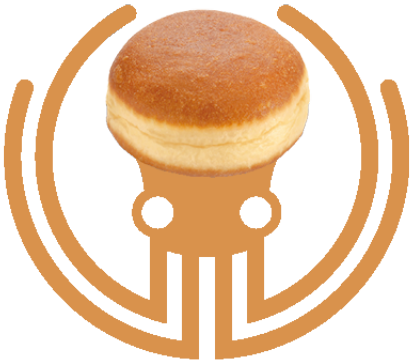

Verbale Interno

Gruppo GitKraffen – ***Progetto IronWorks***

gitKraffen.swe16@gmail.com

Versione	Unica
Redazione	Elia Rebellato
Verifica	Mihai Eni
Approvazione	Mattia Poloni
Uso	Interno
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo GitKraffen

Descrizione

Questo documento riassume le decisioni prese nell'incontro del gruppo GitKraffen del 17 Maggio 2018.

1 Informazioni Generali

1.1 Informazioni incontro

- **Luogo**: Padova, Aula 1BC45 Torre Archimede;
- **Data**: 17 maggio 2018;
- **Ora**: 10:30 - 13:30;
- **Componenti interni**: Federica Ramina, Elia Rebellato, Iris Balaj, Daniel Rossi, Mattia Poloni, Mihai Eni.
- **Componenti esterni**(se presenti): Nessuno.

1.2 Argomenti

Durante l'incontro si discusso delle seguenti cose:

- discussione su quali design pattern usare per l'applicazione;
- utilizzare backbone per realizzare il pattern MVC;
- al server abbiamo deciso three tier (senza data-tier perchè non abbiamo dati persistenti);
- Template method.

2 Riassunto incontro

Il gruppo ha deciso di usare i seguenti design pattern:

- **Pattern MVC**: è stato deciso il patter MVC (model-wiew-control) in quanto risulta essere uno dei pattern più diffusi nello sviluppo di sistemi software, soprattuto nell'ambito della programmazione ad oggetti. Nel nostro caso:
  1. **model**: model e collection estese dalla libreria backbone di Javascript;
  2. **view**: HTML5, CSS3 e i template implementati della libreria Underscore di Javascript;
  3. **controller**: view e router estese dalla libreria backbone di Javascript.
- **Architettura three-tier**: è stata decisa in quanto viene consigliata per le applicazioni web come quella che vogliamo implementare, infatti indica un'architettura software di tipo multi-tier per l'esecuzione di un'applicazione web. La quale prevede la suddivisione dell'applicazione in diversi moduli:
  1. **interfaccia**: che gestisce le richieste dal server;
  2. **application**: che gestice i dati che sarano generati;
  3. **gestione dati persistenti**: nel nostro caso non viene sfrutatto in quanto non abbiamo dati persistenti;
- **Pattern template method**: è stato scelto in quanto uno dei design patter fondamenti per la programmazione orientata agli oggeti come nel nostro caso, ed è costituito da una classe astratta e da più classi concrete che la estendono. Nel nostro caso abbiamo:
  - la classe astratta **Generatore Template** che dovrà definire le operazioni, che classi concrete utilizzeranno e si dividono in:
    1. **operazioni primitive**: metodi che devono essere ridefiniti nelle sottoclassi;

2. **metodi hook**: che possono essere ereditati oppure ridefiniti dalle sottoclassi;
3. **metodi non sovrascrivibili**: metodi che vincolano le sottoclassi ad utilizzare l'implementazione definita nel template.

– le classi concrete derivate:

1. **Generatore Java**: che deve generare il codice Java;
2. **Generatore SQL**: che deve generare il codice SQL;
3. **Generatore XML**: che deve generare il testo XML.