## Assignment 2 - DD

The *Design document (DD)* must contain a functional description of the system, and any other view you find useful to provide. You should use all the UML diagrams you need to provide a full description of the system. Alloy may also be useful, but not mandatory. You will also include information on the number of hours each group member has worked towards the fulfillment of this deadline. As a reference structure for your document please refer to the following one:

### INTRODUCTION

* 1. *Purpose*
  2. *Scope*
  3. *Definitions, Acronyms, Abbreviations*
  4. *Revision history*
  5. *Reference Documents*
  6. *Document Structure*

### ARCHITECTURAL DESIGN

* 1. *Overview*: High-level components and their interaction
  2. *Component view*
  3. *Deployment view*
  4. *Runtime view*: You can use sequence diagrams to describe the way components interact to accomplish specific tasks typically related to your use cases
  5. *Component interfaces*
  6. *Selected architectural styles and patterns*: Please explain which styles/patterns you used, why, and how
  7. *Other design deci*sions

1. **USER INTERFACE DESIGN**: Provide an overview on how the user interface(s) of your system will look like; if you have included this part in the RASD, you can simply refer to what you have already done, possibly providing here some extensions if applicable.
2. **REQUIREMENTS TRACEABILITY**: Explain how the requirements you have defined in the RASD map to the design elements that you have defined in this document.
3. **IMPLEMENTATION, INTEGRATION AND TEST PLAN**: Identify here the order in which you plan to implement the subcomponents of your system and the order in which you plan to integrate such subcomponents and test the integration.
4. **EFFORT SPENT:** In this section you will include information about the number of hours each group member has worked for this document.

### REFERENCES

# Microservizi

* **User Manager**: Integrazione con Sistema Universitario per gestire Autenticazione
  + DB con utenti in base alle info ottenute dall’interrogazione al sistema di auth universitario
* **Website Manager**
  + Restituisce SPA (Single Page App) all’interno della quale sono presenti script js che effettuano chiamate API al API Gateway per ottenere le informazioni.
  + Content Delivery Network hosted by AWS o CloudFlare
* **API Gateway**:
  + Componente Duplicato per Load Balancer essendo stateless (REST?)
  + Gestisce l’interazione tra i vari microservices per evitare che High Coupling tra di loro evitando di introdurre componenti come Circuit Breaker.
* **Platform Manager**: Lista dei tornei, Lista delle battaglie, Teams, Scores
  + DB con Tornei (Replicated State Machine)
* **Notification Manager**
  + Si occupa di inviare notifiche Email, SMS, etc. quando richiesto  
    Notifiche Async con sistema di code interno
* **Badge Manager**:Quando un torneo chiude si occupa di controllare quali badges sono stati soddisfatti.
  + DB con i badges disponibili (Replicated State Machine)
* **Build and Compile Server**: Run dei test (Jenkins con loadbalancer giá implementato)
* **Static Analysis**: SonarCube (loadbalancer giá implementato)

**UC1:**  
Utente chiede al Website Manager HTTP GET la HomePage

Compila tutte le info dei soli badges

Le info dei badges vengono inviate in HTTP Post al API Gateway:

* controlla che il Token dell’User sia valido chiedendo all’User Manager;
* crea i badges inviandoglieli al Badge Manager che restituisce degli Id dei badge creati (se sono formattati correttamente)
* risponde alla POST con gli id dei badges creati

Ora l’Utente:

Compila tutte le info (con anche i riferimenti ai badges esistenti o appena creati, teachers id, info del torneo)  
Tutto viene inviato in HTTP Post al API Gateway  
API Gateway:

* controlla che il Token dell’User sia valido chiedendo all’User Manager;
* prende gli id dei teachers chiedendo all’User Manager (vedendo se esistono)
* prende gli id dei badges chiedendo al Badge Manager (vedendo se esistono)
* invia tutte le info al Platform Manager
* il Platform Manager invia richiesta di notifica creazione torneo al Notification Manager
* Notification Manager effettuerá il pull asynch dalla coda di richieste e poi invierá la notifica
* Risponde alla POST con id del nuovo torneo.

**GitHub Action risposta a una PUSH**

GitHub Action notifica l’API Gateway di una nuova push indicando ID della repo

API Gateway chiede al Platform Manager di chi è la repo e ottiene torneo Id, Battaglia Id e Team Id

API Gateway chiede:

* al Build and Compile Serverdi compilare e runnare i test passandogli solo l’ID della repo  
  ottiene la % dei test passati e non passati
* al Static Analysis di effettuare l’analisi bassandogli solo l’ID della repo

ottiene info sull’analisi del codice

API Gateway invia tutte le info ottenute al Platform Manager il quale si calcola lo score e lo salva.