Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

PROGETTAZIONE DI UNA DASHBOARD PER SVILUPPATORI AGILI

Relatore:
Chiar.mo Prof.
PAOLO CIANCARINI

Presentata da:
SALVATORE PERRI

Sessione IIIa
Anno Accademico 2019-20

Introduzione

Nel seguente elaborato analizzeremo il ruolo che gli strumenti di self-tracking, nello specifico le dashboard, hanno assunto e stanno assumendo nel mondo dello sviluppo software. Inizieremo definendo cos'è una dashboard, quali tipologie esistono, che dati solitamente contiene e i rischi che si possono incontrare nell'utilizzo.
Faremo riferimento ad un particolare ambiente di sviluppo: CAS, ospitato sul server Aminsep del DISI. Qui sono messi a disposizione diversi servizi open source utili a chi lavora adottando un modello di sviluppo software agile, nello specifico iAgile.
Successivamente metteremo in relazione l'ambiente CAS con la pratica del self-tracking, analizzando più da vicino con quali modi e strumenti venga messa in pratica.
A tal fine verranno esaminati i plugin per CAS, utili a raccogliere varie statistiche durante la scrittura di codice. Ne descriveremo la tipologia di dati raccolti, l'implementazione, l'installazione ed il funzionamento.
La nuova dashboard prodotta per questa tesi, sarà successivamente introdotta, mostrando i cambiamenti rispetto alla versione attuale.
Per ultimo concluderemo definendo perché il self-tracking possa risultare utile ad uno sviluppatore e ad i suoi colleghi, presentando anche quelle che sono le criticità del suo utilizzo.

Indice

Elenco delle figure

Listings

Capitolo 1

Dashboards nell'ingegneria del software

> A dashboard is a visual display of the most important information needed to achieve one or more objectives which fits entirely on a single computer screen so it can be monitored at a glance.
>
> *Stephen Few*

1.1 Definizione

Una dashboard, letteralmente *"cruscotto"*, è un pannello di controllo che include tutte le informazioni chiave dello stato di una soluzione software. Mediante il suo utilizzo gli stakeholder sono in grado di:

1. individuare eventuali **bottleneck** nel processo produttivo;
2. prendere **decisioni** basandosi sui dati consultati.

La qualità del suo design è direttamente proporzionale alla sua efficacia comunicativa. Una dashboard deve presentare il contenuto in maniera chiara e concisa, per consentire una comprensione rapida di quelli che sono i **KPI** (*Key Performance Indicator*) dell'oggetto in esame. La rappresentazione dei dati raccolti è quindi spesso realizzata con l'ausilio di tabelle, grafici e widget vari, favorendo pertanto i numeri al testo. (fig. 1.1)

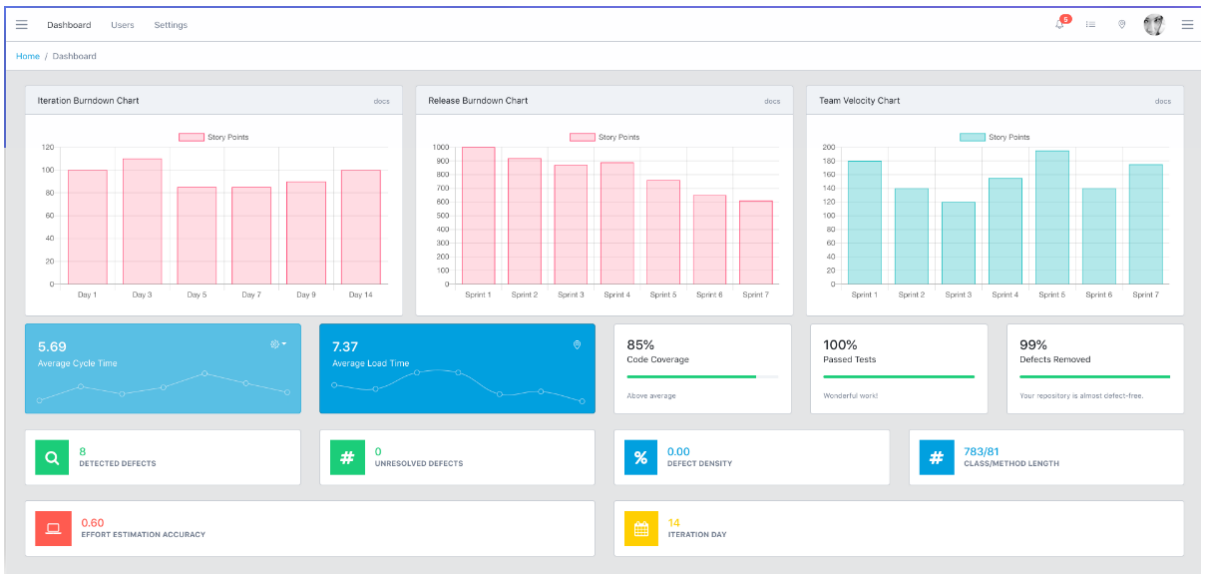


Figura 1.1: Innometrics dashboard [7]

Secondariamente alla facilità di comprensione, deve essere garantito anche un certo livello di interattività, per consentire all'utente di poter restringere il campo di ricerca dell'informazione a cui è interessato.
Com'è facile intuire, al variare dell'utilizzatore variano anche i contenuti presenti in una dashboard. È quindi presente una forte dipendenza dal contesto di utilizzo che fa si che esistano numerose tipologie dello strumento. Nella prossima sezione ne verranno elencate alcune.

1.2 Classificazione

Riprendendo ciò che Stephen Few riporta nel suo lavoro [8], è possibile classificare le dashboard in molteplici modi, a seconda del criterio che si vuole usare. Per semplicità e coerenza al nostro oggetto di studio terremo in considerazione due possibili criteri: **ruolo**, **portata dei dati**.

1.2.1 Suddivisione per ruolo

È possibile effettuare una prima classificazione tenendo conto di quale scopo abbia l'utilizzo dello strumento. Si individuano dashboard di tipo:

- Strategico
- Operazionale
- Analitico

1.2.1.1 Dashboard Strategiche

Sono la tipologia più utilizzata in diversi ambiti, al di fuori dello sviluppo software, prevalentemente supportano i manager di qualsivoglia azienda o organizzazione dando una visione chiara del suo stato di salute aiutandoli nel processo decisionale.
Tipicamente includono pochi ed importanti parametri da tenere sotto controllo, per risultare efficaci non necessitano obbligatoriamente di dati in tempo reale e possono a volte prevedere l'integrazione di strumenti di previsione al loro interno.

1.2.1.2 Dashboard Analitiche

Le dashboard di tipo analitico sono utili agli analisti per individuare pattern nel ciclo di vita di un progetto, di un reparto, di una compagnia ecc., al fine di aiutarli a comprendere le cause di tale fenomeno.
A questo proposito è necessario offrano un certo livello di interattività coi dati, di comparazione ed una fonte di informazioni sufficientemente ampia a livello temporale.

1.2.1.3 Dashboard Operazionali

Per i progetti software sono la categoria più frequentemente utilizzata [22]. L'aspetto che più caratterizza le dashboard operazionali è la dinamicità e mutevolezza dei dati, che sono aggiornati in tempo reale.
Così come per le dashboard strategiche presentano un'interfaccia semplice, anche se le informazioni rappresentate sono più specifiche ed hanno un maggiore livello di dettaglio.

1.2.2 Suddivisione per portata dei dati

Come già anticipato è possibile categorizzare le dashboard in base alla portata dei loro dati. Procedendo per livello di crescente di risoluzione vengono raccolti dati inerenti a:

- Sviluppatore
- Team
- Progetto
- Community

1.2.2.1 Singolo sviluppatore

Uno sviluppatore può rifarsi ai dati personalmente raccolti per avere un'idea di come ha impiegato il suo tempo, su quali file, e quindi su quali feature ha speso più energie, quante linee di codice ha aggiunto e/o rimosso in un determinato periodo, quanti commit ha sottoposto in un repository piuttosto che quanti test ha effettuato sul suo prodotto.
L' utente che utilizza una dashboard in questo modo vuole avere una percezione della sua produttività e dell'efficacia del suo lavoro, per poter successivamente mettere in atto eventuali miglioramenti.

1.2.2.2 Team di sviluppatori

Il passo immediatamente successivo è quello di monitorare l'attività di un intero team di sviluppo aggregando i dati dei singoli membri. In questo caso chi ha interesse a consultare la dashboard è un team leader o un manager che ha l'intenzione di migliorare l'ambiente di lavoro e i processi di sviluppo.

Nonostante possa sembrare controverso sul piano della privacy degli sviluppatori, è molto importante avere dati aggregati del team, ancor di più in quelli distribuiti. È di cruciale importanza per capire se il modo in cui si sta operando sia funzionale o meno al raggiungimento degli obiettivi prefissati.
Particolarmente inclini all'utilizzo dello strumento sono i team che lavorano seguendo la filosofia Agile [14], che sono interessati a monitorare le loro prestazioni durante i diversi sprint.

1.2.2.3 Progetto

Ipotizzando si abbia interesse a tracciare un intero progetto software, l'attenzione è da rivolgere non verso le dinamiche di gruppo e processi, ma verso la qualità ed affidabilità del software prodotto. È usuale inoltre che siano presenti dati che riguardano il processo

CI/CD [17].
Alcune dashboard tengono in considerazione persino la gestione della clientela.

1.2.2.4 Community

In maniera analoga al livello precedente alcune dashboard sono usate per controllare lo stato di salute di community. Sadowski e Zimmermann fanno l'esempio della community di Linux, che ha inserito tra i suoi *indici di salute* il numero di licenze utilizzate e si avvale di CHAOSS, un progetto della Linux Foundation nato cono lo scopo di definire quali siano i parametri utili a determinare lo stato di una community.

1.3 Natura dei dati

Tenendo a mente quanto detto nella precedente sezione, nello specifico che il contenuto di una dashboard varia a seconda del contesto, proviamo ora ad analizzare quali siano le grandezze tipiche delle dashboard nel mondo dello sviluppo software.

Un gruppo di indicatori comuni, anche detti **metriche**, è composto da:

- Linee di codice (LOC)
- Problemi irrisolti (Open issues)
- Problemi risolti (Closed issues)
- Test effettuati e risultati
- Tempo di lavoro su uno specifico file
- Commit effettuati

Al variare degli strumenti di sviluppo, della tipologia di processi produttivi adottati, e degli obiettivi degli stakeholder è possibile determinarne molti altri. Ad esempio i team che lavorano seguendo un framework Agile sono soliti tenere conto di:

- User stories completate
- Punti per sprint
- Sprint burndown
- Code coverage

Altri ancora possono essere interessati a monitorare la gestione dei clienti, quindi rivolgeranno le loro attenzioni ad eventuali feedback e soddisfazione complessiva dell'utente finale, basando su queste le loro future scelte implementative.

1.4 Rischi dell'utilizzo

Avendone elencato i pregi, analizziamo quindi quali siano secondo Sadowski e Zimmerman i rischi che si corrono utilizzando una dashboard, che sono accentuati se ci si dimentica che le dashboard sono uno **strumento di supporto** e non costituiscono verità assoluta.

Ripartendo dal concetto espresso in sezione 1.1, le dashboard danno molto più valore ai numeri che al testo. Sorge spontaneo il problema di conciliare la natura testuale degli artefatti che solitamente vengono prodotti nel mondo software con l'utilizzo delle dashboard. È vero che in linea di massima esistono grandezze che rappresentano abbastanza fedelmente le caratteristiche di un artefatto, ma spesso questa traduzione da testo a numeri si lascia dietro dei pezzi di informazione più o meno rilevanti.

L'accumulazione di informazioni è una **selezione** delle informazioni di partenza, non il suo completo insieme. Questo può comportare che alcuni dettagli vengano trascurati e che quindi venga meno il contesto degli stessi. La mancanza di contesto rende conseguentemente difficile la comparazione tra ad esempio due progetti software o due sviluppatori.

Come osservano i due autori, gli sviluppatori che sono consapevoli delle misurazioni in gioco tendono a mettere in atto comportamenti che alterano la qualità e veridicità delle stesse. Se una certa dashboard considera nelle sue rappresentazioni il livello di produttività come il tempo impiegato all'interno dell'IDE dagli utenti, gli stessi esiteranno a cercare informazioni all'esterno, in un browser ad esempio.

Ipotizzando di potersi avvalere di una dashboard il cui design sia ottimale, l'aspetto influente resta quello della qualità dei dati.

$$dashboard = design + dati$$

$$=> dashboard = dati$$

Eliminando pertanto dall'equazione la qualità del design, ciò che più influirà sul risultato finale è:

i Quali dati raccogliere;

ii Come raccoglierli.

Se questi due aspetti non vengono affrontati in modo adeguato, è possibile che il contenuto della dashboard risulti fuorviante.

È anche vero che alcune delle attività che risultano fondamentali durante lo sviluppo

software, come uno scambio di pareri tra due sviluppatori, non possano essere catturate dagli strumenti in uso. Per quanto ci si sforzi nella pianificazione della fase di raccolta dati, ci sarà sempre qualche informazione chiave non catturabile, rimarcando il concetto che a volte le dashboard non sono in grado di fornire il quadro completo della situazione.

Per ultimo, ma non per ordine di importanza, è presente il rischio di interpretare dati che riguardano la performance di un individuo come dati che rispecchino la produttività dello stesso.
Chiunque si sia mai cimentato nella pratica di scrittura del software sa bene quanto misurare la mole di lavoro prodotta in linee di codice sia un esercizio a dir poco inutile e poco rappresentativo dell'attuale sforzo.
Per questo motivo spesso gli sviluppatori sono restii ad utilizzare strumenti che riducano la complessa attività di sviluppo a dei semplici numeri.

1.5 Esempi

1.5.1 Wakatime

WakaTime è una dashboard che offre un servizio di tracciamento automatico del tempo impiegato in attività di coding. Per il suo funzionamento necessita dell'installazione di un plugin (open-source) per l'IDE o editor che si intende utilizzare, il quale registra le statistiche quando verifica la presenza di input da parte dell'utente.

Figura 1.2: Wakatime dashboard

In pratica usa dei cosiddetti *heartbeat* per inviare le statistiche alla API del servizio, ogni volta che rileva uno dei seguenti eventi:

- **File cambiato**: quando l'attenzione dell'editor viene spostata su un altro file;
- **File modificato**: quando l'utente modifica il file in osservazione;
- **File salvato**: ogni volta che i cambiamenti di un file vengono salvati su disco.

Tra le informazioni raccolte sono presenti:

- Nome del progetto
- Nome dell'editor
- Linee di codice dei file
- Linguaggio utilizzato

- Librerie e dipendenze presenti
- Informazioni varie sull'hardware in uso

È compatibile con oltre 500 linguaggi, grazie all'utilizzo dello strumento **pygments** e con molti degli editor ed IDE più utilizzati.

Prevede sia un utilizzo da singolo che da team, consentendo di impostare degli obiettivi giornalieri di produttività da raggiungere, generando classifiche private e pubbliche.

In figura 1.2 sono raffigurate le diverse sezioni contenenti le statistiche personali dello sviluppatore, in particolare si individuano:

1. Grafico dello sforzo giornaliero degli ultimi sette giorni;
2. Grafici raffiguranti linguaggi ed editor utilizzati;
3. Diagramma di Gantt in cui viene mostrato come lo sforzo è stato impiegato nei vari progetti ed in che fascia oraria;
4. Grafico del raggiungimento dell'obiettivo di ore giornaliero e comparazione con la media.

In figura 1.3 è mostrato un esempio di leaderboard pubblica inerente l'attività dei singoli sviluppatori negli ultimi sette giorni, comprensiva di ore totali di coding, media giornaliera e linguaggi utilizzati.

Rank	Programmer	Hours Coded	Daily Average	Languages Used	
1	A	126 hrs 25 mins	18 hrs 3 mins	Go, TypeScript, SCSS, JavaScript, HTML, PHP, JSON, Text, Nginx configuration file	hireable
2	Š	97 hrs 4 mins	13 hrs 52 mins	Bash, Config, Python, INI, TOML, YAML, Markdown, Lua, Dockerfile, ...	hireable
3	C	95 hrs 17 mins	13 hrs 36 mins	Lua, Svelte, Vue.js, JavaScript, SCSS, HTML, CSS, JSON, INI, ...	
4	J	88 hrs 24 mins	12 hrs 37 mins	Python, Bash, Text, INI, Git Config, Makefile, XML, Jinja2, Markdown, ...	
5	P	81 hrs 21 mins	11 hrs 37 mins	TypeScript, sh, Python, JSON, Markdown, Terraform, HTML, JavaScript, Text, ...	hireable
6	S	76 hrs 38 mins	10 hrs 56 mins	JavaScript, PHP, JSON, TypeScript, Vue.js, HTML, Markdown, EJS, CSS, ...	
7	L	76 hrs 33 mins	10 hrs 56 mins	TypeScript, Vue.js, TOML, Rust, JavaScript, JSON, YAML, GraphQL, Markdown, ...	
8	M	75 hrs 24 mins	10 hrs 46 mins	TypeScript, XML, JavaScript, JSON, Batchfile, Text, PHP, CSS, HTML, ...	
9	V	74 hrs 51 mins	10 hrs 41 mins	TypeScript, YAML, JSON, JavaScript, LESS, Docker, Terraform, MDX, PowerShell, ...	hireable
10	F	72 hrs 41 mins	10 hrs 23 mins	TypeScript, HTML, JavaScript, Markdown, JSON, Text, Lua, Git Config	

Figura 1.3: wakatime leaderboards

1.5.2 Process Dashboard

Nata in ambito militare, per supportare il Personal Software Process [12], successivamente mantenuta secondo il modello open-source. Come WakaTime offre un servizio sia ai singoli sviluppatori che ai team. Ciò che la caratterizza è il diverso modo di catturare dati: non dipende da alcun IDE o ambiente, è sufficiente che l'utente avvii il timer e scelga l'attività che sta eseguendo, sia scrittura di codice, debugging o testing. L'applicazione genera dei log temporali mentre è attiva la registrazione, questi saranno poi utilizzati per generare le statistiche a cui è interessato l'utente.

Process Dashboard offre interessanti funzionalità di previsione, com'è possibile apprezzare in figura 1.4, dove un grafico mostra l'andamento pianificato della percentuale di completamento del progetto (**Plan**) e quello effettivo (**Actual**).
Tenendo in considerazione lo scostamento tra le due grandezze, il software suggerisce un nuovo andamento (**Replan**), basando la sua previsione sul numero di task completati, il tempo speso su quelli da completare ed il tempo rimasto a disposizione fino alla prossima scadenza; dopodiché stabilisce una nuova data (**Forecast**) di completamento partendo dallo stato di avanzamento attuale.

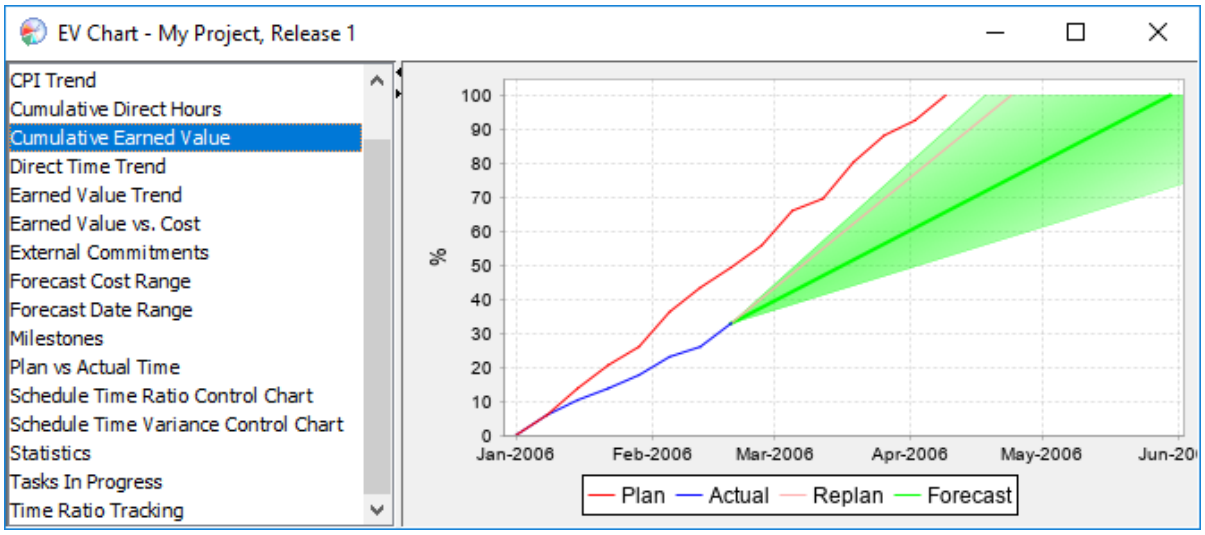


Figura 1.4: Strumento di previsione di Process Dashboard

Capitolo 2

Compositional Agile System

2.1 Premessa: Agile Software Development

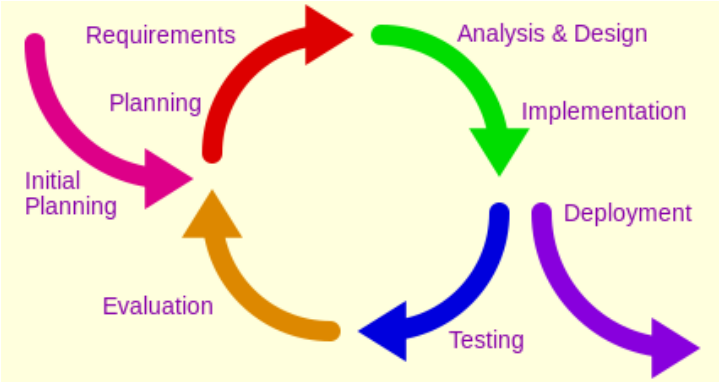
Lo sviluppo agile, consiste in un insieme di linee guida inerenti il processo di produzione di software. Nasce come alternativa al più classico e rigido modello **waterfall**, con l'ambizione di essere più responsivo ai cambiamenti in corso d'opera e di rendere più importanti i contatti con i clienti, anche durante la fase di sviluppo.

I suoi dettami derivano fondamentalmente dai principi del cosiddetto **Manifesto Agile** [2], che recita testualmente:

Stiamo scoprendo modi migliori di creare software,
sviluppandolo e aiutando gli altri a fare lo stesso.
Grazie a questa attività siamo arrivati a considerare importanti:

Gli individui e le interazioni più che i processi e gli strumenti
Il software funzionante più che la documentazione esaustiva
La collaborazione col cliente più che la negoziazione dei contratti
Rispondere al cambiamento più che seguire un piano

Ovvero, fermo restando il valore delle voci a destra,
consideriamo più importanti le voci a sinistra.



In pratica un team agile si impegna a rilasciare versioni **funzionanti** del software con una cadenza regolare seguendo intervalli di tempo denominati **sprint**, tipicamente lunghi dalle due alle quattro settimane. Alle fine di uno sprint al software vengono implementate le nuove funzionalità sviluppate durante la sua durata.

Questo approccio è noto come **sviluppo incrementale** (e iterativo). Tra i vantaggi dell'agire in questo modo vi è la possibilità di apportare modifiche ai requisiti del prodotto prima di iniziare un nuovo sprint, aumentando la flessibilità del processo ma soprattutto rendendo i cambiamenti meno costosi.

Il software funzionante è quindi il metro di giudizio di un team per stabilire il livello di progresso del lavoro. Spesso i team agili organizzano riunioni per analizzare il proprio operato e riflettere su eventuali soluzioni per risultare più efficaci, questa pratica è detta **retrospezione**.

Per ultimo la metodologia agile promuove la semplicità e l'auto organizzazione ma allo stesso tempo ritiene essenziale la buona progettazione e la qualità tecnica.

2.2 Definizione

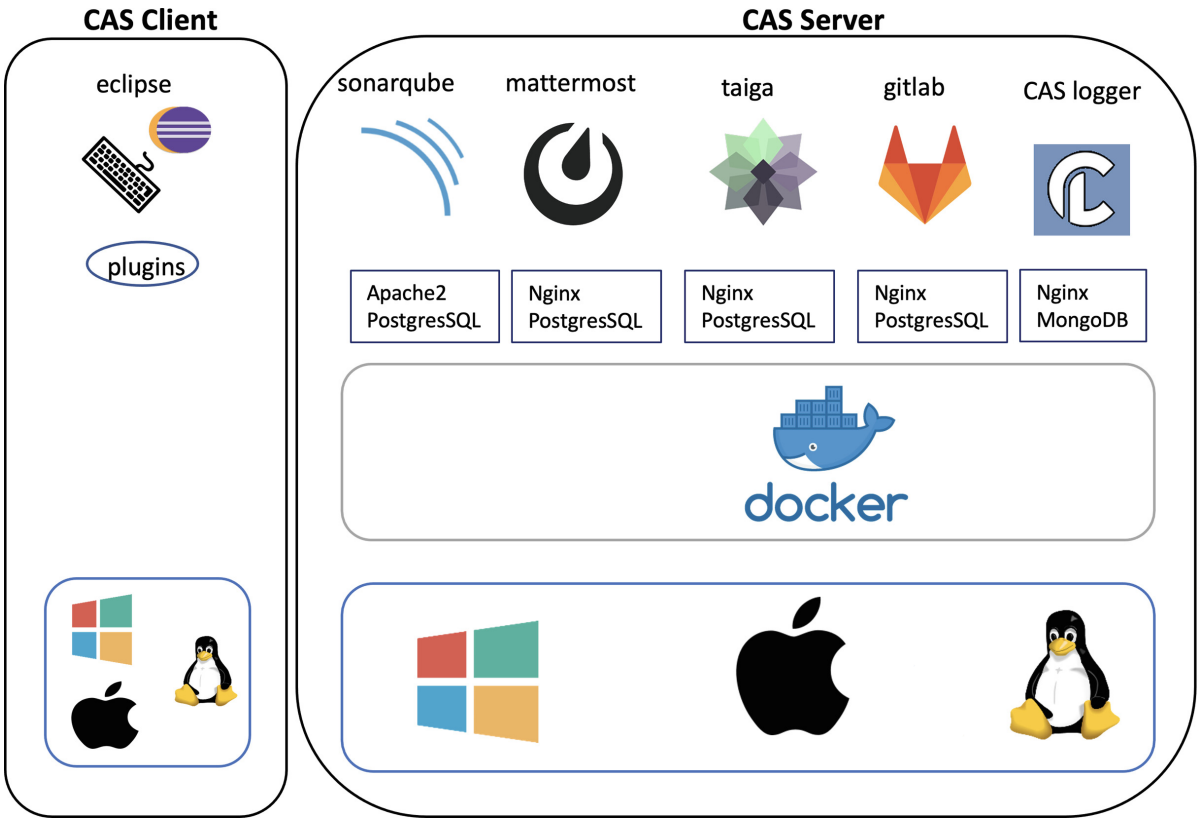
Il Compositional Agile System (**CAS**) è un ambiente di sviluppo nato per supportare un approccio **iAgile** [6], un modello agile derivato da **Scrum**, introdotto appositamente per lo sviluppo di sistemi critici in ambito militare.
Lo scopo della sua implementazione è quello di fornire un ambiente open source adatto ad essere distribuito sia in una rete privata o in cloud ibrido, mantenendo completo controllo su di esso.

2.3 Struttura

L'architettura di CAS è riconducibile a due sottosistemi fondamentali:

a. **CAS Server**, un insieme di microservizi modulari;

b. **CAS Client**, IDE all'interno dei quali viene installato un apposito plugin.



2.3.1 CAS Server

Come già stabilito tutti i servizi sono open source, ed ognuno di essi è utile a svolgere alcune delle diverse attività solite dello sviluppo:

- Taiga, **project management**;
- SonarQube, **analisi statica del codice**;
- Mattermost, **comunicazione**
- GitLab, **VCS**
- Logger, **self-tracking**

2.3.1.1 Taiga

È un servizio ideato per consentire a startup e sviluppatori agili di gestire i loro progetti, registrandone lo stato nel tempo. Supporta i team che lavorano con metodologie agili come *Scrum* e *Kanban*.

Durante la fase di creazione di un progetto, gli utenti possono decidere se fare uso di: **Epiche**, **Backlog** o **Kanban**, sezioni **Issues** e **Wiki**, impostare il sistema di **video-conferenza** preferito, e integrare eventuali sistemi di **VCS**. In sostanza Taiga fornisce agli sviluppatori una visione chiara e completa dello stato di avanzamento del progetto, della ripartizione dei compiti e degli eventuali problemi.

2.3.1.2 SonarQube

Sviluppata da SonarSource, è una piattaforma che mediante l'analisi statica del codice genera dei cosiddetti report automatici di qualità del software, individuando bug, code smell e vulnerabilità.

Quando viene avviata una scansione del codice sorgente dall'utente, l'esito viene inviato al server che ne interpreta i risultati secondo le regole prestabilite. Alcuni parametri riguardano: affidabilità, sicurezza, manutenibilità, dimensione e complessità.

Eseguendo analisi periodiche ad ogni modifica significativa, SonarQube permette di visualizzare come nel tempo i suddetti indici di qualità siano cambiati nel tempo.

2.3.1.3 Mattermost

Mattermost è un sistema di messaggistica self-hosted per la comunicazione dei team, consente agli utenti di condividere file, avviare chiamate audio e video ed organizzare le

chat in canali pubblici e privati.
Grazie alle numerose integrazioni è possibile aggiungere funzionalità alle chat o collegare altri servizi, ad esempio si può ricevere una notifica quando sul proprio repository git viene creato un nuovo issue.

2.3.1.4 GitLab

È una piattaforma per la gestione di repository git, che fornisce Wiki, issue-tracking e funzionalità CI/CD.
Come è risaputo i repository git sono fondamentali per la coordinazione durante lo sviluppo, poiché agevolano la collaborazione tra sviluppatori che lavorano parallelamente ad uno stesso progetto, tenendo una cosiddetta *commit history*.
In questo modo si semplificano problemi di *merge* di versioni differenti, o di ripristino di altre che risultavano meglio funzionanti.

2.3.1.5 Logger

Ogni utente della piattaforma può tenere traccia delle azioni che esegue durante l'attività di scrittura di codice nell'editor di cui fa uso. A tal fine è necessario che lo sviluppatore installi il plugin di logging per uno degli IDE compatibili (Eclipse, IntelliJ, Atom).
Successivamente potrà collegarsi alla piattaforma del CAS per poter consultare nel dettaglio le statistiche inerenti la sua attività di programmazione.

2.3.2 CAS Client

Per CAS Client si intende un IDE, compatibile con la piattaforma, che mediante l'integrazione con un plugin registra le metriche degli sviluppatori per monitorarne l'attività.
Nel prossimo capitolo descriveremo l'argomento più dettagliatamente.

Capitolo 3

Self-tracking in CAS

3.1 Premessa: self-tracking

Iniziamo col riprendere la definizione formale di self-tracking: *Disciplina di cura di sé, basata sul monitoraggio delle attività quotidiane [...] e sulla quantificazione di ogni singola azione attraverso i più avanzati dispositivi tecnologici, al fine di restaurare, mantenere o incrementare la qualità del proprio benessere vitale* [25]

Esportando questa definizione nel mondo dello sviluppo di software è immediato collegarla a quanto detto finora sulle dashboard. Il self-tracking per uno sviluppatore altro non è se non l'attitudine a monitorare il modo in cui lavora.

3.2 Premessa: plugin

Un plugin è un software esterno ad un' applicazione, realizzata per aggiungere ulteriori funzionalità a quest'ultima. Nel caso in esame si considerano plugin sviluppati per IDE, quindi estensioni che vanno ad incrementare la capacità degli stessi di interagire con:

- **linguaggi di programmazione**: le funzionalità aggiuntive sono syntax highlighting, autocompletion, linter e toolchain specifici;
- **sistemi di versioning control**: accesso a comandi e statistiche repository git direttamente dall'interfaccia dell'editor;
- **strumenti di collaborazione**: con alcune estensioni è possibile condividere il proprio spazio di lavoro in tempo reale con membri del team. [4]

Per quanto riguarda ad esempio Atom, i plugin giocano un ruolo cruciale per la sua versatilità, in quanto le funzionalità che offre di base sono minime - Atom è un editor,

non un IDE - ma grazie ai pacchetti mantenuti dalla community open-source le sue potenzialità vengono notevolmente aumentate rendendolo uno strumento adattabile a qualsiasi esigenza. [3]

3.3 I plugin del CAS logger

Gli applicativi che agiscono come client per il servizio Logger del CAS sono dei plugin per IDE quali Eclipse, IntelliJ ed Atom che hanno lo scopo di tenere traccia dell'attività dell'utente all'interno dell'editor che sta utilizzando per scrivere codice.

Atom-logger, ad esempio, è un'estensione per il noto editor open-source Atom, realizzato in collaborazione con gli studenti C. Caramaschi e S. Propato [5].
Come per ogni pacchetto di Atom è stato creato utilizzando Javascript e Less (CSS). In figura 3.1 è mostrata l'interfaccia del plugin, che contiene grafici indicanti:

- Tempo trascorso dall'inizio della sessione di coding
- Statistiche sulle linee di codice (aggiunte, rimosse, modificate)
- Statistiche sui commenti del codice (aggiunti, rimossi, modificati)
- Statistiche sui file di test del progetto

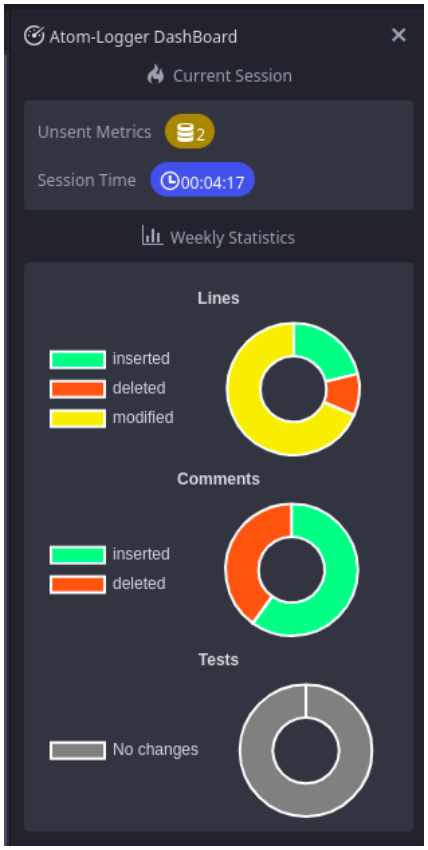


Figura 3.1: Interfaccia del plugin

3.3.1 Tipologia di misurazioni

Di seguito saranno elencati quali sono i dati che vengono tracciati dai plugin, che essendo stati creati per interfacciarsi con lo stesso servizio, raccolgono in maniera omogenea tra loro le metriche:

- **Linee di codice**: mediante l'utilizzo delle rispettive librerie *diff* in Java e JavaScript, i plugin sono in grado di determinare il numero di righe **aggiunte**, **modificate** ed **eliminate** in un dato file ogni volta che viene modificato;
- **Commenti**: similmente a quanto avviene per il codice, per i commenti, che vengono riconosciuti mediante pattern matching, è possibile stabilire in che quantità siano stati aggiunti o eliminati;
- **File di test**: le estensioni sono in grado di stabilire se il file attualmente in modifica, sia un file per il **testing** automatico, nel caso in cui lo sia le modifiche apportate ad esso verranno classificate come modifiche ai test e non al codice sorgente standard;
- **Refactoring**: dialogando con le API messe a disposizione dai rispettivi editor, i plugin *catturano* tutti gli eventi che vengono generati quando l'utente scrive del codice o invoca funzioni di refactoring, determinando la durata ed il tipo di ogni singola operazione. Vengono invece **ignorati** eventi di *cut, paste, copy, backspace, enter, save, undo, redo* perché non considerati significativi dell'attività;
- **Sessione**: un utente autenticato genera delle metriche di attività che comprendono oltre alle grandezze appena descritte, anche dettagli hardware della macchina sulla quale sta lavorando, in particolare indirizzo **IP** e **MAC**, che verranno associati alla tipologia di attività che sta svolegendo.

3.3.2 Installazione

3.3.2.1 eclipse-logger

È reperibile presso http://aminsep.disi.unibo.it/cas-project/latest/plugins/EclipseLogger_1.0.0.201901141717.jar. Una volta scaricato, tramite il menu Help di Eclipse selezionare la voce Install New Software e successivamente indicare il file .jar appena scaricato.

3.3.2.2 atom-logger

In quanto pacchetto ufficialmente pubblicato, per l'installazione è necessario eseguire il comando:

```
$ apm install atom-logger
```

Listing 3.2: Comando shell per installare atom-logger

3.3.2.3 intellij-logger

È sufficiente reperire il file *.jar* nel repository https://gitlab.com/fulvio1993/logger-intellij, ed aggiungerlo nella directory *plugins* di IntelliJ.

3.3.3 Funzionamento

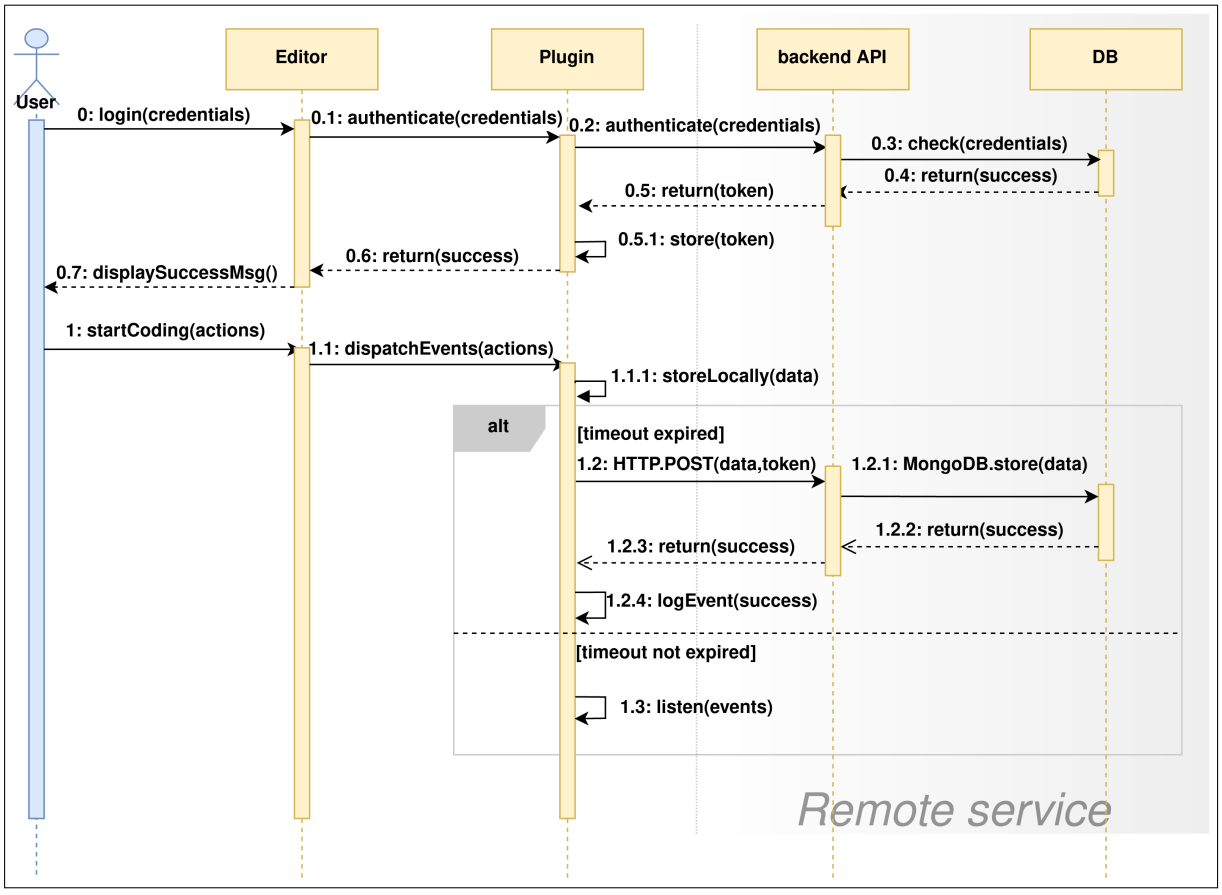


Figura 3.3: Diagramma UML di Sequenza: funzionamento dei plugin

Nel diagramma in figura 3.3 è mostrato in linea generale il funzionamento di un plugin durante l'attività dell'utente e le azioni che intercorrono tra le varie componenti in gioco: *Editor, Plugin, API* e *Database* del back end.

0. **Login**: l'utente avvia l'editor ed inserisce le sue credenziali CAS nel form per autenticarsi, questa fase è identica in tutti e tre i casi;

1. **Inizio del coding**: qui sono presenti delle differenze fra un caso e l'altro; se in Atom ed IntelliJ il plugin inizia a catturare gli eventi senza che l'utente debba fare nulla, quando si utilizza Eclipse è necessario che l'utente inizi la fase di registrazione utilizzando l'apposito pulsante.
   Un'ulteriore differenza è da considerarsi la gestione dei dati appena raccolti (**1.1.1**). Se atom-logger ed eclipse-logger fanno ricorso ad un **database sqlite locale** (fig. 3.4) nel quale conservano i dati prima di inviarli in blocco al server ad intervalli regolari di tempo, intellij-logger invia i dati ogni volta che un'azione viene eseguita.

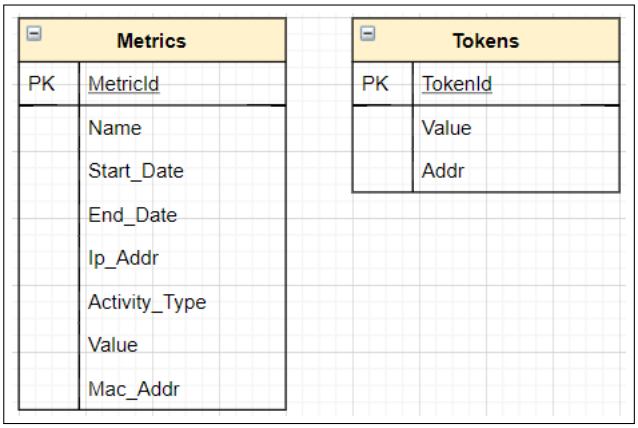


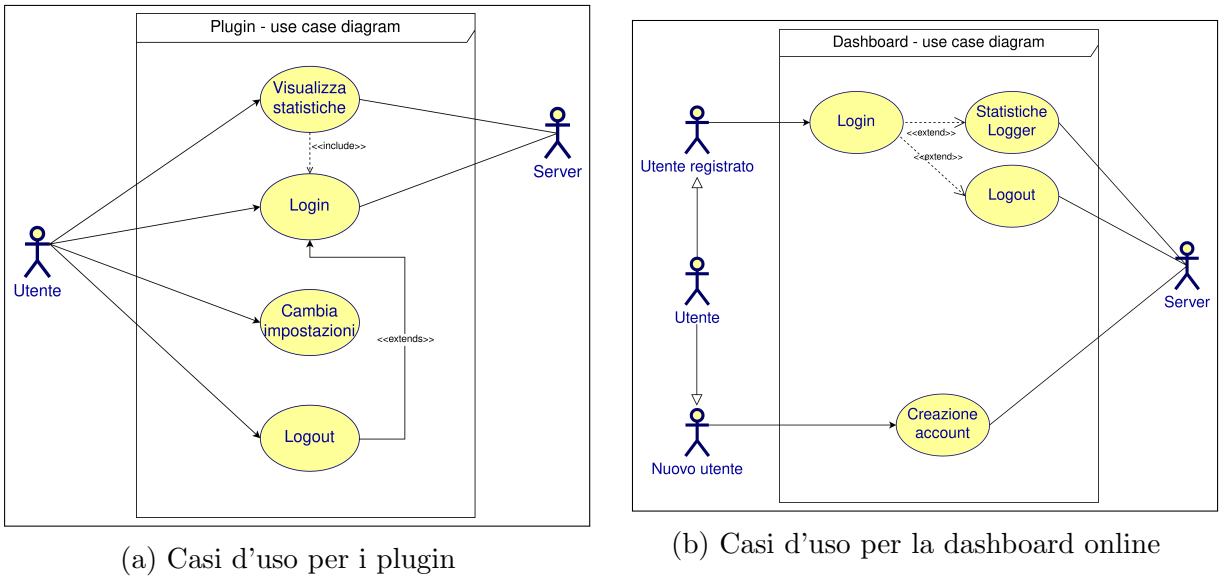
Figura 3.4: Tabelle del database locale di atom-logger ed eclipse-logger

```
CREATE TABLE IF NOT EXISTS METRIC (
ID INTEGER PRIMARY KEY,
name          TEXT         NOT NULL,
start_date    varchar(50)  NOT NULL,
end_date      varchar(50)  NOT NULL,
ip_addr       varchar(50),
activity_type varchar(50)  NOT NULL,
value         TEXT,
mac_addr      varchar(50)
);
```

Listing 3.5: SQL: definizione tabella metriche

```
CREATE TABLE TOKEN (
ID INTEGER PRIMARY KEY,
value varchar(100) NOT NULL,
addr  varchar(100) NOT NULL
);"
```

Listing 3.6: SQL: definizione tabella token



(a) Casi d'uso per i plugin

(b) Casi d'uso per la dashboard online

Figura 3.7: Diagrammi casi d'uso

Una volta installati correttamente, i plugin eseguono il loro compito senza interferire con l'usuale attività dell'utente.

Capitolo 4

Nuova dashboard per CAS

Come oggetto di questo elaborato è stata realizzata una web app che funziona da dashboard per CAS, di tipo operazionale pensata per aggregare i dati di un singolo sviluppatore. Seguendo l'evoluzione della nuova dashboard di Innopolis [7], ho voluto riproporre lo stesso stile minimale, utilizzando elementi grafici di material-ui [15], con i quali avevo già avuto modo di lavorare in precedenza. Le tecnologie impiegate per il front end sono framework Javascript quali **React** e **Redux**, scelti per la loro affidabilità, alta manutenibilità e natura modulare, oltre che per una pregressa esperienza nel loro utilizzo. Per quanto riguarda il back end è stato utilizzato quello già presente sul CAS, innometrics-backend, implementato in **Python** [13].

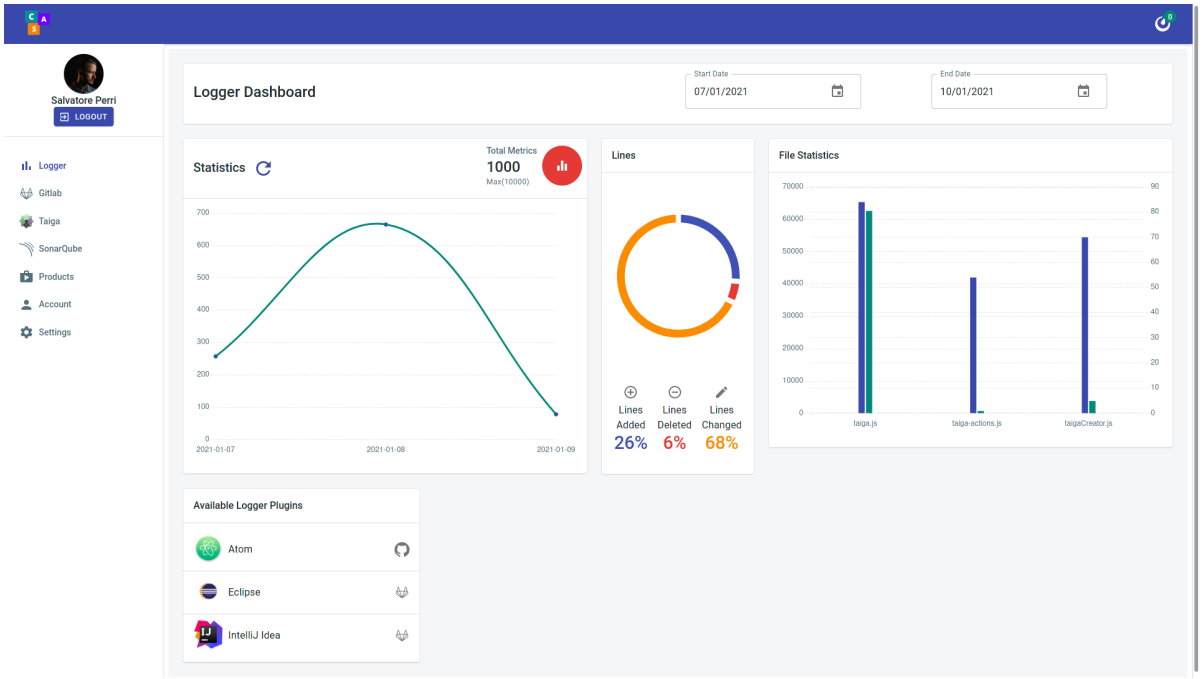


Figura 4.1: Homepage della nuova dashboard

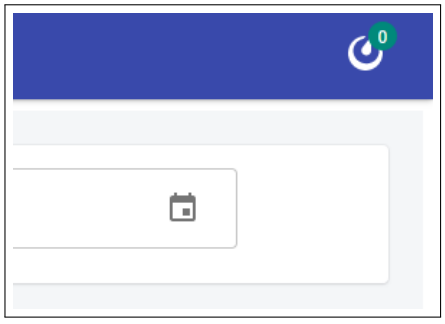
4.1 Panoramica

Essendo stata concepita come *hub* per i servizi CAS, non esiste un unico pannello che raggruppi tutti i dati aggregati, al contrario sono state create diverse sezioni, una per ogni servizio (fig. 4.2a).

I grafici sono stati implementati con ChartJS, grazie al pacchetto npm **react-chartjs-2**, che agisce da wrapper della libreria per React.
Sono state utilizzate le classiche tipologie che includono:

- Grafici a linea
- Istogrammi
- Diagrammi circolari

(a) Menù di navigazione



(b) Area notifiche Mattermost

Figura 4.2: Sezioni UI

4.1.1 Login

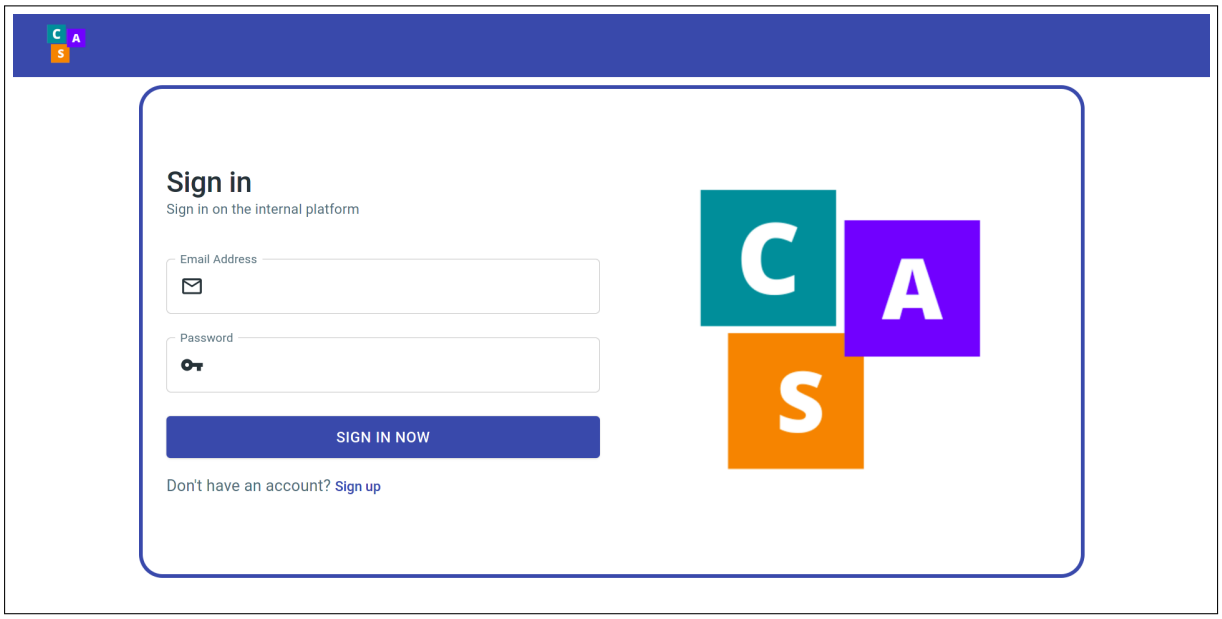


Figura 4.3: Schermata di login

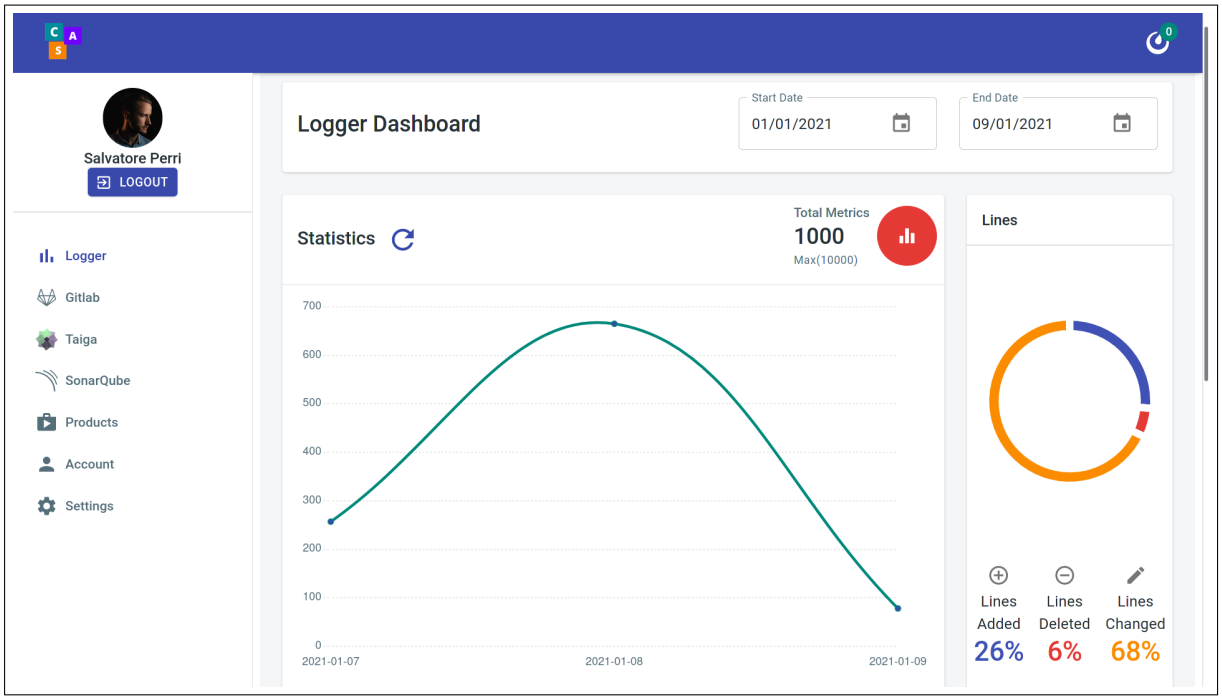
4.1.2 Logger

Una volta che si è autenticato, l'utente viene indirizzato sulla homepage del sito, ossia la sezione del servizio Logger (fig. 4.4).
Sono presenti in 4.4a:

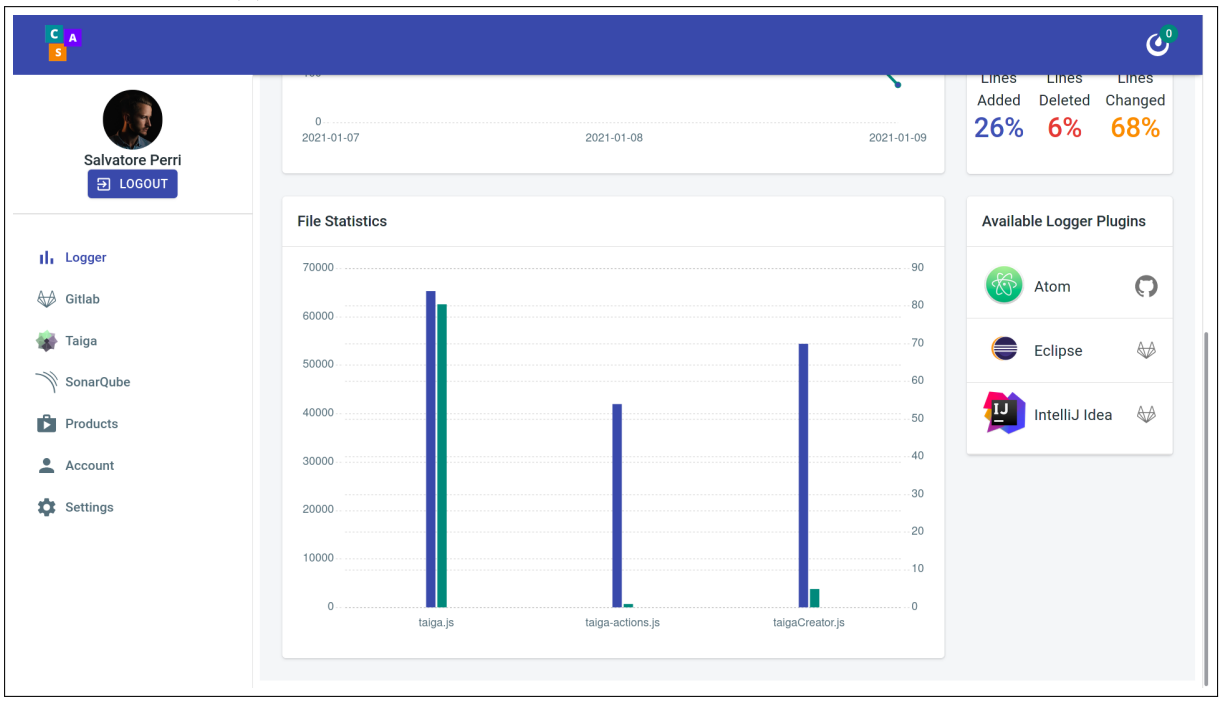
- un widget per scegliere un intervallo temporale;
- un grafico a linea delle azioni registrate dai plugin;
- un diagramma circolare delle statistiche sulle linee di codice.

ed in 4.4b:

- un istogramma che per file individua: durata temporale delle modifiche e numero di azioni;
- dei link utili a raggiungere i repository dei plugin Logger.



(a) Statistiche azioni registrate dai plugin e linee di codice



(b) Statistiche sui file

Figura 4.4: Homepage del sito

4.1.3 GitLab

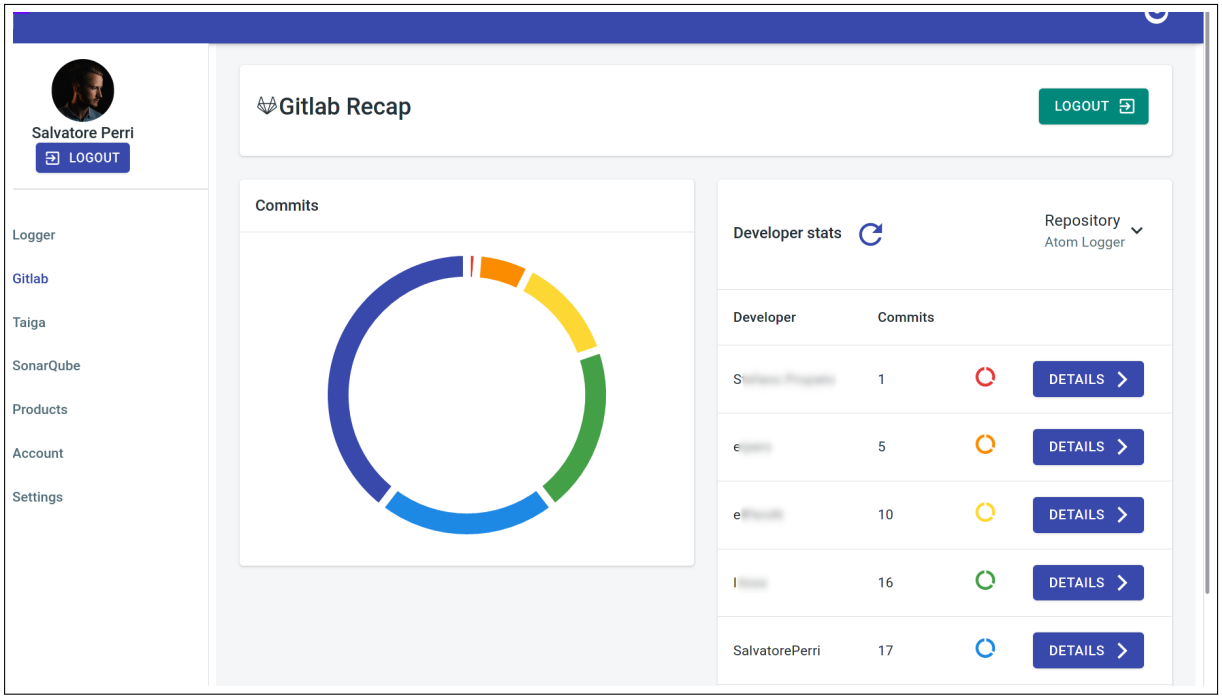


Figura 4.5: Sezione GitLab

Nelle sezione GitLab gli utenti possono analizzare le statistiche dei diversi repository di cui sono collaboratori.
La figura 4.5 mostra:

- un diagramma circolare che rappresenta proporzionalmente il contributo degli utenti al repository in termini di numero di commit;
- una lista dei membri del gruppo, con un link diretto al loro profilo GitLab.

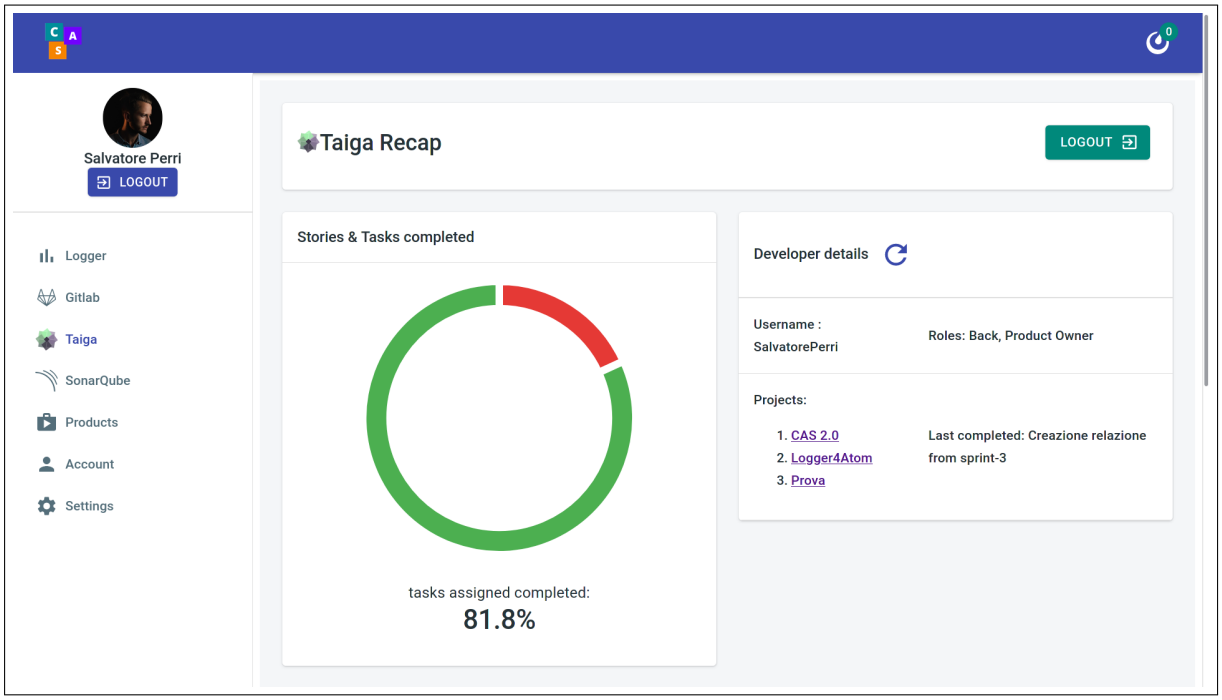
4.1.4 Taiga

Le statistiche raccolte dal servizio Taiga riguardano invece:
4.6a

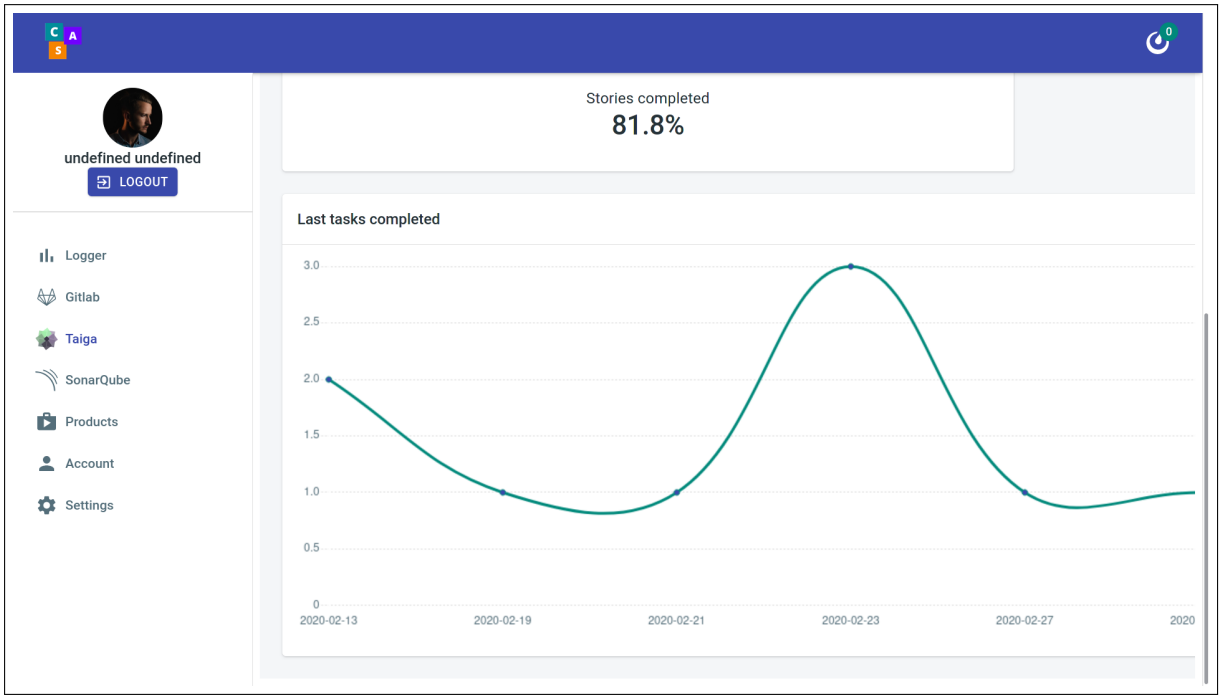
- un diagramma circolare della percentuale di compiti completati (task e user stories);
- ruoli dello sviluppatore nei progetti di cui fa parte;
- link diretti ai progetti

4.6b

- grafico numero di compiti completati giornalmente.



(a) Percentuale compiti completati, ruoli e progetti



(b) Grafico compiti completati in funzione della data

4.1.5 SonarQube

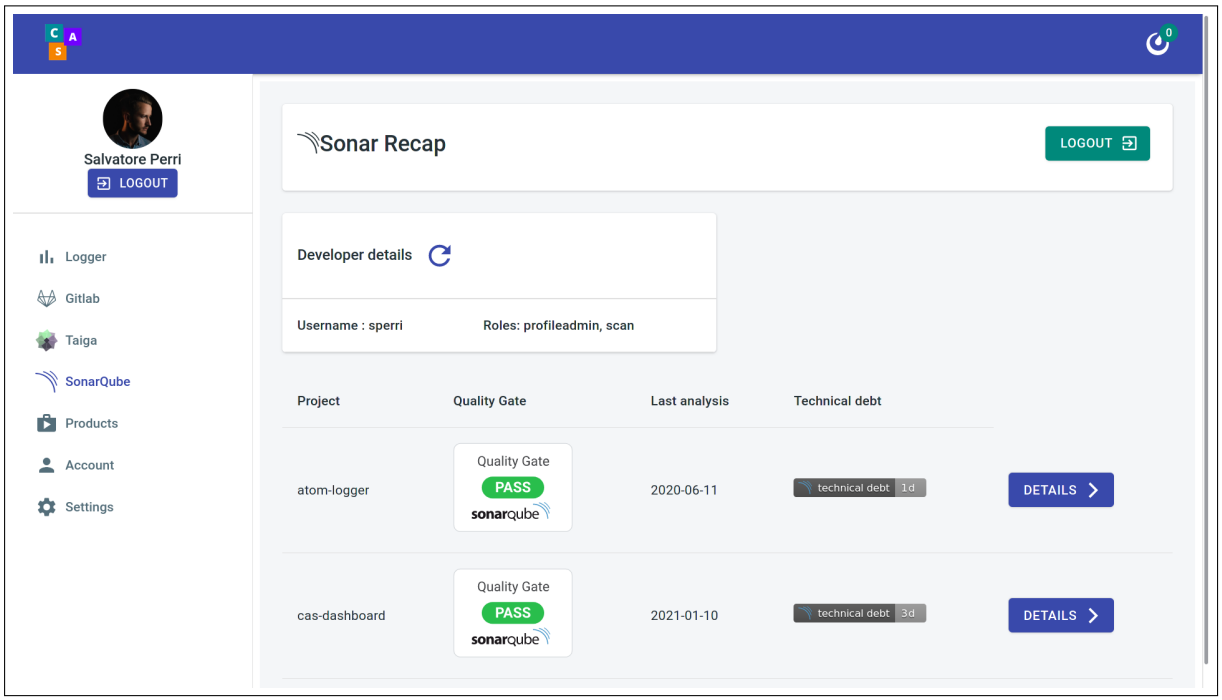


Figura 4.7: Sezione SonarQube

Infine nella pagina di SonarQube vengono riportati i dati di ogni progetto che l'utente ha sottoposto ad analisi:

- nome del progetto
- quality gate di SonarQube
- data dell'ultima analisi
- debito tecnico calcolato
- link alla pagina SonarQube del progetto

4.2 Implementazione

4.2.1 React

React è una libreria javascript open-source [19], mantenuta da Facebook, utilizzata per la creazione di interfacce interattive grazie all'introduzione di JSX (JavaScript XML), che consente di inserire elementi HTML in JavaScript, facilitando di molto il riutilizzo di componenti e la modularità della UI.
Caratteristiche fondamentali di React:

- **Components**: Come detto React consente un alto livello di riusabilità, grazie a JSX ed ai Components. Questi consentono di dividere la UI in molteplici parti indipendenti tra loro. Anche detti componenti funzione, perché per la loro creazione è sufficiente dichiarare una funzione, come avviene in figura 4.8, dove è dichiarato un componente (funzione) TaigaPage che riceve in input un oggetto props, ossia delle proprietà.
- **Conditional Rendering**: In base allo stato dell'applicazione possono essere raffigurate delle parti della UI piuttosto che altre. In figura 4.8 ciò che viene renderizzato dal componente TaigaPage varia a seconda del valore props.token.

```
function TaigaPage(props) {

    let content;

    if (!props.token)
        content = <TaigaLogin/>
    else
        content = <TaigaDash/>

    return (
        <DashPage>
            {content}
        </DashPage>
    );
}
```

Listing 4.8: Esempio di conditional rendering

```
class App extends Component{

render() {
  return (
    <ThemeProvider theme={theme}>
      <ReduxProvider store={store}>
        <Router >
          <DashboardLayout >
            <Switch>
              <Route exact path="/">
                <Redirect to="/logger" />
              </Route>
              <Route exact path='/logger' component={LoggerView}/>
              <Route exact path='/login' component={LoginView} />
              <Route exact path='/register' component={RegisterView}/>
              <Route exact path='/gitlab' component={GitlabPage}/>
              <Route exact path='/taiga' component={TaigaPage}/>
              <Route exact path='/sonar' component={SonarPage}/>
              <Route path='/*' component={NotFoundView}/>
            </Switch>
          </DashboardLayout>
        </Router>
      </ReduxProvider>
    </ThemeProvider>
  );
}
};
```

Listing 4.9: Componente principale della web app

In 4.9 è presente l'implementazione della classe App che costituisce l'entry point dell'applicazione web. Il componente **DashboardLayout** delinea l'aspetto dei vari menù di navigazione, ed ogni Route presente nel tag **Switch** serve a renderizzare la corrispondente sezione (attributo *component*) quando il suo *path* combacia con l'URL corrente.
La figura 4.2a mostra l'aspetto del componente NavBar, che contiene i link utili per spostarsi fra le diverse sezioni della Dashboard, sono presenti quindi i collegamenti ai servizi CAS.

4.2.2 Redux

Redux è una libreria open-source JavaScript usata per gestire lo stato delle applicazioni [21], in questo caso è stata utilizzata insieme a React, con il binding ufficiale React-Redux [20]. Il funzionamento di Redux, ispirato da Flux, CQRS ed EventSourcing, ruota intorno ad un concetto fondamentale: il passaggio tra uno stato e l'altro dell'applicazione avviene secondo regole stabilite dallo sviluppatore, quindi in maniera prevedibile e rigida.

Le componenti fondamentali di un'architettura Redux sono:

- **Store**: letteralmente il magazzino che contiene lo stato dell'applicazione. Per design ne può esistere uno solo. E' un oggetto che espone pochi metodi, tra cui **getState** e **dispatch**, che fungono da getter e setter per lo stato.
- **Actions**: le azioni sono degli oggetti che rappresentano l'*intenzione* di cambiare lo stato, sono l'unico modo per immettere dati nello store. Devono avere un campo **type** che definisce qual è il tipo di azione che si vuole spedire ed ovviamente un campo che contenga l'informazione da trasportare.

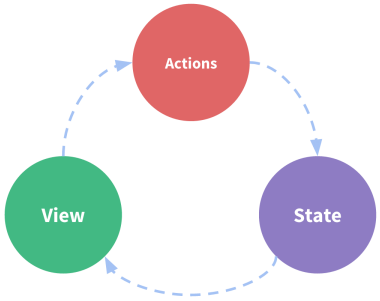


Figura 4.10: Flusso di informazione a senso unico
[21]

- **Reducers**: sono delle funzioni (*riducenti*) che ricevono in input lo stato attuale dell'applicazione ed un'azione, restituiscono come risultato un nuovo stato. Seguendo il comportamento delle funzioni della programmazione funzionale, a cui sono ispirate, lavorano su un valore accumulato (stato) ed uno da accumulare (azione) combinandoli in modo sistematico.

Al componente principale App è passato il riferimento allo store mediante ReduxProvider (listing 4.9, lin. 6). Lo store di CAS-dashboard mostrato nel listing 4.11 contiene gli stati di tutti i servizi, ognuno dei quali collegato al proprio reducer.

```
export const createRootReducer = () =>
  combineReducers({
    app: appReducer,
    logger: loggerReducer,
    gitlab: gitlabReducer,
    mattermost: mattermostReducer,
    taiga: taigaReducer,
    sonar: sonarReducer
  });
```

Listing 4.11: Reducer principale presente nello store

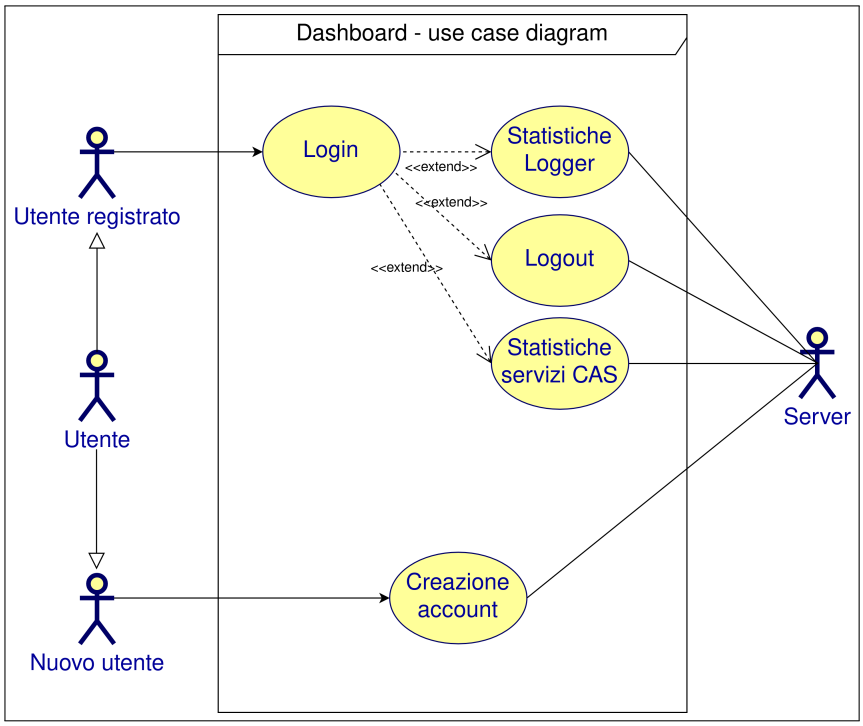


Figura 4.12: Diagramma UML: casi d'uso nuova dashboard

Capitolo 5

Conclusioni

> If you can't measure it, you can't improve it.
>
> *Peter Drucker*

Premettendo che il fine ultimo del self-tracking debba essere quello di rendere consapevoli gli sviluppatori dell'efficacia del loro metodo di lavoro, e non quello di introdurre un determinato livello di controllo sulla loro attività, nell'ultimo capitolo proveremo a spiegare quale sia la filosofia dietro l'impiego di tale pratica.

5.1 Perché tenere traccia delle proprie attività?

Il concetto che Sadowski e Zimmermann esprimono al riguardo è il seguente: se è stato dimostrato che la pratica del self-tracking possa effettivamente aiutare le persone a cambiare il loro comportamento al di fuori della vita lavorativa [9], allora ciò potrebbe valere anche sul posto di lavoro [22].

Nella loro ricerca i due autori hanno realizzato PersonalAnalytics [11], un software di self-tracking che hanno messo a disposizione di alcuni sviluppatori, studiandone l'impatto dell'uso.
Terminato il periodo di osservazione, è risultato che due utenti su tre ritenessero che lo strumento avesse permesso loro di essere più consapevoli di come impiegassero il loro tempo.
In particolare la retrospezione - simile a quella del modello agile, ma in questo caso individuale - secondo l'82% degli intervistati ha permesso loro di confermare o smentire le percezioni personali riguardo il loro metodo lavorativo. Se circa una metà dei soggetti ha ammesso di aver attuato dei cambiamenti basandosi sulle informazioni ricevute, la restante metà non ha effettuato alcun cambiamento, sia perché non lo riteneva necessario, ma anche perché non sapeva come e cosa modificare.

Infine, sebbene si parli di **self**-tracking, aggregare i dati dei membri di un team potrebbe giovare al gruppo, non solo ai singoli. Tenendo conto delle abitudini dei propri colleghi risulterebbe più semplice prendere decisioni più funzionali.

5.2 Problematiche

Senza essere introdotti alla pratica con le dovute spiegazioni, è comprensibile che gli sviluppatori non siano propensi ad adottare gli strumenti di misurazione, in quanto impensieriti dalle eventuali valutazioni delle loro statistiche.
Ciò si è rivelato particolarmente vero quando agli studenti è stato proposto di utilizzare i plugin di logging del CAS, i quali hanno espresso rimostranze temendo per la loro privacy, non avendo chiaro il funzionamento degli strumenti.

5.3 Lavori futuri

L'oggetto dell'elaborato vuole essere una base di partenza sulla quale continuare a costruire. Le eventuali modifiche da apportare sono innumerevoli, dipendono dalla direzione che il progetto CAS intraprenderà in futuro ed eventualmente dalle preferenze espresse dagli studenti stessi. Di seguito alcune proposte di funzionalità aggiuntive da implementare.

5.3.1 Ampliare la portata di CAS-dashboard

Se adesso la dashboard consente solo di visualizzare i dati individuali degli sviluppatori, si potrebbe invece passare ad aggregare dati a livello di team o progetto. Inoltre risulterebbe interessante analizzare i dati a livello della comunità, es. tutti gli studenti della facoltà di Informatica, generando classifiche interne come avviene in WakaTime.

5.3.2 Supporto per altri IDE

Gli IDE già supportati sono sì tra i più utilizzati, ma all'appello mancano strumenti molto diffusi (ed open source) come VSCode, CodeBlocks, NetBeans, AndroidStudio, Vim ed altri. Creare un nuovo plugin per uno di questi renderebbe ancora più compatibile CAS.

5.3.3 Integrazione Jenkins

Recentemente è stata proposta l'aggiunta di un ulteriore servizio al CAS: Jenkins; uno strumento che assiste gli sviluppatori nella cosiddetta integrazione continua. Quando verrà effettivamente installato nell'ambiente, si potrebbe includere nella dashboard anche le sue statistiche.

5.3.4 Nuovo backend per Logger

Come detto il back end del servizio Logger è attualmente innometrics-backend [13]. Così com'è avvenuto con il front end sarebbe opportuno che anche il back end entrasse sotto il controllo diretto del dipartimento, eventualmente creando un fork dall'attuale o realizzando una versione ad hoc. Avendo la possibilità di modificare il database si potrebbe pensare di inserire ulteriori metriche di cui tenere conto.

5.3.5 Migliorare processo autenticazione

Allo stato attuale un nuovo utente che voglia usufruire dei servizi di CAS deve registrare un account su ciascuno di essi.

Rendere sufficiente un unico account, ad esempio implementando un meccanismo quale il single-sign-on o sfruttando le credenziali istituzionali, comporterebbe un notevole miglioramento per la facilità d'utilizzo da parte dell'utente finale.

Bibliografia

[1] AlDanial. cloc github repository. https://github.com/AlDanial/cloc/. (Consultato in data: 01/03/2021).

[2] T. A. Alliance. Manifesto for agile software development. http://agilemanifesto.org.

[3] Atom. Atom flight manual. https://flight-manual.atom.io/. (Consultato in data: 01/03/2021).

[4] Atom. Teletype documentation. https://teletype.atom.io/. (Consultato in data: 01/03/2021).

[5] C. Caramaschi, S. Perri, and S. Propato. atom-logger package page. https://atom.io/packages/atom-logger.

[6] P. Ciancarini, M. Missiroli, F. Poggi, and D. Russo. An open source environment for an agile development model. In *Proc. 16th Int. Conf. on Open Source Systems (OSS)*, volume 582 of *IFIP Advances in Information and Communication Technology*, pages 148–162, Innopolis, Russia, 2020. Springer.

[7] S. Ergasheva, V. Ivanov, I. Khomyakov, A. Kruglov, D. Strugar, and G. Succi. InnoMetrics dashboard: The design, and implementation of the adaptable dashboard for energy-efficient applications using open source tools. In *Proc. 16th Int. Conf. on Open Source Systems (OSS)*, volume 582 of *IFIP Advances in Information and Communication Technology*, pages 163–176, Innopolis, Russia, 2020. Springer.

[8] S. Few. *Information dashboard design : the effective visual communication of data*. Beijing ; Cambridge [MA] : O'Reilly, 2006.

[9] T. Fritz, E. M. Huang, G. Murphy, and T. Zimmermann. Persuasive technology in the real world: a study of long-term use of activity sensing devices for fitness. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014.

[10] Gitlab. Gitlab documentation. https://docs.gitlab.com/. (Consultato in data: 01/03/2021).

[11] HASE-UZH. Personalanalytics github repository. https://github.com/HASE-UZH/PersonalAnalytics. (Consultato in data: 01/03/2021).

[12] W. S. Humphrey. *PSP: a self-improvement process for software engineers*. Addison-Wesley Professional, 2005.

[13] InnopolisUniversity. innometrics-backend github repository. https://github.com/InnopolisUniversity/innometrics-backend. (Consultato in data: 01/03/2021).

[14] D. G. IONOS. Sviluppo agile: di cosa si tratta? https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/sviluppo-agile/. (Consultato in data: 01/03/2021).

[15] MaterialUI. Materialui documentation. https://material-ui.com/. (Consultato in data: 01/03/2021).

[16] Mattermost. Mattermost documentation. https://docs.mattermost.com/. (Consultato in data: 01/03/2021).

[17] S. Pittet. Continuous integration vs. continuous delivery vs. continuous deployment. https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment. (Consultato in data: 01/03/2021).

[18] Processdash. Processdash documentation. https://processdash.com/. (Consultato in data: 01/03/2021).

[19] React. React documentation. https://reactjs.org/docs/react-api.html. (Consultato in data: 01/03/2021).

[20] React-Redux. React-redux documentation. https://react-redux.js.org/. (Consultato in data: 01/03/2021).

[21] Redux. Redux documentation. https://redux.js.org/api/api-reference. (Consultato in data: 01/03/2021).

[22] C. Sadowski and T. Zimmermann. *Rethinking Productivity in Software Engineering*. APress, 2019.

[23] Sonarqube. Sonarqube documentation. https://docs.sonarqube.org/latest/. (Consultato in data: 01/03/2021).

[24] Taiga. Taiga documentation. https://taigaio.github.io/taiga-doc/dist/. (Consultato in data: 01/03/2021).

[25] Treccani. definizione self-tracking.

[26] Wakatime. Wakatime documentation. https://wakatime.com/. (Consultato in data: 01/03/2021).

Appendice A

Il codice sorgente è presente in un repository GitLab pubblico sul server Aminsep al seguente indirizzo: http://aminsep.disi.unibo.it/gitlab/SalvatorePerri/cas-dashboard

CAS dashboard lines of code				
Language	files	blank	comment	loc
JSON	4	0	0	16.184
JavaScript	104	639	98	7.288
SVG	2	0	0	263
Mardown	4	32	0	88
HTML	1	3	0	16
Total	115	674	98	23.839

Tabella 5.1: Statistiche raccolte con il tool cloc [1]

Deployment

NB: nell'attuale stato il server è in modalità di sviluppo non di produzione. Per l'importazione definitiva in CAS sono necessari degli accorgimenti, principalmente è necessario rendere coerenti gli indirizzi IP e numeri di porta dei diversi servizi con quelli reali, mediante il file di configurazione. A titolo informativo si ricorda che nell'attuale versione di CAS i servizi sono collegati ai seguenti numeri di porta:

- Taiga → 80
- SonarQube → 9000
- GitLab → 9022
- Mattermost → 8065
- Logger back end → 8120

Preparazione:

- Utilizzare esclusivamente il sistema operativo Ubuntu Server 20.04;
- Assicurarsi che la versione di NodeJS e npm sia la più recente, per React è necessaria la versione 16.8.6;
- Avviare il MongoDB:

  ```
  sudo service mongod start
  ```

- Avviare il Logger back end:

  ```
  source $INNOMETRICS_BACKEND_PATH/innometricsenv/bin/activate

  python api/app.py
  ```

Procedura per l'installazione ed avvio del server:

1. Clonare il repository:

   ```
   git clone http://aminsep.disi.unibo.it/gitlab/SalvatorePerri/cas-dashboard/
   ```

2. Installare le dipendenze: entrare nella directory appena creata ed eseguire

   ```
   npm install
   ```

3. Avviare il server node:

   ```
   npm run start
   ```

4. L'app è accessibile all'indirizzo localhost:3000

Ringraziamenti

Ringrazio il prof. Paolo Ciancarini per la disponibilità e per avermi permesso di conoscere e partecipare attivamente al progetto dell'ambiente CAS.
Sono grato di aver incontrato in questo percorso Carlos e Stefano, con i quali ho condiviso, anche se a distanza, interessi di studio, ma soprattutto l'esperienza della anomala situazione in cui ci troviamo da un anno a questa parte.
Un enorme grazie alla mia famiglia per avermi sempre supportato e sopportato.
Ringrazio i miei amici Andrea, Barbara, Camillo ed Alfio con cui ho condiviso l'esperienza formativa extracurriculare a tratti traumatica, a tratti comica, più bella di tutte: essere uno studente fuori sede.
Ringrazio Igor, Daniele, Marco e gli altri amici di sempre, per esserci sempre stati, ed anche per avermi spronato, a loro modo.
Infine dedico questo traguardo a mia madre, che so essere fiera di me.