retrospektiva è stata guidata dal gioco del Progress Poker
Come per la scorsa retrospective, abbiamo utilizzato le carte Essence⁶ relative a questi punti per dare le valutazioni. Pertanto nella tabella viene trascritto il valore numerico tra parentesi (genericamente compreso tra 1 e 6, ma a volte i valori cambiano. Ogni valore ha una check list relativa ed un significato importante, fare riferimento alle carte Essence⁷).

Carta \ Membro	Petru	Davide	Saad	Alex	Rafid	Giuseppe	Consenso Finale
Stakeholders	6/6	6/6	6/6	6/6	6/6	6/6	6/6
Opportunity	//	//	//	//	//	//	(non è un prodotto con soluzioni che hanno vincoli economici)
Software System	6/6	6/6	6/6	6/6	6/6	6/6	6/6
Requirements	6/6	6/6	6/6	6/6	6/6	6/6	6/6
Team	5/5	5/5	5/5	5/5	5/5	5/5	5/5
Work	6/6	6/6	6/6	6/6	6/6	6/6	6/6

⁶ Guida per essence: [Essence pocket guide](#)

⁷ Link carte essence: [Essence Kernel - Detail Cards \(Checklist\)](#), [Alpha State Cards \(PDF Version\)](#) | [Ivar Jacobson International](#)

Way of Working	6/6	6/6	6/6	6/6	6/6	6/6	6/6
----------------	-----	-----	-----	-----	-----	-----	-----

I commenti alle sezioni risultano superflui in quanto tutti i membri del team
Le conclusioni della retrospettiva vedono un progetto portato al termine in questo sprint.
I test sono stati eseguiti correttamente ed è stato implementato SonarQube come richiesto.

Sonarqube

Il nostro report finale di SonarQube è il seguente:

☆ [T3 ChessCake](#) PUBLIC

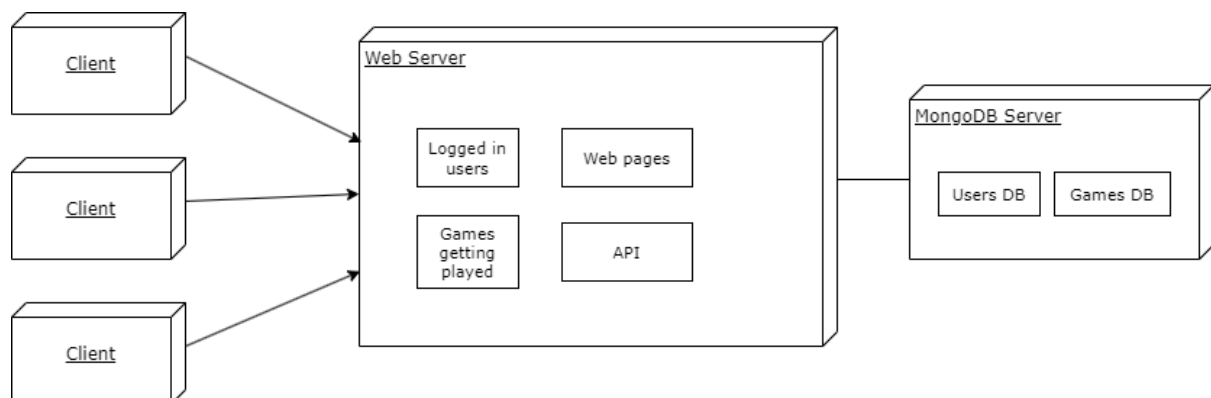
Last analysis: 12 hours ago • 2.3k Lines of Code • JavaScript



In particolare, abbiamo un coverage che supera l'80%, come richiesto dagli stakeholder, un basso numero di bug e vulnerabilità, abbiamo recensito tutti gli hotspot di sicurezza (dove in particolare vi sono molti errori dovuti all'uso di metodi pseudo casuali, che nel nostro caso è motivato in quanto vogliamo generare board con seed) ed abbiamo un numero relativamente alto di duplicazioni, ma in gran parte questo è motivato dalla creazione di Kriegspiel, il quale riusa molti dei metodi già creati per Really Bad Chess con poche modifiche.

Diagramma di Deployment

Il nostro diagramma di deployment ha pressoché il seguente aspetto:



Ovvero, un insieme di client che si connettono al nostro Web server, il quale contiene diverse informazioni salvate in memoria e può restituire dei messaggi (ovvero Web Pages o le risposte alle API calls). Inoltre, per farlo, comunica con MongoDB.

Demo

La demo è possibile visualizzarla al seguente link: <https://youtu.be/pk0GFAa2PIE>.
Si tratta di un breve video della durata di 3 minuti dove il PO presenta le feature principali dell'applicazione.

Lista e descrizione degli artefatti

Il nostro progetto ha generato i seguenti artefatti (tutti i link ad “aminsep” richiedono inviti come autorizzazioni.):

- Le User Stories, Epiche e i vari task si possono trovare su **Taiga**, in particolare sulla versione installata su <https://aminsep.disi.unibo.it/>, ai quali i professori del corso hanno accesso.
- I report generati da SonarQube si trovano anch'essi su <https://aminsep.disi.unibo.it/>. Sono disponibili in particolare a questo indirizzo:
https://aminsep.disi.unibo.it/sonarqube/dashboard?id=Saad2001_chesscake_AYwQGiKoiHTP7memjMGc
- Il database non ha dati particolarmente importanti, pertanto non lo aggiungiamo in questa lista. L'unica cosa di nota è l'utenza “Computer” che viene generata alla creazione del DB.
- Il progetto stesso, ovvero la repository di lavoro, la documentazione (di ogni sprint, che di solito include il burndown chart, una presentazione, i goals e le dichiarazioni di ready and done e le retrospective, oltretutto un link alle review) si trovano sul GitLab messo a disposizione dall'università di Bologna, in particolare si trova a questo link dal quale si può clonare: <https://aminsep.disi.unibo.it/gitlab/Saad2001/chesscake.git>
- Le sprint review si trovano ai seguenti link (sono ordinate dalla sprint 1 alla sprint 4):
<https://youtu.be/7x3iO-WZsS8>, <https://youtu.be/PTCtjgOLF9A>,
<https://youtu.be/wcSxxd1fZTE>, <https://youtu.be/gttxOq1QJ50>
- Il video della demo si può trovare a questo link:

Link prodotto

Il prodotto live si può trovare al seguente link: <https://site222341.tw.cs.unibo.it/>.
Per un'esperienza più completa, consigliamo vivamente la creazione di un account.