

Rapporto del team 8

Patrick Alfieri - 0001030013 Giulia Torsani - 0001020194
Kaori Jiang - 0001019903 Davide Luccioli - 0001028403
Sofia Zanelli - 0001019690

T8 SilverBullets

Indice

1	Introduzione	3
2	Descrizione del prodotto da costruire	3
2.1	Scope	3
2.2	Backlog di prodotto	3
2.3	Diagramma dei casi d'uso	4
2.4	Diagramma architetturale iniziale	4
2.5	Librerie adottate	5
3	Analisi degli Sprint	5
3.1	Sprint 0	5
3.1.1	Sprint Goal	5
3.1.2	Sprint Backlog	6
3.1.3	Descrizione del codice prodotto	6
3.1.4	Definition of Done	6
3.2	Sprint 1	7
3.2.1	Sprint Goal	7
3.2.2	Sprint Backlog - Inizio Sprint	7
3.2.3	Descrizione del codice prodotto	7
3.2.4	Definition of Ready	7
3.2.5	Definition of Done	7
3.2.6	Sprint Backlog - Fine Sprint	8
3.2.7	Test effettuati sulle User Stories	8
3.2.8	Burndown Chart	9
3.2.9	Retrospettiva con Essence	9
3.2.10	Sprint Review	9
3.3	Sprint 2	10
3.3.1	Sprint Goal	10
3.3.2	Sprint Backlog - Inizio Sprint	10
3.3.3	Descrizione del codice prodotto	10
3.3.4	Definition of Ready	10
3.3.5	Definition of Done	10
3.3.6	Sprint Backlog - Fine Sprint	11
3.3.7	Test effettuati sulle User Stories	12
3.3.8	Burndown Chart	13
3.3.9	Retrospettiva con Essence	13
3.3.10	Sprint Review	13
3.4	Sprint 3	14
3.4.1	Sprint Goal	14
3.4.2	Sprint Backlog - Inizio Sprint	14
3.4.3	Descrizione del codice prodotto	14
3.4.4	Definition of Ready	14
3.4.5	Definition of Done	14
3.4.6	Sprint Backlog - Fine Sprint	15
3.4.7	Test effettuati sulle User Stories	15
3.4.8	Burndown Chart	16
3.4.9	Retrospettiva con Essence	17
3.4.10	Sprint Review	17
3.5	Sprint 4	17
3.5.1	Sprint Goal	17
3.5.2	Sprint Backlog - Inizio Sprint	17
3.5.3	Descrizione del codice prodotto	17
3.5.4	Definition of Ready	17
3.5.5	Definition of Done	18
3.5.6	Test effettuati sulle User Stories	18
3.5.7	Retrospettiva con Essence	18

4 Descrizione del processo seguito	19
4.1 Autodescrizione del Team	19
4.1.1 Teambuilding	19
4.1.2 Ruoli nel team	19
4.2 Risultato del teambuilding	19
4.2.1 Scrumble	19
4.2.2 Escape The Boom	20
4.3 Analisi dati gitinspector	20
4.4 Strumenti di comunicazione	21
4.4.1 Pregi di Matrix	21
4.4.2 Difetti di Matrix	21
4.5 Uso di ChatGPT / LLM	22
4.6 Retrospettiva finale in Essence	24
4.7 Test E2E	24
4.8 Schema architetturale finale	25
4.9 Schermata finale di Sonarqube	26
4.10 Diagramma di Deployment	27
4.11 Demo del prodotto	27
4.12 Lista degli artefatti in consegna e modalità di accesso	27
4.13 Link a prodotto live	27
5 Conclusioni	28

1 Introduzione

Il presente documento segna il culmine del nostro impegno nel progetto di **Ingegneria del Software**, CdL informatica. Questa esperienza é stata forgiante sotto molteplici aspetti, arricchendo le nostre competenze e offrendoci un'opportunità unica per applicare i concetti teorici appresi in un progetto pratico e significativo.

Sono state messe alla prova la nostra capacità di adattamento, di collaborazione e più in generale di problem solving. La composizione di un prodotto valido e funzionale è stato fin dal primo giorno il nostro obiettivo e oggi lo presentiamo.

2 Descrizione del prodotto da costruire

2.1 Scope

L'obiettivo del progetto consiste nel realizzare un'applicazione web accessibile tramite internet per giocare a **scacchi eterodossi**, in particolare alla variante **Reconnassaince Blind Chess** (abbreviato **ReconChess**), che nasce dal **JHUAPL** come oggetto di ricerca in AI.

Gli utenti hanno la possibilità di scegliere se cimentarsi contro il computer a difficoltà variabile, oppure in modalità online attraverso una stanza di gioco, dove é possibile invitare altri utenti condividendo il codice associato univoco, eventualmente attraverso i Social Network (es. X).

I giocatori che si sfidano potranno competere globalmente su una Leaderboard, aggiornata ad ogni vittoria/sconfitta utilizzando come metrica di ranking il punteggio ELO.

É possibile usufruire dell'applicazione sia registrandosi e accedendo con le proprie credenziali, sia navigando come guest, ma in tal caso alcuni servizi risulteranno essere non disponibili.

La registrazione può avvenire anche attraverso account esterni (es. Google).

2.2 Backlog di prodotto



Figura 1: Backlog di prodotto finale.

2.3 Diagramma dei casi d'uso

Un diagramma dei casi d'uso è uno strumento visivo utilizzato per mostrare le interazioni tra gli utizzatori dell'applicazione e il sistema stesso.

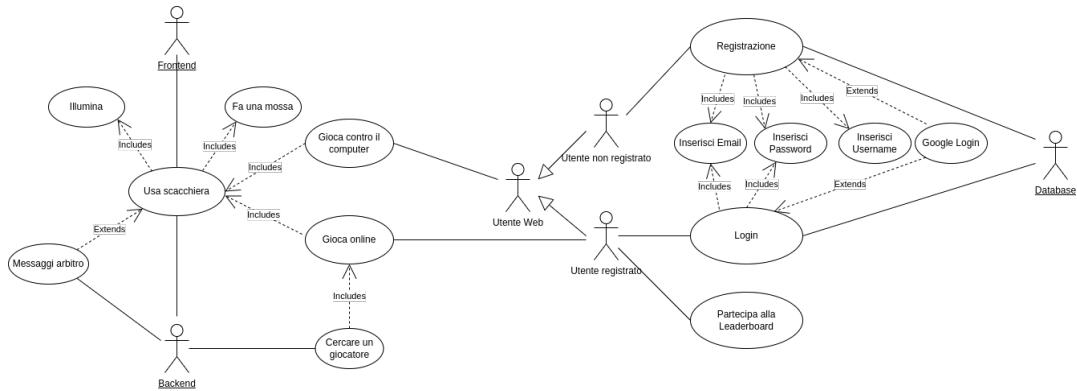


Figura 2: Use Case Diagram per un'applicazione web scacchistica.

nota. Gli elementi sottolineati rappresentano le componenti logiche dell'applicazione.

2.4 Diagramma architetturale iniziale

Lo schema architettonico vuole fornire una panoramica intuitiva della struttura del software, collocando e dividendo i moduli in base alle funzionalità e mostrando come essi interagiscono.

Si noti come molte delle scelte progettuali riportate in questo grafico hanno pesantemente inciso sullo sviluppo, assumendo un ruolo di guida ed evidenziando punti di forza e talvolta i necessari sforzi impiegati durante l'implementazione.

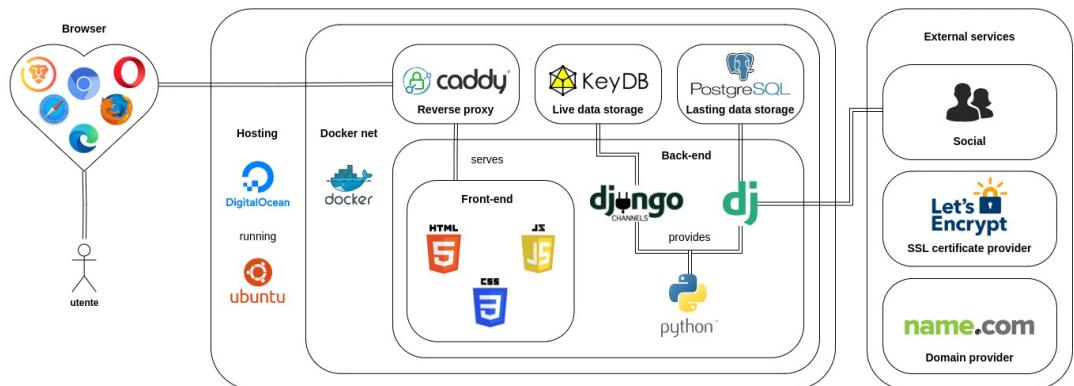


Figura 3: Schema architettonico dell'applicazione.

Descriviamo quindi le componenti principali e l'uso, motivando le decisioni intraprese:

- **HTML, CSS, JavaScript.** Fondamentali del lato client: HTML struttura il contenuto delle pagine, CSS ne definisce l'aspetto e JavaScript aggiunge dinamicità e interattività. Non è stato adottato nessun framework JavaScript (*es. React*) in quanto ritenuti poco utili nei confronti dei vincoli e requisiti di prodotto, rischiando che risultasse in uno sforzo corrispondente ad una resa relativamente bassa.
- **Caddy.** Un web server open source che fa della semplicità di configurazione il suo punto forte. Nell'applicazione, il suo scopo è di fungere da **reverse proxy**, reindirizzando il traffico che riceve ai vari servizi. La scelta rispetto alle alternative è dovuta in gran parte ai certificati di sicurezza SSL/TLS, già forniti in automatico dalla Certification Authority **Let's Encrypt**.
- **Python.** Il lato server è stato realizzato in python, linguaggio di programmazione versatile con notevole materiale di documentazione specifica, principale motivazione d'utilizzo.

L'architettura è realizzata attraverso contenitori Docker.

2.5 Librerie adottate

Con librerie si intendono le risorse software esterne che contribuiscono al funzionamento dell'applicazione. Ogni inclusione delle singole nel prodotto è stata preceduta da uno studio di fattibilità, non solo mirato a verificare utilizzi specifici, ma soprattutto licenze di distribuzione.

Di seguito le principali:

Chess.js - Libreria JavaScript che implementa le regole degli scacchi. Permette la validazione e generazione di mosse, ma nativamente solo sulla versione classica. Licenza: BSD 2-CLAUSE.

ChessboardJS - Libreria JavaScript che visualizza una scacchiera integrabile, permettendo di aggiornarla attraverso le mosse dei pezzi. Licenza: MIT.

JQuery - Libreria JavaScript semplifica la manipolazione del DOM, utile in particolare per i selettori sugli elementi della scacchiera. Licenza: MIT.

JQuery-ui - Libreria JavaScript che fornisce interfacce utente basate su jQuery. Utile per i selettori nei confronti dei pezzi della scacchiera. Licenza: MIT.

Django - Framework per lo sviluppo web python. Fornisce molteplici servizi lato amministrazione e database. Licenza: BSD 3-CLAUSE.

DjangoChannels - Estensione del framework che aggiunge la possibilità di creare websocket e richieste http asincrone. Licenza: BSD 3-CLAUSE.

Psycopg2 - Adattatore di database PostgreSQL per python. Funge da wrapper per il protocollo previsto dal database. Licenza: LPGL.

Python-chess - Libreria python per la logica degli scacchi classici, usata in particolare nei confronti dei bot. Licenza: GPL 3.0.

Reconchess - Pacchetto di API che include logica specifica per la variante eterodossa Reconnaissance Blind Chess, oltre che un arbitro specifico. Licenza: BSWD 3-CLAUSE.

3 Analisi degli Sprint

Il progetto è stato portato a termine secondo i principi agili, in particolare attraverso il modello **Scrum**. Questo paradigma orientato al project management divide l'intero sviluppo in una serie di iterazioni, dette Sprint, di durata compresa fra 2-4 settimane.

Andiamo quindi ora ad analizzare gli artefatti di ognuna delle fasi.

3.1 Sprint 0

Lo sprint in questione è iniziato in data 12/10/23 e si è concluso in data 26/10/23, per una durata totale di due settimane. nota. In questo sprint non sono state prodotte informazioni riguardanti Retrospettiva e Burndown.

3.1.1 Sprint Goal

Lo Sprint 0 è un'iterazione preliminare che rappresenta lo starting point del processo. Pertanto, si collocano come obiettivi una corretta analisi dei requisiti e le conseguenti operazioni di natura infrastrutturale. Il goal in questione è quindi definito dalle seguenti:

- Attività di teambuilding (Scrumble, ETB, Trello).
- Setup dell'ambiente di lavoro.
- Definizione del Product Backlog in vista dei futuri Sprint.
- Definizione dell'Architettura del prodotto.

3.1.2 Sprint Backlog

Durante questo Sprint non è stato prodotto uno Sprint Backlog concreto, dato che nessuna delle attività descritte nello Sprint Goal ricadeva nella definizione di User Stories, che devono necessariamente soddisfare un qualche tipo di bisogno atomico nei confronti di chi usufruirà dell'applicazione.

Di seguito vengono quindi riportate le storyless stories, senza stima associata, (*o task*) svolte:

- Registrazione dei membri del team e setup ambiente CAS (GitLab, Taiga).
- Hosting del server tramite droplet DigitalOcean in cloud.
- Acquisizione dominio su name.com.
- Configurazione server Matrix per comunicazione interna
- Mockup visivi dell'applicazione.

3.1.3 Descrizione del codice prodotto

Non è stato prodotto codice particolarmente rilevante, oltre che i file di configurazione specifici per il reverse proxy Caddy e per i primi servizi dell'applicazione containerizzati su Docker, come ad esempio il database PostgreSQL.

Segue un esempio di un Caddyfile minimale rispetto ad alternative come Nginx:

```
1 # Caddyfile per il server Caddy
2 # Configurazione del dominio con TLS/SSL automatico
3 example.com
4
5 # Configurazione del proxy per il server backend
6 reverse_proxy localhost:8000
7
8 # Configurazione principale per il server Nginx
9
10 # Server block per il dominio con TLS/SSL
11 server {
12     listen 443 ssl;
13     server_name example.com;
14
15     ssl_certificate /etc/nginx/ssl/example.com.crt;
16     ssl_certificate_key /etc/nginx/ssl/example.com.key;
17
18     location / {
19         proxy_pass http://localhost:8080;
20     }
21 }
```

3.1.4 Definition of Done

La DoD descrive le metriche di qualità richieste che le singole User Stories devono soddisfare per poter essere considerate "Done" e generare quindi un risultato tangibile.

Non essendo presenti User Stories, non risulta in questo caso fondamentale.

3.2 Sprint 1

Lo sprint in questione è iniziato in data 27/10/23 e si è concluso in data 9/11/23, per una durata totale di due settimane.

3.2.1 Sprint Goal

- Realizzare un'interfaccia intuitiva per una web app che permetta di giocare a scacchi online.
- Mostrare una scacchiera e i pezzi su essa disposti.
- Permettere ai pezzi solo le mosse consentite dalle regole degli scacchi ortodossi.
- Gestire attraverso un sistema di login e registrazione sul sito, i dati degli utenti.

3.2.2 Sprint Backlog - Inizio Sprint

- Come utente, vorrei poter visualizzare una scacchiera intuitiva e accessibile, per giocare a scacchi sul web. **Stima 19**.
- Come utente, desidero poter effettuare una mossa in una partita di scacchi, in modo da poter giocare. **Stima 10**.
- Come utente non registrato, voglio poter usufruire di un interfaccia di registrazione per accedere al sito con le mie credenziali, per poter salvare i miei progressi di gioco. **Stima 15**.
- Come amministratore, voglio avere un elenco accessibile e aggiornato dei giocatori che si sono registrati ed hanno svolto l'accesso, per monitorarli. **Stima 20**.

3.2.3 Descrizione del codice prodotto

Il focus di questo Sprint è stato principalmente lato front-end: sono state create le pagine principali e i loro fogli di stile, oltre che incluse e sfruttate le librerie necessarie per poter gestire una partita di scacchi classici, ovvero chess.js e chessboardjs. Inoltre, si sono realizzate funzioni specifiche JavaScript che permettono la registrazione degli utenti sul server attraverso i form appropriati. La registrazione avviene nel database PostgreSQL attraverso Django e il protocollo websocket Daphne, sebbene per il momento gli utenti registrati non godano di funzionalità esclusive.

Infine, modificando le proprietà della scacchiera, sono iniziate alcune implementazioni in vista degli sprint futuri, aggiungendo la nebbia e modificando alcune proprietà dei pedoni in riferimento alla specifica variante ReconChess.

3.2.4 Definition of Ready

La DoR stabilisce i criteri specifici per poter includere una User Story all'interno della pianificazione dello sprint. È importante in quanto User Story non pienamente mature e comprese rischiano di rallentare lo sviluppo e più in generale portare incomprensioni all'interno del team. Per essere "Ready", riteniamo che una storia debba essere:

- Ben Definita, oltre che compresa e accettata da tutto il team.
- Deve essere stabilita la sua priorità rispetto al backlog dal Product Owner, mentre deve essere stimata in punti storia dal team.
- Chi porta a termine la User Story, generando un incremento del prodotto, deve essere definito.
- Deve essere correttamente testabile (*non necessariamente testata!*), ovvero deve essere possibile adottare TDD come tecnica di sviluppo (Test Driven Development).

3.2.5 Definition of Done

Affiché una User Story venga considerata "Done" dovrebbe avere le seguenti caratteristiche:

- Tutto il codice di implementazione ed essa associato è stato scritto e revisionato da almeno un altro membro del team, a prescindere dal ruolo che ricopre nella struttura Scrum.
- Il codice è stato integrato con successo con il ramo principale della repository GitLab e passato tutti i test di integrazione.

- Il lavoro svolto è stato presentato al Product Owner nelle sedi opportune ed è stato compreso, valutato ed infine accettato.
- Ogni membro del team non direttamente coinvolto nell'implementazione di una User Stories, ha comunque fornito feedback esterni.

3.2.6 Sprint Backlog - Fine Sprint

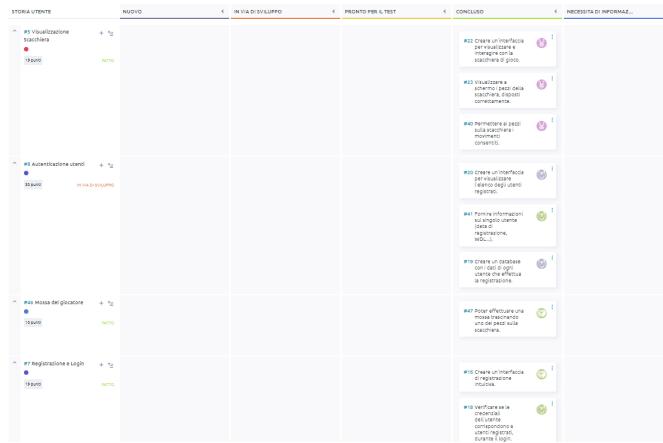


Figura 4: Sprint Backlog sprint 1, con US e task.

3.2.7 Test effettuati sulle User Stories

”Come utente, desidero poter effettuare una mossa di scacchi, in modo da poter giocare ”:

```

1 // Test per mosse Frontend. Testato attraverso Jest.
2
3 describe('makeMove', () => {
4   let game;
5   let valid_moves;
6   let config;
7
8   beforeEach(() => {
9     game = {
10       get: jest.fn(),
11       remove: jest.fn(),
12       put: jest.fn()
13     };
14     valid_moves = ['e2e4', 'e7e5'];
15     config = {
16       draggable: true
17     };
18   });
19
20 // Test case 1: When the move is legal and no promotion
21 it('should make the move', () => {
22   // Set up initial values
23   const move_cfg = {
24     from: 'e2',
25     to: 'e4'
26   };
27
28   // Call the function
29   makeMove(game, move_cfg, false);
30
31   // Assert the expected results
32   expect(game.get).toHaveBeenCalledWith('e2');
33   expect(game.remove).toHaveBeenCalledWith('e2');
34   expect(game.put).toHaveBeenCalledWith(piece, 'e4');
35   expect(config.draggable).toBe(false);
36 });
37 });

```

"Come utente non registrato, voglio poter usufruire di un interfaccia di registrazione per accedere al sito con le mie credenziali, per poter salvare i miei progressi di gioco."

```

1  ## Test per registrazione utenti. Testato attraverso unittest.
2
3  def test_register_view_post(self):
4      client = Client()
5
6      # Simulate a POST request to the register view
7      response = client.post(reverse('register'), data=self.test_user_data)
8
9      # Check if the response status code is 200 (OK)
10     self.assertEqual(response.status_code, 200)
11
12     # Check if the user was created successfully
13     self.assertEqual(User.objects.filter(username='testuser').exists(), True)

```

Il resto delle US é stato testato attraverso test E2E, seguendo lo schema:

- Visualizzazione scacchiera → **Obiettivo**. Verificare che la scacchiera venga mostrata nella pagina apposita. **Azione**. Utilizzare un browser per aprire la suddetta pagina. **Verifica**. La scacchiera risulta interagibile.
- Elenco dei giocatori → **Obiettivo**. Verificare che la pagina admin di django presenti gli iscritti. **Azione**. Andare nella sezione amministrazione. **Verifica**. Effettuare una registrazione utente sul momento e verificare il risultato.

3.2.8 Burndown Chart

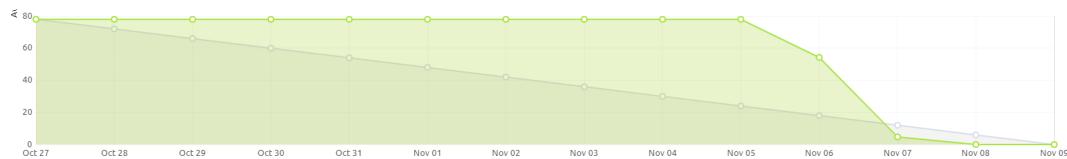


Figura 5: Burndown Chart Sprint 1.

3.2.9 Retrospectiva con Essence

RETROSPECTIVE		GOOD		OK		NOT GOOD	
HIGH		Stakeholders Hosted ✓ Representatives assist the team ✓ Design and detailed provided ✓ Changes promptly communicated		Software System Architecture Selected ✓ Architecture selection criteria ✓ API platform identified ✓ System boundary known ✓ System requirements known ✓ System architecture known ✓ Built, review, iteration made ✓ Key technical risks agreed to		Way of Working Principles Established ✓ Team actively work principles ✓ Team members agree with principles ✓ Team members understand principles ✓ Operational control understood ✓ Practice A/E consistency known	
MEDIUM		Team Collaborating ✓ Works as one unit ✓ Communicates and honest ✓ Focused in mission ✓ Motivates one another		Requirements Bounded ✓ Requirements established ✓ Requirements clear ✓ Requirements traceable ✓ Requirements format agreed ✓ Requirements scope clear ✓ Requirements identified & prioritized ✓ Requirements agreed to		Opportunity Value Established ✓ Opportunity value quantified ✓ Opportunity value communicated ✓ System value understood ✓ Opportunity scope clear ✓ Outcome clear & quantified	
LOW		Work Concluded ✓ Only actions taken ✓ Results achieved ✓ Learning system adopted					

Figura 6: Risultato della retrospettiva sprint 1 in Essence.

L'immagine rappresenta le conclusioni a cui il team é giunto a seguito della Sprint Retrospective, concettualizzata attraverso carte Essence, in particolare Alphas. La divisione verticale rappresenta la priorità concordata dal team rispetto ad un certo ambito, visualizzato attraverso le diverse carte, mentre la divisione orizzontale vuole mostrare la metrica associata alle pratiche, corrispondenti alle valutazioni Good, To Improve e Not Good della Retrospettiva. Per informazioni più dettagliate, si consulti la Sprint Retrospective associata allo Sprint 1.

3.2.10 Sprint Review

Link review Sprint 1: [Review Sprint 1](#).

3.3 Sprint 2

Lo sprint in questione è iniziato in data 9/11/23 e si è concluso in data 22/11/23, per una durata totale di due settimane.

3.3.1 Sprint Goal

- Perfezionamento dell'implementazione della variante ReconChess (mosse e sensing).
- Inserimento timer nel contesto delle regole della variante.
- Integrazione dell'arbitro nella partita.

3.3.2 Sprint Backlog - Inizio Sprint

- Come utente, voglio un timer per le mie partite di scacchi, in modo da poter giocare in modo celere. **Stima 16**.
- Come utente, voglio poter visualizzare le peculiarità di ReconChess sulla scacchiera, per capire come giocare ad una nuova variante di scacchi. **Stima 18.5**.
- Come utente, voglio poter ricevere comunicazioni da parte di un arbitro ReconChess, per comprendere mosse legali e illegali. **Stima 19**.
- Come utente, voglio poter effettuare un'operazione di sensing su una scacchiera oscurata, per vedere correttamente i pezzi avversari e poter elaborare strategie. **Stima 19**.
- Come utente, voglio visualizzare le regole di gioco della variante ReconChess, per poterle consultare all'occorrenza. **Stima 6**.

3.3.3 Descrizione del codice prodotto

In questo sprint, il team si è concentrato per implementare le funzionalità caratteristiche della variante, in particolare la **Fog** che genera una conoscenza iniziale incompleta della scacchiera, rappresentata dall'oscuramento delle caselle, e l'operazione di **sensing**, che permette di illuminare un'area 3x3 a scelta dell'utente, diradando la fog. Questo è stato possibile tramite i selettori offerti da JQuery e l'uso di CSS dinamico.

Lato backend, si è realizzata la logica del gioco attraverso le API ReconChess in python. Le informazioni sullo stato della partita prodotte, come messaggio, posizione dei pezzi sulla scacchiera e timer vengono inviate al client attraverso una WebSocket dedicata. Tali dati vengono poi usati per aggiornare lo stato della board lato client attraverso i metodi offerti da chessboardjs.

È stato inoltre implementato Trout, un bot open source specifico per la variante ReconChess che fa uso del motore di scacchi Stockfish per eseguire le sue mosse.

3.3.4 Definition of Ready

Non ci sono modifiche rispetto alla DoR presentata nello Sprint 1.

3.3.5 Definition of Done

Affiché una User Story venga considerata "Done" dovrebbe avere le seguenti caratteristiche:

- Tutto il codice di implementazione ad essa associato è stato scritto e revisionato da almeno un altro membro del team, a prescindere dal ruolo che ricopre nella struttura Scrum.
- Il lavoro svolto è stato presentato al Product Owner nelle sedi opportune ed è stato compreso, valutato ed infine accettato.
- Ogni membro del team non direttamente coinvolto nell'implementazione di una User Stories, ha comunque fornito feedback esterni.

Si vuole far notare come sia stata rimossa la condizione "Il codice è stato integrato con successo con il ramo principale della repository GitLab e passato tutti i test di integrazione". Le ragioni di questa scelta sono da ritrovarsi nel fatto che alcune delle funzionalità previste dalle User Stories risultavano essere bloccanti nei confronti di altre (ad es. implementazione ReconChess e conseguente arbitro), dunque dover adattare il ramo principale in vista di features future, sebbene possibile, rischiava di richiedere uno sforzo poco remunerativo, durante questo sprint.

3.3.6 Sprint Backlog - Fine Sprint

STORIA UTENTE	NUOVO	IN VIA DI SVILUPPO	PRONTO PER IL TEST	CONCLUSO
^ #49 Timer di gioco 16 punti	+ * FATTO			#52 Causare sconfitta quando giunge il time-up. #50 Stampare tempo rimasto al giocatore per il turno. #51 Sincronizzazione timer backend-frontend.
^ #58 Funzione di nebbia 18,5 punti	+ * FATTO			#59 Nascondere la parte avversaria della scacchiera. #60 Nascondere i pezzi della scacchiera avversari.
^ #61 Arbitro 19 punti	+ * FATTO			#63 Comunicare il turno. #62 Comunicare il sensing di turno (a chi spetta). #64 Comunicare che è avvenuta una cattura.

Figura 7: Sprint Backlog sprint 2, con US e task - 1.

STORIA UTENTE	NUOVO	IN VIA DI SVILUPPO	PRONTO PER IL TEST	CONCLUSO
				#64 Comunicare che è avvenuta una cattura. #65 Comunicare che è avvenuta una mossa illegale.
^ #66 Sensing 19 punti	+ * FATTO			#67 Selezionare zona da rivelare. #68 Illuminare zona selezionata.
^ #9 Regole di gioco 6 punti	+ * FATTO			#29 Inserimento delle regole di gioco. #30 Interfaccia per visualizzare le regole lato utente
^ Compiti senza storie	+ *			#69 Collegare frontend a giocatore software backend.

Figura 8: Sprint Backlog sprint 2, con US e task - 2.

3.3.7 Test effettuati sulle User Stories

”Come utente, voglio un timer per le mie partite di scacchi, per giocare in modo celere ”;

```

1 // Test per Timeout Frontend. Testato attraverso Jest.
2
3 import { JSDOM } from 'jsdom';
4
5 describe('updateTimer', () => {
6   beforeAll(() => {
7     dom.window.document.getElementById('timer').innerText = '00:00';
8   });
9   // Test case 1: When seconds is 0
10  it('should decrement minutes and set seconds to 59', () => {
11    // Set up initial values
12    const time = 120
13
14    // Call the function
15    updateTimer(time, dom.window.document.getElementById('timer'));
16
17    // Assert the expected results with to JSDOM Virtual DOM
18    expect(dom.window.document.getElementById('timer').innerText).toBe('01:59');
19  });
20  ... // Other tests
21});
```

”Come utente, voglio poter ricevere comunicazioni da parte di un arbitro ReconChess, per comprendere mosse legali e illegali. ”

```

1 ## Test per Socket Backend. Testato attraverso unittest.
2
3 async def connect(self):
4     communicator = WebsocketCommunicator(GameConsumer.as_asgi(), "/ws/game")
5     connected, _ = await communicator.connect()
6     self.assertTrue(connected)
7     return communicator
8
9 async def test_start(self):
10    communicator = await self.connect()
11    await communicator.send_json_to({
12        'action': 'start_game',
13        'color': 'white',
14        'bot': 'random'
15    })
16
17    self.assertEqual(await communicator.receive_json_from(), {
18        'message': 'game started',
19        'board': chess.STARTING_FEN,
20        'color': 'w',
21        'opponent_name': 'RandomBot',
22        'time': 900
23    })
24    await communicator.disconnect()
```

”Come utente, voglio poter visualizzare le peculiarità di ReconChess sulla scacchiera, per capire come giocare ad una nuova variante di scacchi.

```

1 ## Test per il pass turn, mossa peculiare della variante ReconChess. Testato
2     attraverso unittest.
3
4 async def test_pass(self):
5     communicator = await self.connect()
6     await communicator.send_json_to({
7         'action': 'start_game'
8     })
9     # await the game started message, it's not necessary to test it as it is tested
10    in test_start
11    await communicator.receive_json_from()
12    # await the sense message and send a pass action
13    await communicator.receive_json_from()
14    await communicator.send_json_to(({'action': 'pass'}))
15
16    response = await communicator.receive_json_from()
17    self.assertEqual(response['message'], 'turn ended')
18
19    board = chess.Board()
20    board.push(chess.Move.null())
```

```

19     self.assertEqual(await communicator.receive_json_from(), {
20         'message': 'move result',
21         'requested_move': 'None',
22         'taken_move': 'None',
23         'captured_opponent_piece': False,
24         'capture_square': 'None',
25         'board': board.fen()
26     })
27
28     await communicator.disconnect()

```

Il resto delle User Stories è stato testato attraverso test End2End globali piuttosto che test di unità, che analizziamo in seguito. Se ne può comunque presentare uno schema del procedimento, in riferimento alle US presentate nella sezione backlog:

- Visualizzare regole → **Obiettivo**. Verificare che le regole vengano mostrate nella apposita pagina. **Azione**. Utilizzare un browser per aprire la suddetta pagina. **Verifica**. La pagina risulta scorrevole e intuitiva.
- Operazione di sensing → **Obiettivo**. Verificare che il sensing avvenga su una determinata area della scacchiera. **Azione**. Porre il cursore su una casella e cliccare. **Verifica**. Deve essere illuminata l'area 3x3 attorno alla casella cliccata e devono mostrarsi i pezzi avversari.

3.3.8 Burndown Chart

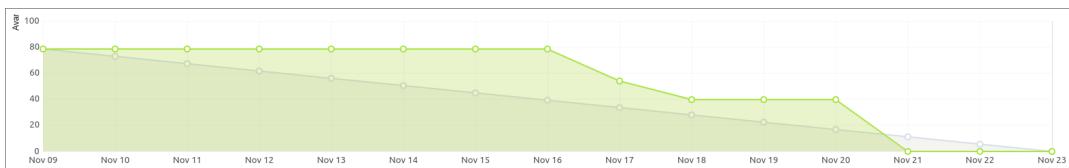


Figura 9: Burndown Chart Sprint 2.

3.3.9 Retrospectiva con Essence

RETROSPECTIVE	GOOD	OK	NOT GOOD
HIGH	Software System <ul style="list-style-type: none"> • System works well • System handles errors gracefully • System provides clear feedback • System integrates well with other components • System handles concurrent requests effectively • System provides clear documentation • System is highly available • System is reliable • System is secure • System is user-friendly Way of Working <ul style="list-style-type: none"> • Foundation Established • Requirements understood • Practices needed to do well • Practices refined over time • Non-negotiable practices & tools • Practices well understood • Practices well communicated • Practices well explained • Practices well explained • Practices well explained 	Team <ul style="list-style-type: none"> • Performing • Continuously adapting to change • Adapting to feedback • Novel and innovative solutions • Waste continuously eliminated Stakeholders <ul style="list-style-type: none"> • Involved • Represented the best • Freely feedback and decisions • Changes promptly communicated 	
MEDIUM		Requirements <ul style="list-style-type: none"> • Collected • Requirements clear • Requirements accurate • Requirements well defined • Requirements well prioritized • Priorities clear • Stakeholders involved • Stakeholders agrees on what to build Opportunity <ul style="list-style-type: none"> • Benefit Accrued • Solution accords benefits • ROI acceptable 	
LOW	Ro	Work <ul style="list-style-type: none"> • Consulted • Work items well defined • Work items well prioritized • Work items well assigned • Work items well tracked 	

Figura 10: Risultato della retrospettiva sprint 2 in Essence.

Per informazioni più dettagliate, si consulti la Sprint Retrospective associata allo Sprint 2.

3.3.10 Sprint Review

Link review Sprint 2: [Review Sprint 2](#).

3.4 Sprint 3

Lo sprint in questione è iniziato in data 23/11/23 e si è concluso in data 7/12/23, per una durata totale di due settimane.

3.4.1 Sprint Goal

- Implementazione multiplayer online.
- Scelta di bot differenti con cui giocare alla variante.
- Collegamento con social network.

3.4.2 Sprint Backlog - Inizio Sprint

- Come utente, voglio poter sfidare players online, per migliorare le mie abilità. **Stima 23.**
- Come utente, voglio usufruire di un social network, per condividere le mie giocate. **Stima 7.**
- Come utente, voglio salvare i risultati delle partite, per rivederli in seguito. **Stima 11.**
- Come utente, voglio poter sfidare diversi bot, per affinare le mie abilità. **Stima 10.**
- Come utente, voglio vedere le info. personali, per distinguermi da altri giocatori. **Stima 7.5.**
- Come utente, vorrei poter visualizzare una classifica globale, per comprendere la mia posizione attuale sul sito rispetto agli altri giocatori. **Stima 10.**

3.4.3 Descrizione del codice prodotto

L'implementazione del multiplayer è stata realizzata nel backend mediante una WebSocket condivisa tra i due giocatori e il server, che comunica le mosse a entrambi i client, aggiornando la scacchiera mano mano che il gioco procede. La stanza da gioco viene realizzata attraverso un codice univoco, che è possibile condividere attraverso i social grazie alle API fornite da essi. È possibile concordare timer, rematch e uscita di una partita attraverso gli appositi bottoni presenti nell'interfaccia, che chiamano opportunamente funzioni backend.

Sono stati inoltre implementati altri due bot ReconChess open source: Attacker e StrangeFish, che fa uso del motore di scacchi StockFish.

Lato front end, è stato rifinito l'aspetto dell'arbitro per permettere al giocatore (o ai giocatori) di comprendere meglio lo stato del gioco e quali azioni è necessario compiere, ad esempio sensing e movimento.

3.4.4 Definition of Ready

Non ci sono modifiche rispetto alla DoR presentata nello Sprint 2.

3.4.5 Definition of Done

Non ci sono modifiche rispetto alla DoD presentata nello Sprint 2.

3.4.6 Sprint Backlog - Fine Sprint

STORIA UTENTE	NUOVO	IN VIA DI SVILUPPO	PRONTO PER IL TEST	CONCLUSO	NECESSITA DI INFORMAZIONI
#11 Sfidare Altri Utenti + * 1 punto	PRESENTE			#31 Possibilità di interrompere la partita. #32 Creazione stanza di gioco multiplayer. #33 Implementazione dell'abilità di rematch.	
#14 Collegamento a social + * 7 punti	PRESENTE	#79 Condividere il codice stanza attraverso i social	#26 Inserimento nel database dei risultati delle partite dell'utente	#71 Login attraverso social (Google, X...)	
#15 Salvataggio partite + * 11 punti	PRESENTE		#27 Visualizzazione dei risultati delle partite del sense		
#10 Giocatore Software + * 10 punti	NUOVO			#32 Implementazione di un giocatore IA #33 Possibilità di selezionare il bot con cui giocare.	
#53 Informazioni utenti + * 7 punti	PRESENTE			#33 Inviare propPic a frontend. #34 Inviare nome utente a frontend. #35 Mostrare informazioni ricevute da backend. #36 Richiedere informazioni utente a backend.	
#12 Leaderboard + * 10 punti	PRESENTE			#37 Creare un'interfaccia per visualizzare la classifica dei giocatori-VS-ID	

Figura 11: Sprint Backlog sprint 3, con US e task.

Si fa notare come non tutte le task (*quindi User Stories annesse*) siano state portate a termine in tempo nei confronti dello Sprint, generando debito tecnico. In particolare, mancano la gestione del salvataggio delle partite e la condivisione del codice stanza attraverso i social network.

L'obiettivo è quello di recuperarlo prontamente nello sprint successivo.

3.4.7 Test effettuati sulle User Stories

"Come utente, voglio poter sfidare players online, per migliorare le mie abilità" :

```

1 ## Test per multiplayer. Testato attraverso unittest.
2
3     async def test_start(self):
4         communicator1 = await self.connect('start')
5         self.assertEqual(await self.start_game(communicator1), {
6             'message': 'waiting for opponent'
7         })
8
9         communicator2 = await self.connect('start')
10        response = await self.start_game(communicator2)
11        # check the start message of both players
12        self.assertDictContainsSubset({
13            'message': 'game started',
14            'board': chess.STARTING_FEN,
15            'time': 900
16        }, response)
17        #set the expected color communicator1 depending on the color of communicator2
18        color1 = 'w' if response['color'] == 'b' else 'b'
19
20        self.assertDictContainsSubset({
21            'message': 'game started',
22            'board': chess.STARTING_FEN,
23            'color': color1,
24            'time': 900
25        }, await communicator1.receive_json_from())
26
27        await communicator1.disconnect()
28        # let the second communicator receive the sense message if it's his turn
29        if(color1 == 'b'):
30            await communicator2.receive_json_from()
31
32        #check if disconnection is treated correctly
33        self.assertDictContainsSubset({
34            'message': 'game over',
35            'reason': ('white' if color1 == 'w' else 'black') + 'resigned'
36        }, await communicator2.receive_json_from())
37        await communicator2.disconnect()

```

”Come utente, voglio poter sfidare diversi bot, per affinare le mie abilità” :

```
1 ## Test del rematch contro i bot. Testato attraverso unittest.
2
3     async def test_resign(self):
4         communicator = await self.connect()
5         await communicator.send_json_to({
6             'action': 'start_game'
7         })
8         # receive the game started message and sense messages
9         await communicator.receive_json_from()
10        await communicator.receive_json_from()
11
12        await communicator.send_json_to(({'action': 'resign'}))
13        self.assertDictContainsSubset({
14            'message': 'game over',
15            'winner': False,
16            'reason': 'white resigned'
17        }, await communicator.receive_json_from(), )
18
19        await communicator.disconnect()
```

”Come utente, voglio vedere le info. personali, per distinguermi da altri giocatori” :

```
1 ## Test per il corretto login da parte dell'utente. Testato attraverso unittest.
2
3     def test_user_login_successful(self):
4         # Data to be sent in the POST request
5         data = {
6             'email': 'testuser@example.com',
7             'password': 'testpassword',
8         }
9
10        # Send a POST request to the userLogin view
11        response = self.client.post(self.login_url, data)
12
13        # Check if the response is successful (HTTP 200 OK)
14        self.assertEqual(response.status_code, 200)
15
16        # Check if the response content is as expected
17        self.assertEqual(response.content, b'User logged in successfully!')
```

Altre US testate:

- Classifica globale → **Obiettivo.** Verificare che i dati relativi a un match vengano salvati e riportati in classifica. **Azione.** Effettuare un match valido e poi controllare la leaderboard. **Verifica.** Deve esserci un aggiornamento visibile.

3.4.8 Burndown Chart



Figura 12: Burndown Chart Sprint 3.

3.4.9 Retrospettiva con Essence

RETROSPECTIVE	GOOD	OK	NOT GOOD
HIGH	<p>Software System: Operational - System available for use - User interface responsive - Agreed user test results reported</p> <p>Way of Working: Working Well - Previous progress level - Previous quality level - Previous stakeholder level - Customer level</p> <p>Work: Started - Development model - Development approach - Development tools - Tasks being progressed</p>	<p>Opportunity: Addressed - Stakeholders addressed - Solution works effectively - Business value achieved</p>	
MEDIUM	<p>Requirements: Achievable - Change control - Change control process - Change control system - Test cases fully documented</p>	<p>Stakeholders: In Agreement - Stakeholders involved - Stakeholders informed - Stakeholders satisfied - Stakeholder value & perspectives considered</p>	
LOW			

Figura 13: Risultato della retrospettiva sprint 3 in Essence.

Per informazioni più dettagliate, si consulti la Sprint Retrospective associata allo Sprint 3.

3.4.10 Sprint Review

Link review Sprint 3: [Review Sprint 3](#).

3.5 Sprint 4

Lo sprint in questione è iniziato in data 21/11/23 e si è concluso in data 4/1/24, per una durata totale di due settimane.

3.5.1 Sprint Goal

- Compensazione del debito tecnico derivato dallo Sprint precedente.
- Miglioramento dell'interfaccia utente.
- Bug fixing.
- Code Refactoring.

3.5.2 Sprint Backlog - Inizio Sprint

Durante questo Sprint, non sono state prodotte User Stories in quanto ci si è concentrati a migliorare funzionalità già presenti e descritte attraverso altre US nei precedenti Sprint.

3.5.3 Descrizione del codice prodotto

È stato colmato il debito tecnico cumulato durante lo Sprint precedente, implementando dunque le features mancanti. Si è denotato un generale miglioramento nella qualità del codice prodotto, aggiungendo nuovi test, sia lato backend che lato frontend. La grafica e i fogli di stile hanno subito varie modifiche al fine di effettuare un notevole revamp dell'applicazione, rendendo il gioco più intuitivo e godibile.

È stata inoltre migliorata la responsività generale in modo da permettere una corretta navigazione anche da mobile.

Lato Backend, le modifiche si sono concentrate sul miglioramento e l'aumento della stabilità della socket, oltre che dei controlli specifici sui messaggi che gestiscono lo stato del gioco attraverso l'API ReconChess. Sono anche state riviste le routes che permettevano ai dati di venire inseriti nel database PostgreSQL.

Infine, sono stati modificati i container docker a causa di alcune incompatibilità che si sono presentate in seguito a modifiche apportate in backend.

3.5.4 Definition of Ready

Non ci sono modifiche rispetto alla DoR presentata nello Sprint 3.

3.5.5 Definition of Done

Non ci sono modifiche rispetto alla DoD presentata nello Sprint 3.

3.5.6 Test effettuati sulle User Stories

US testate derivanti dallo sprint precedente:

- Social Network → **Obiettivo.** Verificare che le API per la condivisione del codice stanza funzionino correttamente. **Azione.** Premere le icone corrispondenti al social. **Verifica.** Il messaggio risulta condivisibile in tutti e tre i casi. *Non testata pienamente a questo punto dello sviluppo.*
- Risultati partite → **Obiettivo.** Verificare che i dati relativi a un match vengano salvati nel db. **Azione.** Effettuare un match valido e poi controllare il database. **Verifica.** Deve esserci un aggiornamento visibile. *Non testata pienamente a questo punto dello sviluppo.*

3.5.7 Retrospettiva con Essence

RETROSPECTIVE	GOOD	OK	NOT GOOD
HIGH			
MEDIUM	 		
LOW	 		

Figura 14: Risultato della retrospettiva sprint 4 in Essence.

Per informazioni più dettagliate, si consulti la Sprint Retrospective associata allo Sprint 4.

4 Descrizione del processo seguito

Questa sezione fornisce una panoramica del processo di sviluppo dell'applicazione web. Attraverso una serie di stadi e iterazioni dettate dai principi agili, il team ha svolto tutte le fasi comprese nel ciclo di vita del software: analisi e specifica dei requisiti, progettazione, codifica, testing e deployment.

4.1 Autodescrizione del Team

4.1.1 Teambuilding

Trello é stato il software proposto per la costruzione iniziale del team. Esso consente di organizzare, attraverso una bacheca Kanban, task, ma anche personale. Una volta determinati i gruppi, sono stati effettuati meeting propedeutici alla separazione e gestione del lavoro e conseguente assegnamento dei ruoli secondo il framework Scrum.

Inizialmente si é sperimentato un periodo di prova durante il quale svolgere due attività proposte, Escape The Boom e Scramble, il cui obiettivo era risaltare i punti di forza di ognuno nei confronti di un determinata posizione, ad esempio un membro del team particolarmente orientato all’organizzazione degli incontri stessi risultava più probabile che si sarebbe trovato bene nel ruolo di Scrum Master.

4.1.2 Ruoli nel team

Trello é stato utile anche per determinare sottoruoli non esclusivi di Scrum. Nel nostro caso:

- **Project Owner:** Patrick Alfieri
- **Scrum Master:** A rotazione, il ruolo é stato eseguito da molti.
- **Backend-dev:** Davide Luccioli, Giulia Torsani
- **Frontend-dev:** Kaori Jiang, Sofia Zanelli

In particolare in riferimento allo Scrum Master, il cui compito é essere responsabile di guidare e facilitare il processo, seguendo le pratiche e i principi Scrum, é stata adottata la proposta di rotazione del ruolo durante gli Sprint. Si ritiene però che un ruolo così importante mostri la sua maggiore efficacia con una risorsa stabile che possieda esperienza acquisita e maturata, piuttosto che tante menti che interpretano in modo differente le metodologie e artefatti agili.

4.2 Risultato del teambuilding

4.2.1 Scramble

Scramble é stata una delle attività proposte durante lo Sprint 0 al fine di favorire il Team Building. Lo scopo era quello di simulare, attraverso una sorta di gioco dell’oca, il processo di sviluppo agile/Scrum che il team si sarebbe trovato ad affrontare. Nonostante ci siano alcune evidenti

GOAL	QUESTIONS	PO Patrick Alfieri	SM Davide Luccioli	DEV Kaori Jiang	DEV Sofia Zanelli	DEV Giulia Torsani
Learn	Q1	4	5	4	4	4
	Q2	5	4	4	4	5
	Q3	5	5	5	5	5
Practice	Q4	5	5	5	5	5
	Q5	4	4	5	4	4
	Q6	5	3	3	4	3
Cooperation	Q7	3	4	3	1	3
	Q8	5	4	3	3	4
	Q9	4	4	5	4	4
Motivation	Q10	5	5	4	4	4
	Q11	4	4	4	4	4
	Q12	4	5	4	5	5
Problem Solving	Q13	3	4	5	5	3
	Q14	3	3	4	5	3
	Q15	3	4	5	4	3

Figura 15: Risultato dell’autovalutazione Scramble

differenze tra la simulazione e il processo reale, si ritiene che sia stato comunque un ottimo punto di partenza per ingranare con i concetti Scrum. Maggiori informazioni a riguardo sono presenti nel report specifico, presente in /doc/teambuilding/RelazioneScramble.pdf .

4.2.2 Escape The Boom

Escape The Boom è invece un gioco che mette alla prova la collaborazione e il problem solving utilizzando il tempo disponibile come limite.

Risulta essere un'attività più leggera rispetto alla precedente, ma non per questo meno stimolante. Il team, attraverso quest'attività, ha potuto sperimentare vie per comunicare in modo più efficace nelle situazioni critiche, superandole e proseguendo in vista degli obiettivi del processo. Maggiori informazioni a riguardo si trovano nel report specifico presente in /doc/teambuilding-/RelazioneETB.pdf .

4.3 Analisi dati gitinspector

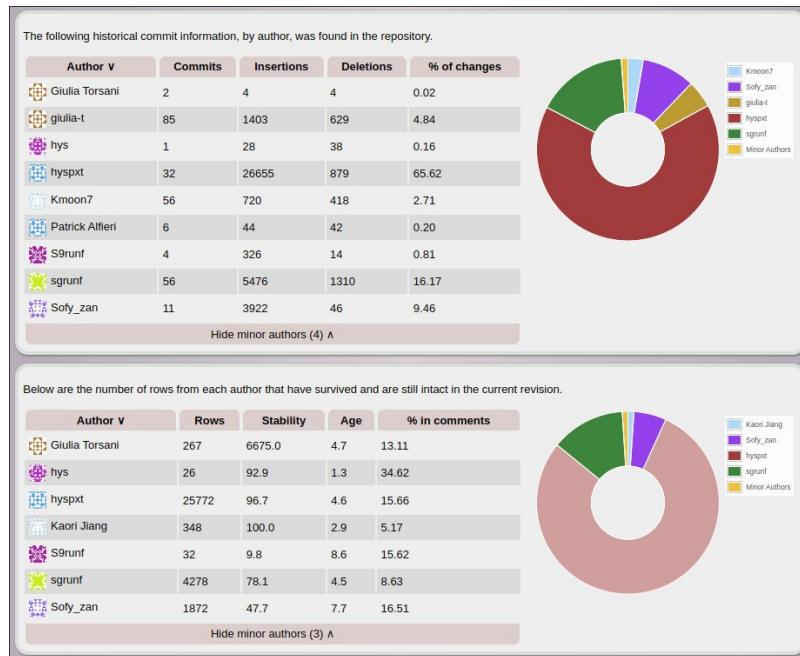


Figura 16: Schermata gitinspector, analisi dei commit - 1.



Figura 17: Schermata gitinspector, analisi dei commit - 2.
nota. Le misurazioni gitinspector sono state effettuate sul branch principale.

4.4 Strumenti di comunicazione

Come sistema di comunicazione interno al team, abbiamo preferito garantire la totale proprietà e controllo dei nostri dati utilizzando il protocollo **Matrix**, completamente self-hostato attraverso una droplet dedicata DigitalOcean e un dominio apposito. Il servizio è accessibile mediante il client **Element**, che fornisce un interfaccia intuitiva e funzionale per le comunicazioni del gruppo. Il server è organizzato in stanze, in modo da poter facilitare e organizzare i topic di cui si parla.

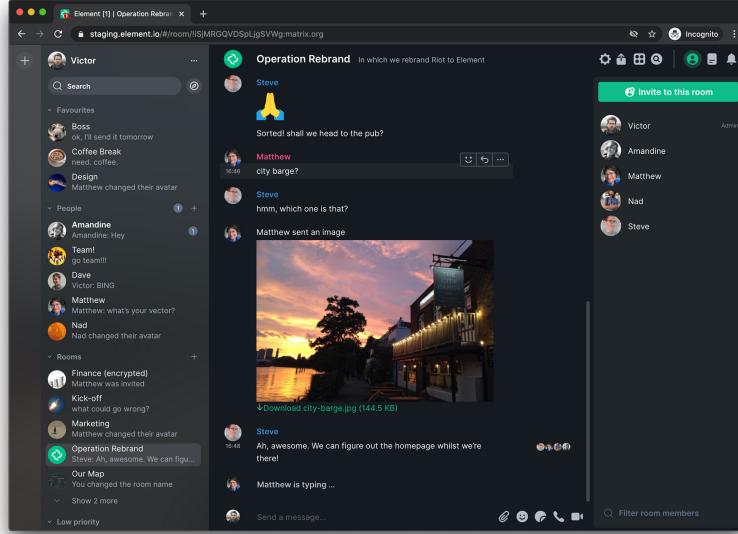


Figura 18: Interfaccia di Matrix.

A livello di networking, l'accesso a Matrix nella Droplet, viene gestito attraverso DNS Records specifici e accessibili tramite il dominio dedicato. Si accede correttamente al servizio grazie al Caddyfile che configura il reverse proxy, nel nostro caso verso chat.silverbullets.rocks.

Ovviamente, essendo un server privato è possibile l'accesso previa registrazione da parte degli amministratori.

4.4.1 Pregi di Matrix

Il più grande punto di forza del protocollo è sicuramente la decentralizzazione, oltre che il sistema crittografico incluso nelle chat (sia private che nelle stanze). Quel che è stato determinante però nella scelta rispetto alle alternative proposte, è stata la possibilità di gestire in totale autonomia le stanze, i dati degli utenti e l'intero sistema con facilità. Si aggiunge inoltre che sia Synapse sia Element, che sono rispettivamente il lato backend e lato client di Matrix, sono Open Source.

Oltre a queste, sebbene siano feature non del tutto sfruttate durante il processo, si segnalano:

- Possibilità di creare bridge con altri servizi (es. Whatsapp, Telegram, Mattermost...) federando Matrix ai loro server.
- Diffusione su ogni piattaforma: Windows, Linux, MacOs, Android, Web Browser...
- Quasi impercettibile a livello di risorse.

4.4.2 Difetti di Matrix

Sebbene i tanti pregi, non è esente però da alcuni difetti:

- L'installazione non è un'operazione molto intuitiva e richiede un po' di lavoro.
- Le notifiche sull'applicazione Android non sempre arrivano in tempo.
- Generalmente è più lento rispetto ad altri servizi come Telegram.

Ci riteniamo tuttavia molto soddisfatti della scelta fatta e riteniamo che i vantaggi siano da ponderare in modo più massiccio rispetto agli svantaggi.

4.5 Uso di ChatGPT / LLM

Durante lo sviluppo, è stato integrato ChatGPT come strumento di supporto al progetto, sia lato codice che non. Abbiamo però notato che su domande nemmeno troppo specifiche che riguardano la variante ReconChess, ChatGPT non è in grado di rispondere, probabilmente perché il suo addestramento si basa su dati antecedenti rispetto all'introduzione del gioco da parte della John Hopkins University, avvenuta circa nel 2022.

Dunque piuttosto che fornire soluzioni concrete (spesso in realtà errate), è stato comunque molto utile per indirizzare verso possibili librerie e metodologie che, se studiate accuratamente, potevano fornire la soluzione desiderata.

Di seguito alcuni prompt particolarmente rilevanti ai fini dello sviluppo:

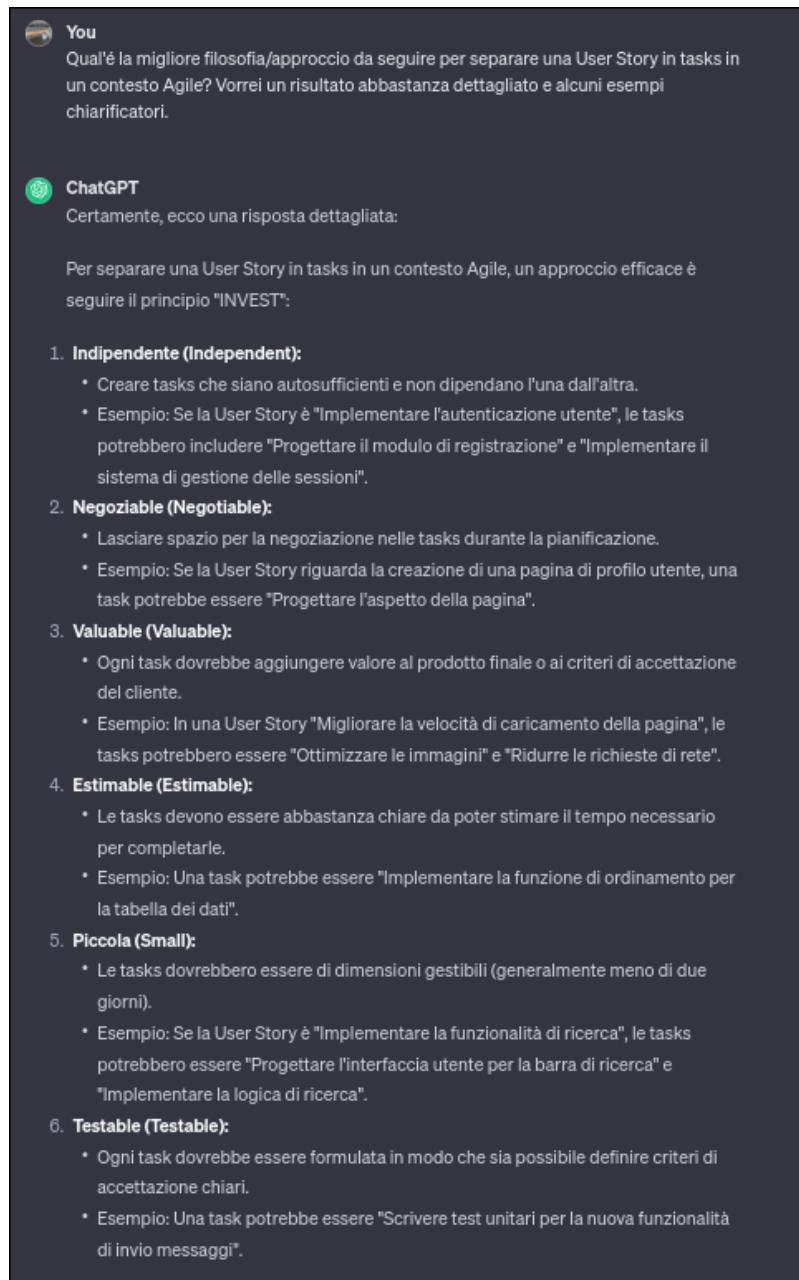


Figura 19: Prompt ChatGPT legato alle US.

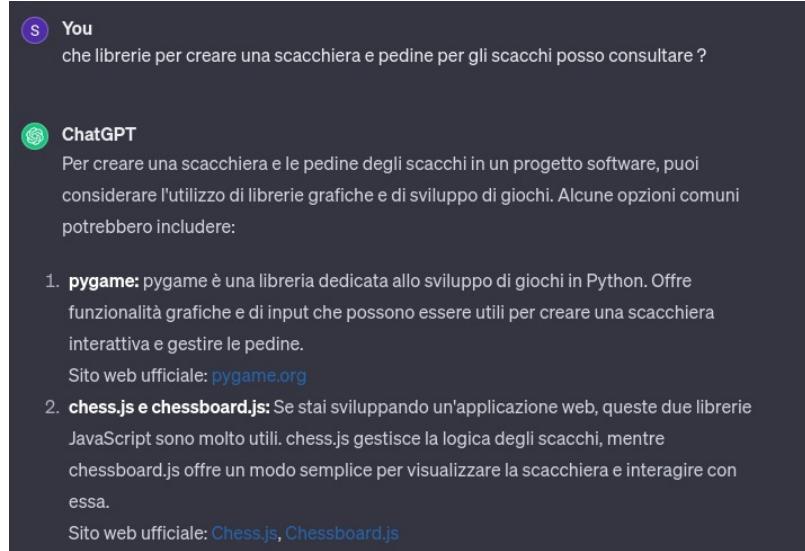


Figura 20: Prompt ChatGPT legato alle librerie scacchistiche.

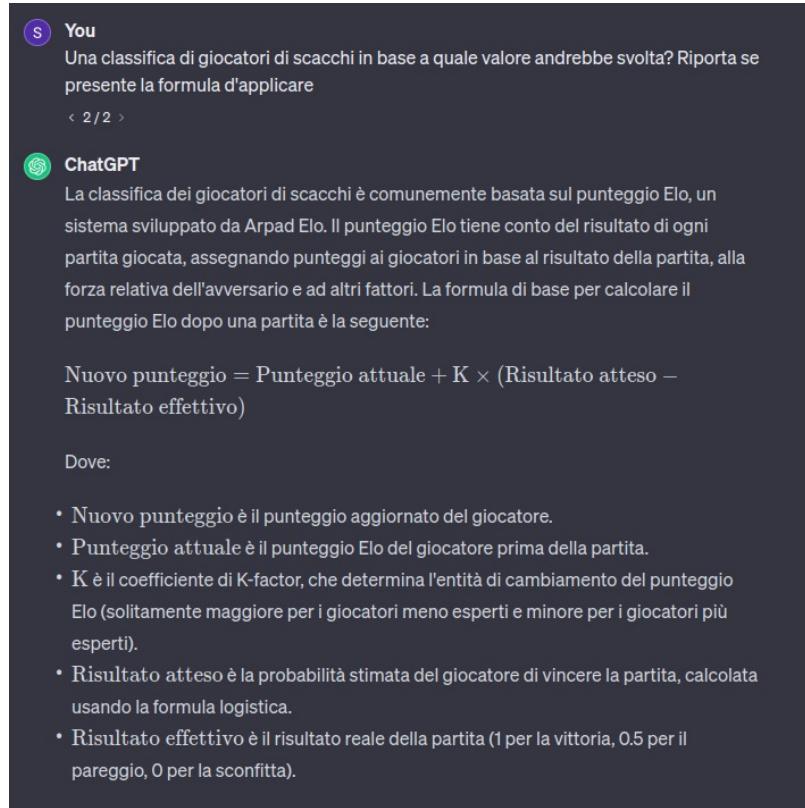


Figura 21: Prompt ChatGPT legato all'ELO score.
nota. Per il calcolo dell'ELO è stato utilizzato il paradigma **FSI**(Federazione Scacchi Italiana) che presenta la costante $k = 30$ e punteggio di partenza 1440. Le formule per calcolare la statistica presentate sopra sono infatti leggermente sbagliate e si sono invece utilizzate le seguenti:

$$E_A = \frac{1}{1 + 10^{\frac{R_B - R_A}{400}}} \quad R' = R_A + K(S_A - E_A)$$

dove R_A, R_B sono rispettivamente forze reali dei giocatori A, B ; E_A è il punteggio atteso di A in punti mentre ha ottenuto in realtà S_A punti. Dunque, R' è il punteggio ELO aggiornato di A .

4.6 Retrospettiva finale in Essence

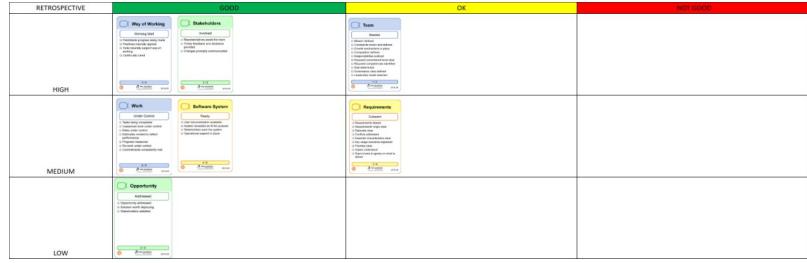


Figura 22: Retrospettiva sintetizzante del processo totale, in Essence.

4.7 Test E2E

I test End-to-End sono stati effettuati con il software open source [Cypress](#), il quale permette di creare test automatici sia a livello componenti, sia appunto E2E.

Rispetto ad alternative come Selenium che usa il WebDriver, Cypress esegue i test direttamente nel browser in maniera molto veloce e in tempo reale, fornendo maggiore controllo su ciò che si sta testando e rendendo più facile l'individuazione dei breakpoint.

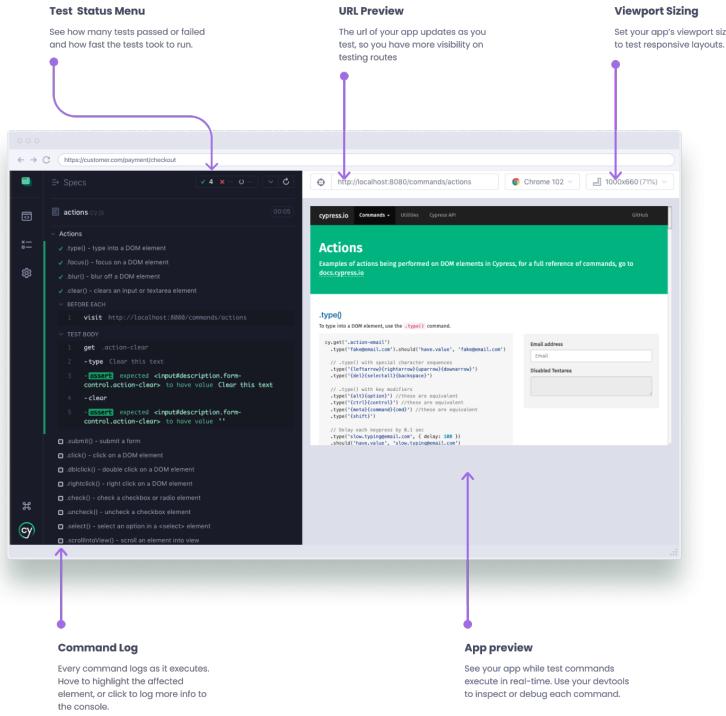


Figura 23: Interfaccia di testing di Cypress.

Il software dà anche la possibilità di cambiare rapidamente browser e ambiente di testing in base a quelli installati sul dispositivo, riconoscendoli automaticamente. Il browser di default indicato da Cypress è Electron.

Altro grande punto di forza è la sintassi intuitiva, molto simile ad altre librerie di testing JavaScript, come Jest, che permette di imparare facilmente l'utilizzo del software. La documentazione a riguardo è inoltre molto esaustiva e di semplice lettura.

```

1 // Esempio di test E2E
2
3 describe('Game initialization', () => {
4   beforeEach(() => {
5     cy.reload();
6     cy.visit('https://silverbullets.rocks')
7   })
8   it('Fill out registration form, well', () => {
9
10    cy.get('.btn_sign').click()
11    cy.get(':nth-child(1) > .input').type('cypressTest')
12    cy.get(':nth-child(2) > .input').type('cypressTest@gmail.com')
13    cy.get(':nth-child(3) > .password').type('*****')
14    cy.get(':nth-child(4) > .password').type('*****')
15    cy.get('#signup').click()
16    cy.wait(2000);
17    cy.get('.alert').should('contain', 'already in use')
18  })
19
20  it('Check user in DB', () => {
21    cy.visit('https://silverbullets.rocks/...')
22    cy.get('#id_username').type('*****')
23    cy.get('#id_password').type('*****')
24    cy.get('.submit-row > input').click();
25    cy.get('.model-user > th > a').click();
26    cy.get(':nth-child(2) > .field-email').should('contain', 'cypressTest@gmail
27 .com')
28 })
}

```

Il setup é molto semplice, in quanto basta posizionarsi in un ambiente nodejs, lanciare `npm install cypress` per l'installazione e `npx open cypress` per l'inizializzazione guidata, che creerà una directory cypress dove poter scrivere i propri test modulari.

4.8 Schema architetturale finale

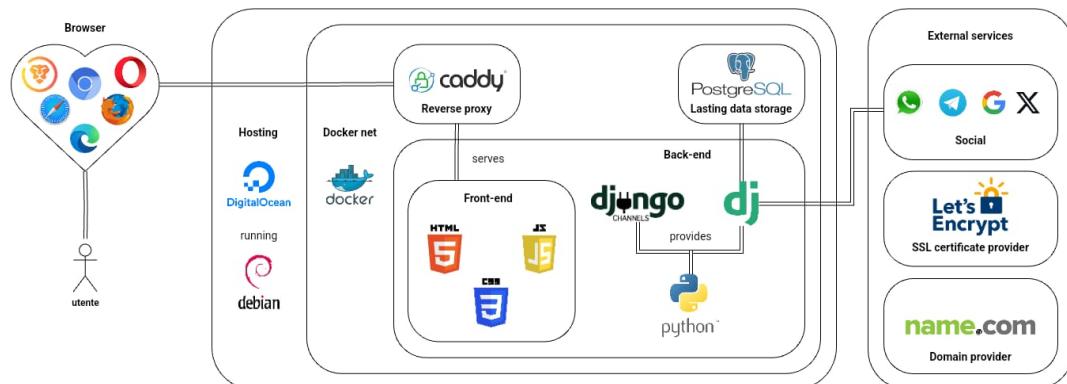


Figura 24: Schema architetturale finale.

Abbiamo scelto di non fare uso di KeyDB in quanto non era fondamentale avere un database estremamente performante; la gestione tramite le metodologie offerte da Django con PostgreS ci sono sembrate adeguatamente efficienti. Il sistema operativo presente sulla droplet é stato cambiato da Ubuntu a Debian per ragioni di potenziamento della macchina virtuale, in particolare per sostenere le reti create da StockFish.

4.9 Schermata finale di Sonarqube

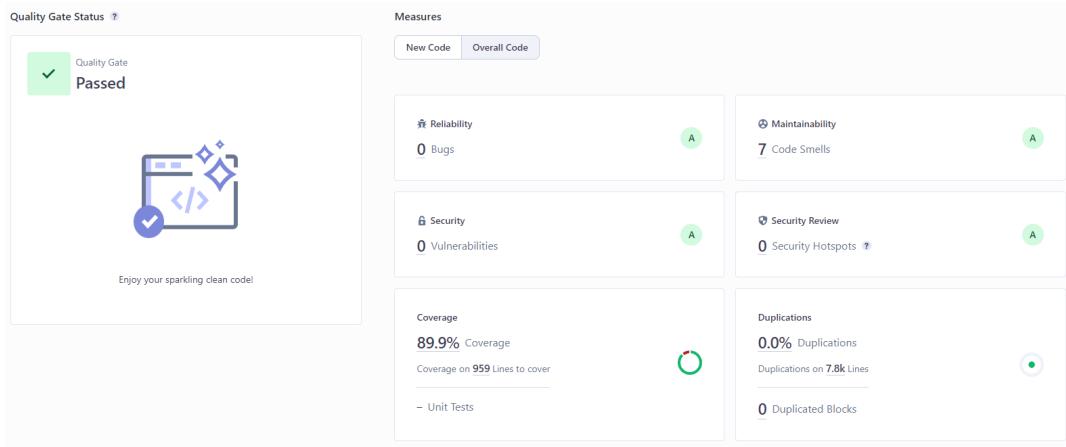


Figura 25: Risultati dashboard di SonarQube.

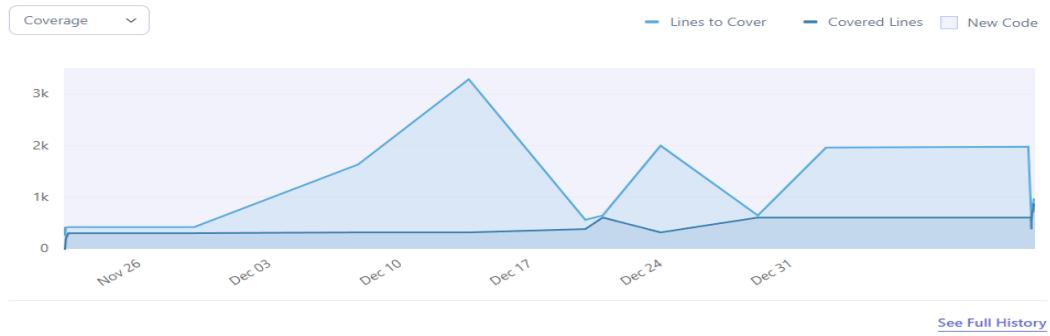


Figura 26: Analisi Coverage.

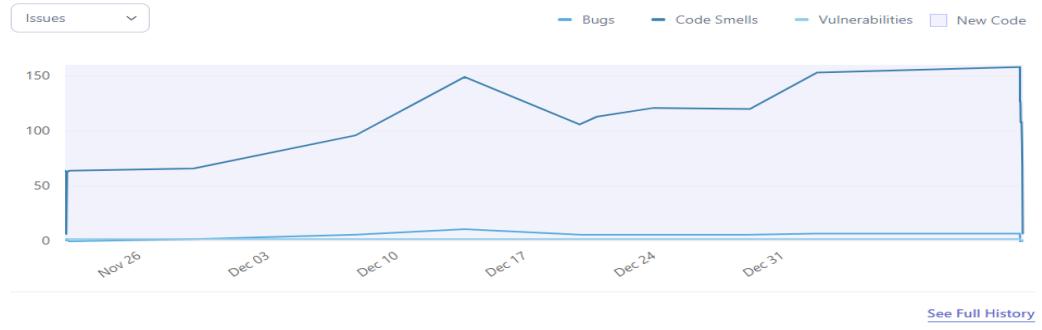


Figura 27: Analisi Issues.

4.10 Diagramma di Deployment

Un diagramma di deployment rappresenta la disposizione e distribuzione dei componenti hardware/software di un sistema e le loro connessioni, attraverso dei nodi.

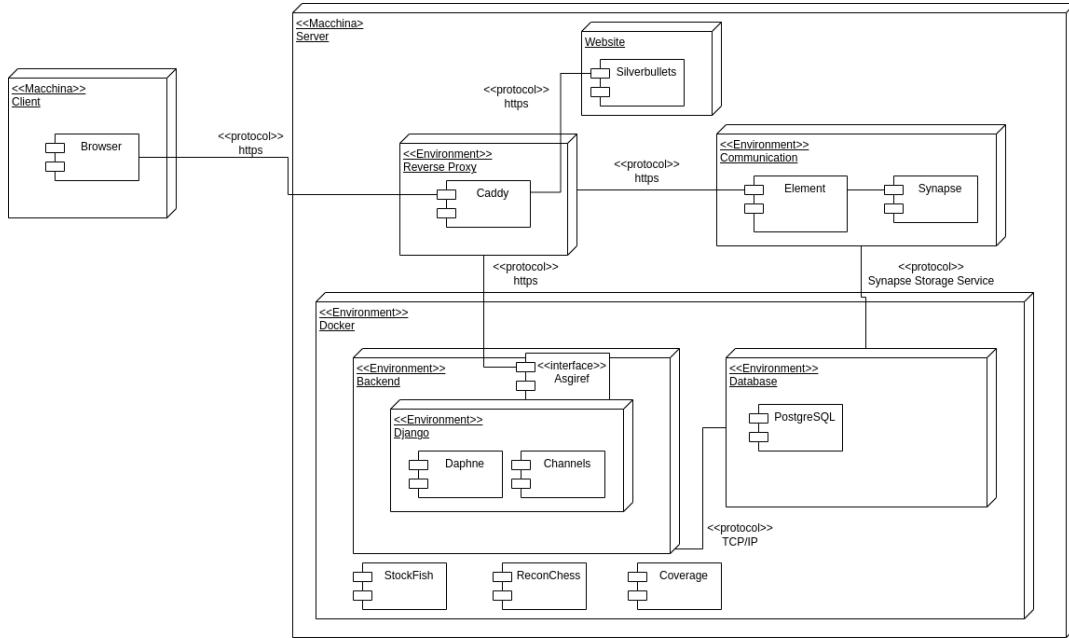


Figura 28: Deployment Diagram dell'applicazione.

4.11 Demo del prodotto

Link alla demo del prodotto: [Link](#)

4.12 Lista degli artefatti in consegna e modalità di accesso

Artefatti di prodotto presenti nella [cartella GitLab](#).

- Codice sorgente applicazione web in /code.
- Test di unità in /test.
- Test E2E in /cypress.
- Dump del database in /database.

Artefatti di processo in consegna in /doc.

- Per ogni sprint escluso 0, nelle omonime directory sono presenti: Sprint Analysis (DoR, DoD, Backlog,...), Sprint Retrospective e Sprint Review (per lo Sprint 4 vale quella dello Sprint 3, dato che non sono state prodotte US).
- Relazioni di teambuilding, quali Escape The Boom e Scrumble.
- Mockup e schemi architetturali iniziali.

Modalità di accesso al prodotto. Si può creare un nuovo account attraverso il form di registrazione se si desidera loggare come utente standard, altrimenti attraverso usr: **admin** e pw:**admin** è possibile accedere come amministratore.

4.13 Link a prodotto live

Link prodotto: silverbullets.rocks

5 Conclusioni

Lo sviluppo del prodotto é stato molto stimolante ed istruttivo, sia nell'ottica del seguire i principi agili, sia utilizzare nuovi applicativi che hanno permesso al processo di giungere allo stato odierno. Si ritiene che il processo di sviluppo affrontato, sebbene in forma ridotta, sia davvero utile e chiarificatore anche nei confronti di progetti reali di dimensioni molto maggiori.

Infine, il team é d'accordo sul fatto che molte delle corrette scelte progettuali prese rispecchino la struttura comunicativa adottata; ulteriore riprova della veridicità della Legge di Conway.