Piattaforma di gestione dei lavori di un Team Scrum

Un'applicazione serverless per l'organizzazione dei compiti relativi a progetti in via di sviluppo all'interno di un gruppo di lavoratori che utilizza il framework agile Scrum

Studentessa: Ilenia Rocca
Corso di studi in ingegneria informatica
Università degli studi di Roma "Tor Vergata"
matricola: 0285151

1. INTRODUZIONE

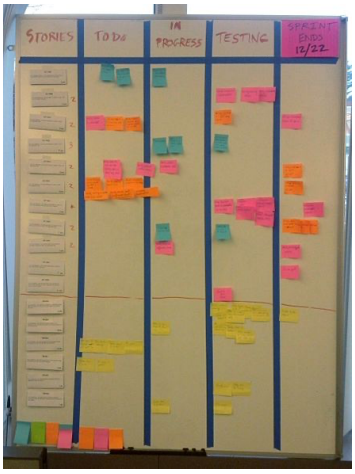
L'idea di questa applicazione nasce dallo studio della metodologia Agile ed in particolare del framework di processo *Scrum.*

Inizialmente si voleva realizzare una semplice bacheca per la gestione dei task all'interno di un'azienda di sviluppo software e non solo, poi si è deciso di rendere l'applicazione più completa e specifica per l'utilizzo da parte di un Team Scrum.

Lo sviluppo in Scrum è organizzato in unità, chiamate *Sprint.* Ognuna ha una durata che va da una a quattro settimane, è preceduta da una riunione di pianificazione e seguita da una di riepilogo. Nel corso di ogni Sprint, il team crea porzioni complete di un prodotto: infatti l'obiettivo di una scelta processuale di tipo agile è proprio quello di avere, in ogni fase dello sviluppo, un prodotto sì parziale, ma finito e testato. Questa metodologia si contrappone a quella tradizionale "a cascata", in cui vengono prima realizzati in maniera molto minuziosa tutti i documenti necessari, si procede con lo sviluppo e solo alla fine di questa fase avviene la verifica dell'effettiva rispondenza ai requisiti tramite i test.

L'insieme delle funzionalità che vengono inserite in un determinato Sprint provengono dal *Product Backlog*, che è una lista ordinata di requisiti. La selezione di quali item verranno effettivamente inseriti nello Sprint a formare lo *Sprint Backlog*, viene effettuata nel meeting di pianificazione.

È proprio a questo punto che emerge la necessità dell'applicazione in oggetto. Per ogni progetto, sarà necessario gestire la lista di funzionalità (dette anche "story") che verranno in seguito suddivise dal team di sviluppo in attività (task), che possono richiedere da un minimo di quattro ad un massimo di sedici ore di lavoro.



Nella foto un esempio di bacheca per la gestione di uno sprint realizzata a mano.

2. L'APPLICAZIONE

L'accesso all'applicazione avviene tramite un login con credenziali (email e password) personali per ogni utente, previa registrazione.
La piattaforma permette di gestire una lista di **progetti**, ad ognuno dei quali è associata una lista di **story**, ad ognuna delle quali è associata una lista di **task**.
Dunque, le entità protagoniste della piattaforma con i relativi attributi sono:

- Progetto (id, titolo, scadenza, descrizione, commenti)
- Story (id, titolo, scadenza, descrizione, progetto di appartenenza, punteggio, stato, priorità)
- Task (id, titolo, story di appartenenza, stato)

Per ciascun oggetto, l'id assegnato ad esso è proprio l'id della richiesta al momento dell'inserimento di quell'oggetto nel database, estratto dal pacchetto http.
Sono possibili inserimento, modifica ed eliminazione di ognuna delle tre entità sopra elencate.

3. PROGETTAZIONE E SVILUPPO

Per la realizzazione della piattaforma sono stati utilizzati diversi servizi cloud di Amazon: Lambda, API Gateway, DynamoDB, Cognito, S3, IAM, CloudFront.
Il frontend è stato implementato in Angular, in ambiente Visual Studio.

3.1 Lambda

Il servizio AWS Lambda ha reso possibile l'implementazione di un backend serverless per l'applicazione. Un approccio serverless per un'applicazione del genere è molto vantaggioso perché dà la possibilità allo sviluppatore di realizzare una piattaforma ospitata senza dover gestire minuziosamente il server; questo arbitrio è lasciato interamente al provider. Uno svantaggio di tale approccio è il fatto che il programmatore non conosce il formato in cui il server si aspetta le richieste ed impacchetta le risposte; pertanto, deve svolgere a priori un lavoro di documentazione su tali aspetti per sapere come inviare e ricevere messaggi.

Le funzioni Lambda sono state scritte in Python 3.9 e sono di seguito elencate in ordine alfabetico.

- deleteBoard: elimina un progetto dato il suo id
- deleteStory: elimina una story dato il suo id
- deleteTask: elimina un task dato il suo id
- getAllBoards: ritorna tutti i progetti nel database
- getBoard: ritorna un progetto dato il suo id
- getStory: ritorna una story dato il suo id
- getStoryByProject: ritorna la lista di story relative ad un progetto dato l'id del progetto
- getTask: ritorna un task dato il suo id
- getTaskByStory: ritorna una lista di task relativi ad una story dato l'id della story
- insertBoard: aggiunge un nuovo progetto
- insertStory: aggiunge una nuova story
- insertTask: aggiunge un nuovo task
- updateBoard: aggiorna gli attributi di un progetto dato il suo id
- updateStory: aggiorna gli attributi di una story dato il suo id
- updateTask: aggiorna gli attributi di un task dato il suo id
- updateTaskName: aggiorna solo il nome del task dato il suo id

3.2 API Gateway

Le API sono state completamente create e gestite tramite il servizio AWS API Gateway. Sono state distribuite tutte le risorse in un'unica fase.

Le risorse istanziate presentano una struttura gerarchica (vedere immagine a sinistra). Tale rappresentazione rispecchia il fatto che ogni task appartiene ad una ed una sola story, ogni story appartiene ad uno ed un solo progetto. L'utilizzo degli id per ciascun oggetto è stato di fondamentale importanza in tal senso.

La principale difficoltà riscontrata in questa fase riguarda CORS. Probabilmente l'abilitazione richiede un certo tempo prima che sia effettivamente attiva e per chi è alle prime armi questo potrebbe essere motivo di confusione. È stata spesa almeno una settimana solo per comprendere questo tipo di problematica.

3.3 DynamoDB e IAM

L'utilizzo di questo database NoSql di tipo chiave-valore e la sua integrazione con Lambda si sono rivelati molto semplici e ben spiegati nella documentazione di AWS. Sono state create tre tabelle, una per ogni entità (*Task, Story, Board*) ed in questa fase è stato introdotto anche il servizio IAM per la gestione dei permessi di accesso alle tabelle in lettura, scrittura, aggiornamento da parte di Lambda.

Per fare questo è stato creato un nuovo ruolo IAM, chiamato *LambdaBoard*, il quale, in seguito all'assegnazione dei permessi per ciascuna tabella di DynamoDB, è stato associato ad ogni funzione lambda per autorizzare così lo svolgimento di operazioni sulle tabelle.

3.4 Cognito

Cognito è un servizio che permette il mantenimento e la gestione di insiemi (pool) di utenti. Tramite l'interfaccia fornita da Cognito è possibile registrate un nuovo utente con email e password ed accedere all'applicazione tramite le credenziali inserite in fase di registrazione. Tutto quello che riguarda lo storage dei dati, l'autorizzazione dell'utente,

l'invio di un token in fase di registrazione e simili, è interamente gestito dal servizio cloud. Le uniche accortezze dello sviluppatore sono quelle di creare un client di app (che tramite un id servirà poi per consentire l'autenticazione a quella specifica app), creare un nome per il dominio di Cognito e specificare i link per il routing rispettivamente a seguito di login e logout.

L'interfaccia mostrata nella piattaforma è quella di default fornita dal servizio, ma c'è la possibilità di modificarla tramite css personalizzato.

3.5 Angular

Il lato frontend dell'applicazione è stato realizzato in Angular. Questa parte dello sviluppo è stata forse la più dispendiosa in termini di tempo, poiché è stata letteralmente la mia prima esperienza con il frontend. Sicuramente ad oggi penso che questo progetto sia stato un'ottima occasione per imparare questo lato dello sviluppo che era completamente assente nelle mie competenze.

Per la stesura del codice Angular è stato utilizzato Visual Studio. La struttura è organizzata come segue:

- un servizio "task.service" per la gestione degli url che vanno a chiamare le API
- un servizio "web-request.service" che si occupa di chiamare i vari metodi http
- un insieme di component:
    - "dashboard.component" offre la vista dei progetti e permette di svolgere operazioni di questi
    - "edit-project.component" offre l'interfaccia per la modifica di un progetto
    - "edit-story.component" offre l'interfaccia per la modifica di una story
    - "edit-task.component" offre l'interfaccia per la modifica di un task
    - "new-project.component" offre l'interfaccia per l'inserimento di un nuovo progetto
    - "new-story.component" offre l'interfaccia per l'inserimento di una nuova story
    - "new-task.component" offre l'interfaccia per l'inserimento di un nuovo task
    - "task-view.component" offre la vista delle story e dei task relativi a ciascuna story e permette di svolgere operazioni su entrambi

Punto fondamentale di questa fase di sviluppo è stato capire in che modo Angular avrebbe dovuto inviare le richieste (formato, numero di parametri ecc) affiché AWS le accettasse e le processasse nella maniera desiderata.

In generale, per aggiornamento ed inserimento vengono inviati gli attributi all'interno di una struttura di tipo json, mentre per i metodi GET e DELETE i parametri vengono passati nell'uri destinato ad API Gateway che poi li va ad inserire nel campo del pacchetto denominato "pathParameters". Nella funzione Lambda basterà poi accedere a questo campo per estrarre i dati necessari per la richiesta.

3.6 S3 e CloudFront

Il servizio AWS S3 è stato destinato alla funzione di hosting. Durante lo sviluppo della piattaforma il frontend è sempre rimasto in locale, ma sembrava consono, per rimanere in linea con i principi insegnati in questo corso e massimizzare l'utilizzo del cloud, rendere l'intera applicazione accessibile da qualsiasi dispositivo o meglio, da qualsiasi computer, in quanto non è stata ancora realizzata la versione mobile della stessa.

Per l'hosting è stato creato un bucket S3 con il nome *serverlessboard* ed è stata abilitata la modalità di hosting. In seguito è stata buildata l'applicazione Angular e caricati i file di build nel bucket.
Un problema riscontrato in questa fase è stato il reindirizzamento da Cognito all'endpoint di S3, in quanto questo utilizza il protocollo http, mentre Cognito consente l'accesso solamente a siti che utilizzano il protocollo sicuro https. Per questo motivo, è stata realizzata una distribuzione di CloudFront per fare da ponte tra l'endpoint non sicuro di s3 ed un endpoint sicuro, realizzato con il protocollo https. Fornendo questo nuovo url a Cognito, l'utente dopo il login viene indirizzato correttamente alla dashboard.

4. SVILUPPI FUTURI

Quella che è stata realizzata è sicuramente una versione alpha di quello che potrebbe essere un prodotto finale. Si potrebbero aggiungere tantissime funzionalità, come l'assegnazione dei task ai membri del team, la gestione tramite app dei ruoli dei vari membri di ciascun team, l'invio di una notifica prima e dopo uno Sprint, la possibilità di inserire degli allegati (ad esempio per caricare il Product Backlog di un progetto).

Un'applicazione di questo tipo sarebbe un supporto dinamico e veloce per la gestione dei lavori nelle realtà che usano Scrum.