UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Architecture specification

Project name: Arcane Arcade

Client: Tony vd Linden

Team name: Terabites

Team members

NG Maluleke	D Mulugisi	C Nel	LE Tom
13229908	13071603	14029368	13325095

October 13, 2016

Contents

1 Introduction **2**

2 Vision **2**

3 Architecture Requirements **3**

3.1 Access channel requirements 2

3.2 Quality requirements 2

3.3 Architecture styles 3

3.4 Integration requirements 3

3.5 Architecture constraints 3

3.5.1 Technologies 3

3.5.2 Architectural patterns 4

3.5.3 Architectural Frameworks 4

1 Introduction

The name of our project is Arcane Arcade which references the esoteric language that users will have to use to interact with the system. It also highlights the gamification approach that makes it a fun activity. The platform will be used to test prospective employees in a fun gamified way. It will indicate in which programming area the user is likely to be better suited, by looking at how the user completed the challenges, such as whether the user used any hints and the badges that the user earned. Users will be presented with several challenges which they will have to complete and based on their performance,placed on appropriate positions within the company.

2 Vision

The vision is to provide a fun and easily accessible platform that can be used to gauge the programming aptitude and capabilities of existing or future software developers using a custom esoteric programming language

3 Architecture Requirements

3.1 Access channel requirements

The system requires two interfaces:

- The user will have access to the system through Web interface (for both the game and maintenance portal) on any preferred browser.
- The system will also be made available to the user on a Tablet application.

3.2 Quality requirements

The following quality aspects needs to be addressed:

- **Perfomance:** Performance is the most important quality requirement for this application. This requirement pertains to how fast the application responds to certain actions within a set time interval. The application will use a compiler instead of an interpreter which will translate the esoteric language into object code, which performs better than an interpreter.
- **Reliability:** Reliability is the second most important requirement and will be achieved by implementing user validation on inputs so that they can't submit unexpected values. Fault tolerance is also needed, such as rolling back changes when errors occur.

- **Maintainability:** Maintainability can be achieved by properly documenting the source code. Loose coupling will make it easy to completely change the esoteric language (by simply changing the ANTLR grammar file and the parser) and the front end of the program as long as it conforms to the specified API calls.
- **Scalability:** The system should be able to service at least one administrator and a few users (potential employees), and thus it is required for processes to be able to execute concurrently, which the GlassFish server enables.
- **Security:** Only authenticated API calls should be allowed and user details should be properly encrypted.
- **Cost:** Since some users will be accessing the application via mobile internet, low bandwidth communication is essential. Thus, communication will be done via JSON instead of XML.

3.3 Architecture styles

3.4 Integration requirements

The system will be primarily used via a web interface which will interact with the server through a RESTful API, decoupling the client from the server as best as possible.

3.5 Architecture constraints

3.5.1 Technologies

- HTML
- JavaScript
- CSS
- JQUERY
- Java Standard Edition (J2SE)
- PostgreSQL for persistence because it is a mature, efficient and reliable relational database implementation which is available across platforms and for which there is a large and very competent support community.

  PostgreSQL satistiïňĄes the Scalability requirement rather nicely. It can handle a variety of diïňĂerent workloads, from single machine applications to much larger web applications with many concurrent users. This means that the program can be easily used to ensure that our sysstem can scale to the workload that is required of it.

3.5.2 Architectural patterns

- **Model-View-Controller** is the main architectural pattern used. The *model* refers to the component on the server which manages the data and logic of the application, such as the web server and compiler. The *view* refers to the output on the front-end of the application, ie. on the web and mobile interface. The *controller* is the part which accepts the user input and changes it into the appropriate commands, or API calls between the model and the view components.

3.5.3 Architectural Frameworks

- **ANTLR 4**
  ANTLR 4 is used to generate a lexer and parser for the esoteric language using a grammar file as input. A visitor implementation is also created to deal with the tokens created from an input string (the esolang code) and calculate the result of the user's code.
- **AngularJS**
  Angular js is used for dependency injection. It allows us to speficify fields into which data from the database can simply be placed without the need to explicitly know the data or.
- **Gson**
  Gson will be used for the serialization and deserialization of Java objects JSON. It aids in creating JSON objects that encapsulate all the fields and relations that the Java object we are serializing has.
- **Java Jersey**
  Java Jersey will be used for mapping a resource classes (POJOs) as a web resources. It allows us to use annotations when creating web services according to the Representational State Transfer (REST) architectural pattern.This framework is considerably useful in our situation as we are using both Java and a RESTful API approach to making the system.