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

August 22, 2016

Contents

1 Introduction

Our project *Arcane Arcade* has a vision to provide a profoundly manner of testing the abilities of developers, who are potential employees, and using the results of the captured data from these tests to be able to assign them accordingly to a specific postion within the company which is best suited for their skills.

Since the core focus is on software developers, *Arcane Arcade* does testing on the programming abilities of the candidates using an esoteric language (esolang). Candidates are provided with a series of programming challenges and questions which they have to complete using the esoteric language.

2 Architecture Requirements

2.1 Access channel requirements

The system requires two interfaces:

- Web interface (Maintenance portion)
- It is preferred that the game portion be built to run on a tablet device

2.2 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 quality requirement. Care should be given to make the application as reliable as possible without sacrificing performance. The application should have maximum uptime and proper fault tolerance. The application should help the user in avoiding errors, such as submitting incompatible values.
- **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.

2.3 Architecture styles

2.4 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.

2.5 Architecture constraints

2.5.1 Technologies

- HTML and JavaScript for the front end functionality for both the browser and game application
- Java Standard Edition (J2SE) will be used for backend functionality.
- 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.

2.5.2 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.

2.5.3 Architectural Frameworks

- **AngularJS**
- **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.