DISCORSO RP

15 Maggio 2017

Correzione errori RR - Alessia Bragagnolo (1.45 max)

Buongiorno, siamo il gruppo DigitalCookies.
Oggi vi introdurremo la nostra analisi delle tecnologie e dell'architettura di SWEDesigner. Prima però è opportuno evidenziare le correzioni apportate ai documenti dopo le segnalazioni in sede di Revisione dei Requisiti e agli incrementi effettuati ai documenti, tra i principali:

-ristrutturazione del Piano di Qualifica, che appare ora con struttura più snella e pone maggiore attenzione sugli obiettivi di qualità che il gruppo si pone;

-modifica della pianificazione presentata nel Piano di Progetto, che si prefigge la consegna di una rp min, al fine di permettere ai componenti un maggiore studio delle tecnologie, l'inizio della stesura del documento di definizione di prodotto;

-incremento delle norme di progetto, ampliate nel processo di sviluppo in vista della fase di progettazione e nel processo di verifica con la descrizione delle metriche adottate dal gruppo e nell'individuazione di strumenti per effettuare i test come mocha e karma;

-introduzione del caso d'uso principale UC0 dedicato alla pagina iniziale di SWEDesigner e nuovi casi d'uso intermedi per il caso d'uso di modifica del diagramma attività che risultavano poco fruibili.

Questi hanno portato ad una modifica dei requisiti che sono passati ora da 147 ad 153, di cui sono stati resi facoltativi, in accordo con Zucchetti SpA, i requisiti per l'autenticazione con credenziali GitHub ed è stato invece eliminato il requisito per la latenza di risposta del server il quale ha anche fornito degli spunti interessanti per la visione utente del prodotto.

(tabelline albi grazie)

Librerie e REST - Carlo Sindico (2 max)

Per quanto riguarda la scelta delle tecnologie il gruppo ha deciso di sfruttare i servizi offerti da aws per ospitare SWEDesigner.
Tra le tecnologie scelte dal gruppo lele più significative per la progettazione dell'architettura sono:

-il framework Angular2 per la parte front-end poichè molto utile per la realizzazione di SPA e inoltre fornisce vantaggi per l'identificazione di dipendenze tra le componenti;

- la libreria per la realizzazione dell'editor dei diagrammi GoJs che a differenza di altre presenta il pregio di poter sfruttare elementi di altri tipi di diagrammi a piacere, il che ci è utile per il diagramma delle attività da noi modificato.

- il framework event-driven Node.js per la parte back-end poiché sfrutta l'approccio asincrono e modulare che permette un'organizzazione più semplice.

- Express per gestire il routing delle richieste

- un database NoSql, MongoDb al fine di ospitare i template dei pattern e una libreria base per i giochi di carte che SWEDesigner fornisce e per interfacciarsi con esso sfrutteremo la modellazione di Mongoose per la validazione dei dati inseriti a database.

Abbiamo deciso inoltre di seguire le linee guide dell'approccio REST(nella slide ci vanno i principi), che presenta diversi vantaggi: sfrutta il protocollo HTTP, con operazioni CRUD (nostra tabellina), e il formato dei dati scambiati può variare e in particolare si può usare il formato JSON, a differenza di come avviene in SOAP dove i file devono avere formato xml, funzionalità risulta particolarmente utile per l'utilizzo di GoJs che genera file di estensione json.

MockUp - Albertini Davide (2 max)

Per fornire una linea guida per la progettazione e la realizzazione del prodotto abbiamo creato un mockup per illustrare le principali funzionalità di SWEDesigner presentandolo preventivamente al proponente che ci ha dato la sua approvazione.
cambio slide

Il prodotto è una Single Page Application, quindi accessibile da broswer.
La pagina iniziale propone le possibilità di creare un nuovo progetto o caricarne da locale uno già esistente.
cambio slide
In particolare se viene premuto il tasto di creazione di un nuovo progetto viene richiesto dal sistema tramite una finestra di dialogo di inserire il nome di un nuovo progetto.
cambio slide
Una volta aperto l'utente potrà spostarsi tra le tre schede delle funzionalità proposte.

Nell'editor delle classi a destra abbiamo la possibilità di inserire nuove classi, collegarle tramite relazioni e aggiungere dei commenti tramite drag&drop, mentre a sinistra si richiamare le classi della libreria dei giochi di carte o pattern da noi offerti ed eventualmente applicare filtri di visualizzazione.
cambio slide
La seconda scheda visualizza il diagramma delle attività, da noi rielaborato, nella barra laterale di destra sono presenti dei blocchi di vario tipo da poter trascinare nell'editor per andare a definire il diagramma delle attività di ciascun metodo.
cambio slide
Inoltre è presente un pulsante per la libreria per richiamare i metodi legati ai giochi di carte.
A sinistra invece c'è la lista di tutti i metodi divisi per la classe a cui appartengono con quello attualmente attivo evidenziato in giallo e delle x per eliminarne il contenuto in caso sia sbagliato.
cambio slide
Nell'ultima scheda si può andare a generare il codice, scegliendo il linguaggio target nella destra - che per il momento sarà solamente Java - e poi premendo su GENERA.
Una volta generato a sinistra ci sarà la lista delle classi e si potrà andare a selezionare ciascuna di esse per visualizzare e in caso modificare il suo codice nell'editor nel blocco centrale. Una volta finite le modifiche tramite il tasto ESPORTA a sinistra l'utente può scaricare uno zip contenente tutti i file.
In qualsiasi momento inoltre l'utente tramite l'icona del floppy in alto può salvare il proprio progetto scaricando il file JSON relativo, oppure tornare alla pagina iniziale per aprire un altro progetto premendo l'icona home.

Front-end - Alberto Rossetti (2.30 max)

L'applicazione front-end sarà essenzialmente l'editor di diagrammi UML appena visto e abbiamo deciso di adottare il design pattern MVVM la cui scelta è stata un po' "obbligata" dall'uso di Angular2:
Questo ci ha portato a suddividere i package con la seguente struttura (indicare slide).
A differenza del classico MVC, MVVM delega alcune funzionalità di cui si occuperebbe il controller alla View, ma porta comunque ai vantaggi classici dell'adozione del pattern MVC separando nettamente la rappresentazione interna dei dati dalla loro presentazione.

Andando nel dettaglio:

- il package View si occupa dell'utilizzo degli elementi grafici di GoJs per creare diagrammi personalizzati, offrendo all'utente un'interfaccia con cui interagire;

Al suo interno abbiamo adottato il design pattern abstract factory, con lo scopo di rendere indipendente il sistema dall'implementazione degli oggetti concreti di cui si occupano le sotto classi.

- il package ViewModel che si occupa di regolare le interazioni tra View e Model;
- il package Model contiene la business logic e interagisce con l'applicazione back-end per fornire template e il codice generato su richiesta dell'utente.
  Al suo interno contiene altri tre package: Objects che raccoglie le classi relative agli elementi grafici forniti, Services che raccoglie le classi per la comunicazione con il back-end e Commands.
  All'interno di Commands abbiamo deciso di utilizzare il design pattern command al fine di identificare e isolare la porzione di codice che si occupa di una determinata richiesta, rendendo l'operazione del client variabile senza conoscerne i dettagli.

Back-end- Saverio Follador (2,30 max)

L'applicazione back-end servirà invece per gestire la generazione del codice e la gestione del materiale offerto dalla libreria. Abbiamo deciso di adottare una struttura Three-Tier motivata dalla volontà di rendere indipendente la business logic dal modulo di gestione dei dati persistenti, delegando ad interfacce ben specifiche l'interazione tra i tier (indicare slide).

-Presentation-tier raccoglie e indirizza tramite Express le richieste del client;
-Application-tier raccoglie le componenti logiche dell'applicazione;
-Data-tier gestisce la raccolta dei dati su MongoDb, sfruttando la libreria mongoose.

cambio

Pertanto nel presentation tier, come suggerito dall'uso di Express, si è adottato il design pattern architetturale Middleware al fine di fornire una libreria di funzioni comuni, gestendo le richieste del client.
All'interno del package Middleware viene usato il design pattern comportamentale Chain of responsibility *cambio*, che permette di distinguere con facilità gli oggetti che originano richieste dagli oggetti che si occupano della loro gestione, passando la richiesta da oggetto ad oggetto con chiamate next, finchè non viene trovato l'oggetto destinato a gestirla.

cambio

Il package ApplicationTier contiene
cambio
- il package Error che gestisce gli errori;
cambio
- il package Utility che fornisce lo zip con all'interno i file generati;
cambio
- il package Generator con un'interfaccia chiamata BaseGenerator la quale rappresenta un contratto comune tra tutte le classi che generano codice, *cambio* e rappresenta l'interfaccia Strategy del design pattern Strategy.

L'adozione di questo design pattern permette di definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. Attualmente è presente solo un package contenente tutte le classi necessarie alla generazione di codice Java, ma questa scelta architetturale è stata pensata per permettere di implementare con facilità altri linguaggi.
cambio

L’interazione tra Generator e Utility è poi gestita dalla classe ApplicationController, che si occupa anche di collegarsi a Presentation Tier per ricevere le richieste e ritornare i risultati. Si collega inoltre a Data Tier,*cambio* che è così strutturato, per il recupero di template.

Modello V e Test - Cabrera Christian (2 max)

Per la definizione dei test il gruppo ha deciso di seguire il modello a V, che prevede che ogni attività sia implementabile basandosi sulla documentazione prodotta nell’attività precedente.
La definizione dei test è quindi posta nei primi periodi del progetto, in particolare prima della codifica, e ciò consente di risparmiare un ampio margine di tempo durante lo sviluppo del prodotto.

Definire i test ci aiuta a mantenere e sviluppare la nostra applicazione, portando ad un confronto con il proponente sempre più consapevole del prodotto finale che forniremo.

Nell'attuale stato di sviluppo vengono definiti i test di integrazione, di validazione e di sistema.

- I test di validazione descrivono le funzionalità che il prodotto finale deve avere, al fine di accertarne la conformità rispetto alle attese del proponente;
- I test di sistema hanno il fine di verificare che il comportamento dinamico complessivo dell’intero sistema sia conforme ai requisiti definiti;
- I testi di integrazione hanno il fine di valutare se le varie componenti del sistema software interagiscono tra loro nel modo atteso. La strategia seguita per definire questi test è stata di tipo bottom-up, in modo da poter realizzare il prodotto partendo dalle singole componenti e implementare in questo modo le varie funzionalità in ordine di importanza.

Per effettuare i test abbiamo identificato in particolare due strumenti legati all’utilizzo di JavaScript: mocha e karma.

Mocha è framework che ha molte funzionalità, tra le quali permette l'esecuzione di test asincroni consentendo segnalazioni flessibili e accurate, mentre Karma è un ambiente di testing che permette di automatizzare il processo di esecuzione dei test.

Rischi e consuntivo - Alberto Giudice (2 max)

Per quanto riguarda i rischi riscontrati in questa attività i principali sono stati:

-osticità dell’uso di alcune tecnologie che hanno portato i membri ad impiegare più tempo per l’approfondimento di esse;

-modifiche di natura ingente rispetto a quanto pianificato del piano di qualifica;

-la pianificazione precedente si è rivelata troppo ottimistica;

Per riuscire a mitigare questi rischi il gruppo, come detto ad inizio presentazione, ha cambiato la propria pianificazione per aderire invece ad una RP min. Non abbiamo però cambiato le previsioni di consegna del prodotto che rimane per il 13 Luglio poiché abbiamo adottato un modello incrementale che ci ha consentito di iniziare a lavorare alla definizione di prodotto e di provare ad applicare alcune tecnologie per riuscire a rimanere nei tempi previsti per la prossima revisione.

E il cambio di pianificazione è stato influenzato anche dall’analisi dei processi di documentazione e di verifica, per andare a valutarne il rispetto delle metriche di Schedule Variance e Budget Variance. Durante l’Analisi dei Requisiti di Dettaglio si sono rivelate negative per la stesura del Piano di Qualifica data la mole di lavoro non prevista, mentre in Progettazione Architetturale sono state negative per l’Analisi dei Requisiti per cui non era stato previsto un tempo adeguato per gli interventi correttivi a seguito dell’incontro col proponente. In totale però grazie ad altri documenti con esiti molto

positivi il processo di documentazione si è sempre rivelato entro il range accettabile, mentre quello di verifica è stato ottimale. I livelli SPICE raggiunti dal gruppo sono stati Predicibile per la documentazione e Stabilito per la verifica.

A seguito del nuovo preventivo stilato dal responsabile di periodo l'impegno a persona è restato invariato a 104 persona, riducendo però le ore da responsabile e da verificatore, ritenute eccessive. Sono state introdotte ulteriori ore di Analista per coprire meglio necessità di revisione dell'analisi nei prossimi periodi e quelle da programmatore per poter iniziare il ciclo zero di codifica in contemporanea con la stesura della Definizione di Prodotto. In totale il nuovo preventivo vede un risparmio di 67€ rispetto a quello iniziale, il gruppo ha deciso comunque di non variare il preventivo di 13640€ per ridistribuire i risparmi nel caso in cui attività future lo necessitino.

Grazie per l'attenzione, se qualcuno ha domande può farle ora.