UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Software Requirements 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 20, 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 is the third most important quality requirement for the application. Since the future of the application may require total change of the esolang, developers should be able to easily and relatively quickly change aspects of the functionality the system provides or be able to add new functionality to the system. This means that care should be given to modularity and in making the system clear to understand for future developers. Technologies to be used should also be expected to be available for long.

- **Scalability:** The system should be able to service at least one administrator and a few users (potential employees).
- **Security:** The system should be secure enough so that no unauthorized users will be able to access it and data integrity should be maintained.
- **Cost:** Since some users will be accessing the application via mobile internet, care should be given to keep the required bandwidth to a minimum.

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/frameworks

Three-tier Architecture The main reason for using three-tier architecture is to allow any of the three tiers to be upgraded or replaced independently, when changes in requirements or technology require such upgrades or replacements. Thus addressing **maintainability** of the application. By having a dedicated application-tier which can run on a capable server, **performance** is also addressed since the client side is freed from the more demanding computations such as the compilation of the user code.

The three tiers are:

- **Presentation-tier:** This will be represented the front-end of the management or maintenance portal which will communicate with other tiers by sending results to the browser and other tiers in the network.

- **Application-tier:** This is the logical tier which provides the application's functionality.
- **Data tier:** All the data persistence mechanisms which will be used. Data in this tier is kept independent of application servers or business logic.

Frameworks

- AngularJS
- REST (Representational State Transfer)

3 Functional requirements and application design

3.1 Use case prioritization

3.1.1 Critical

- Add user
- Remove user
- User Authentication[Login/Logout]
- Change keywords usecases
- Manage challenges(Add/Remove/Change Challenge)
- View challenges
- Quit challenge
- Start challenge
- Web interface for both maintenance and game

3.1.2 Important

- View score
- Tablet Interface

3.1.3 Nice to have

- Buy hints
- Email score

3.2 Required functionality

3.2.1 User interaction

- The user may log in to the system if registered.
- User may view available challenges.
- User may choose a challenge to do.
- User may receive a hint for the active challenge, if the user can afford it.
- After doing a challenge, the user can then send his/her code to be compiled.
- The user can finish a challenge if his/her code compiled and returned correctly.

3.2.2 Admin interaction

- The administrator may add or remove users from the system.
- The admin may change esolang keywords.
- The admin may add, remove or change challenges.

3.3 Use case/Services contracts

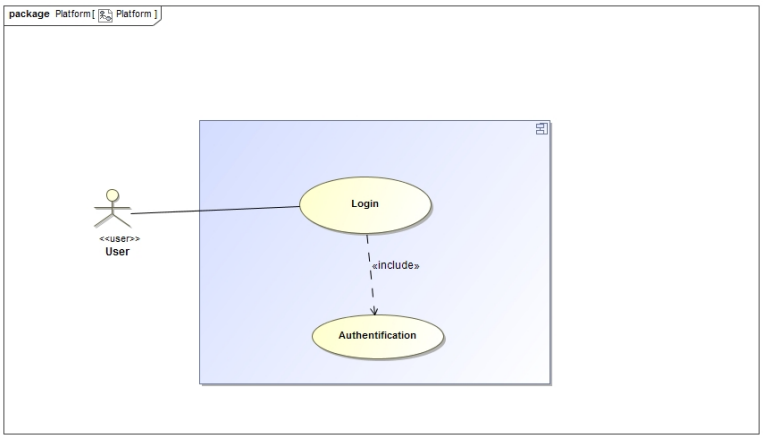
Use Case and Service Contracts are described below.

3.3.1 The Login

- Pre-Conditions
  1. The user is an admninistrator
  2. The user is a player
  3. The login credentials are valid
- Post-Conditions
  1. The user is successfully logged into the system
- Service Contract

The login use case diagram

- Description
  This use case will be used by the REST clients, specifically mobile app(tablet app), to login into the system.

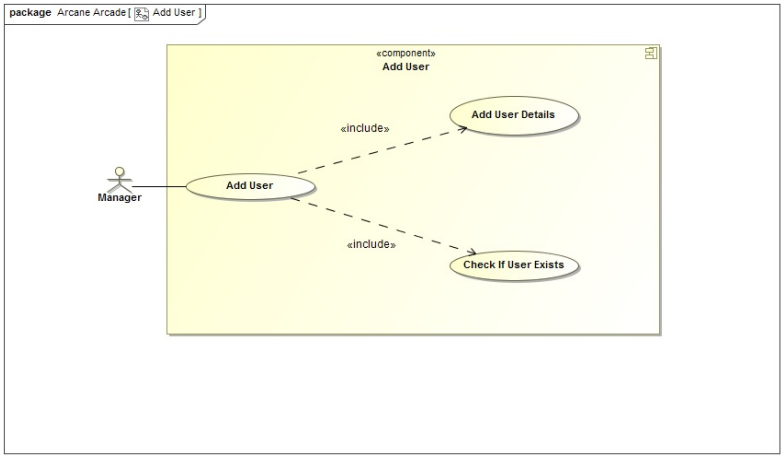


3.3.2 The Add User

- Pre-Conditions
  1. User must be an administrator
  2. No user with the same email exists.
- Post-Conditions
  1. User is added to the system
  2. User(Player) is emailed a link with the username
- Service Contract

The add user use case diagram

- Description
  This use case will be used by the REST clients, specifically the web client since the maintenence portal will be web based, to add a user.

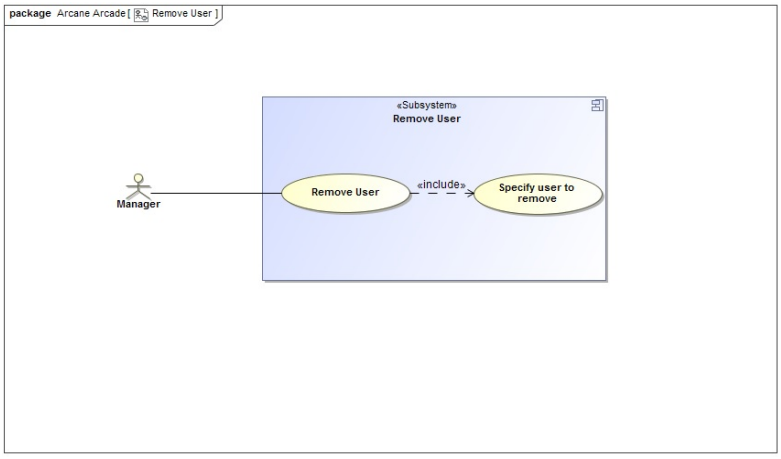


3.3.3 The Remove User

- Pre-Conditions
  1. User must be an administrator
  2. User(player) must exist.
- Post-Conditions
  1. User is removed from the system.
- Service Contract

The remove user use case diagram

- Description
  This use case will be used by the REST clients, specifically the web client since the maintenence portal will be web based, to remove a user.

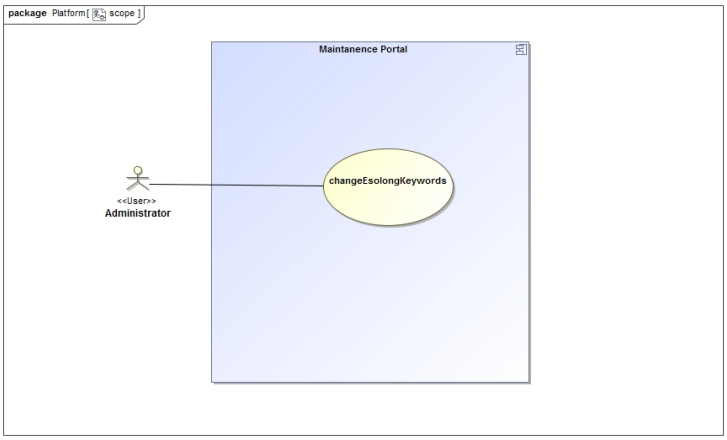


3.3.4 The Change Keywords

- Pre-Conditions
  1. User must be logged in as an administrator
- Post-Conditions
  1. Keywords changed
- Service Contract

The change keywords use case diagram

- Description
  This use case will be used by the REST clients, specifically the web client since the maintenence portal will be web based, to change keywords on the system.

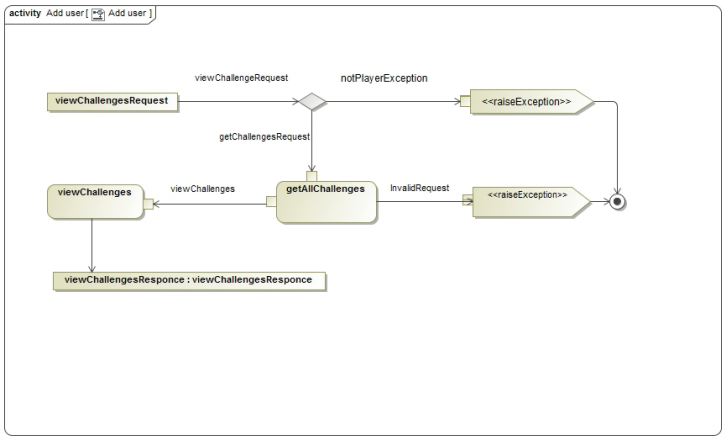


3.3.5 The View Challenges

- Pre-Conditions
  1. User must be logged in as player
- Post-Conditions
  1. All the available challenges are displayed on the system
- Service Contract

The view challenges use case diagram

- Description
  This use case will be used by the REST clients, specifically the mobile app(tablet app), to view all available challenges on the system.

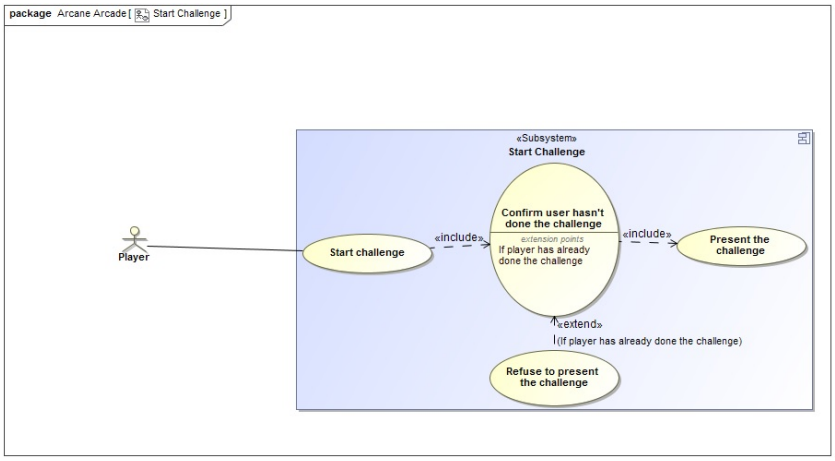


3.3.6 The Start Challenge

- Pre-Conditions
  1. User must be logged in as a player
- Post-Conditions
  1. The challenge is started and timed
- Service Contract

The start challenge use case diagram

- Description
  This use case will be used by the REST clients, specifically the mobile app(tablet app), to start a challenge on the system.

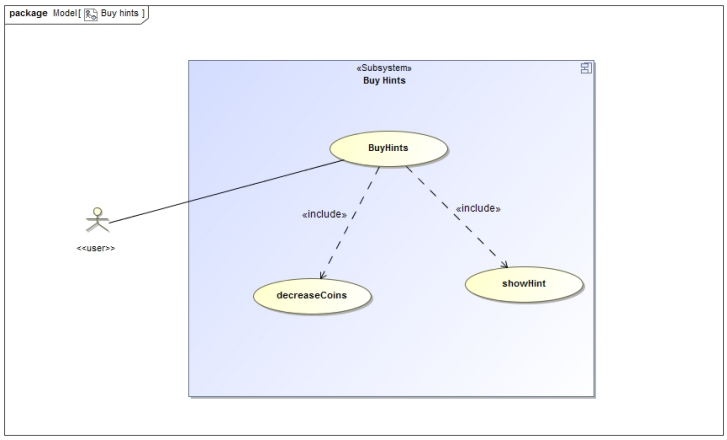


3.3.7 The Buy Hints

- Pre-Conditions
  1. User must be logged in as a player
  2. Player should have started the challenge
- Post-Conditions
  1. A hint is bought and displayed to assist the player, while player coins are decreased.
- Service Contract

The buy hints use case diagram

- Description
  This use case will be used by the REST clients, specifically the mobile app(tablet app), to buy hints during gameplay.

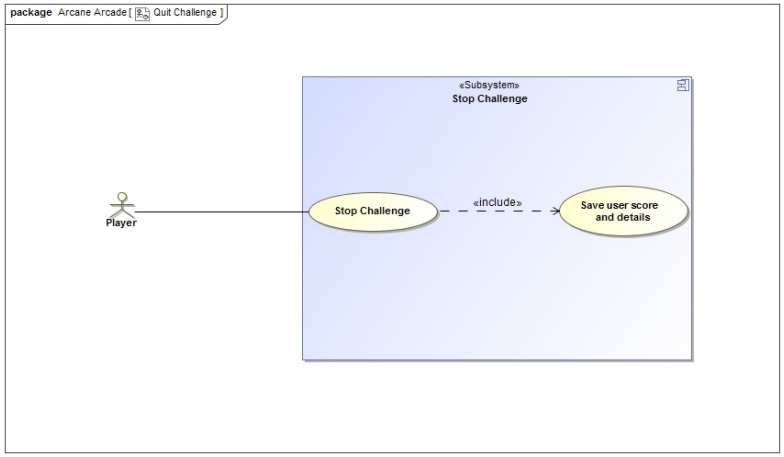


3.3.8 The Quit Challenge

- Pre-Conditions
  1. User must be logged in as a player
  2. Player should have started the challenge
- Post-Conditions
  1. Challenge is stopped.
- Service Contract

The quit challenge use case diagram

- Description
  This use case will be used by the REST clients, specifically the mobile app(tablet app), to quit the cahllenge during gameplay on the system.

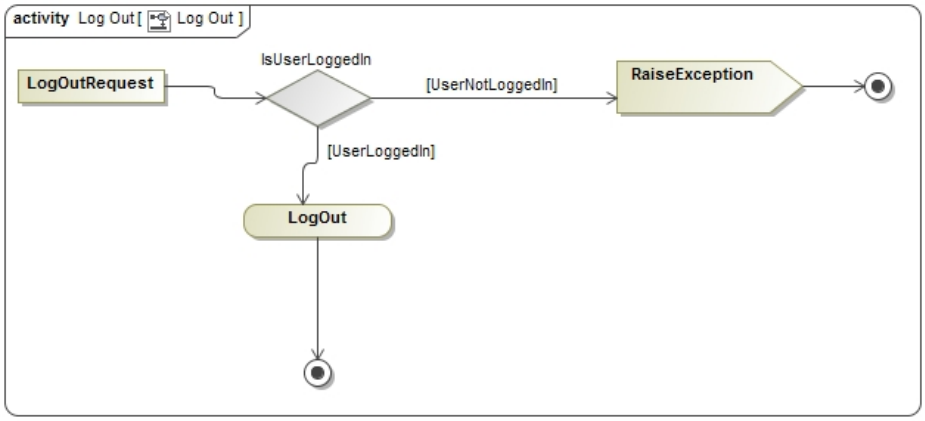


3.3.9 The Logout

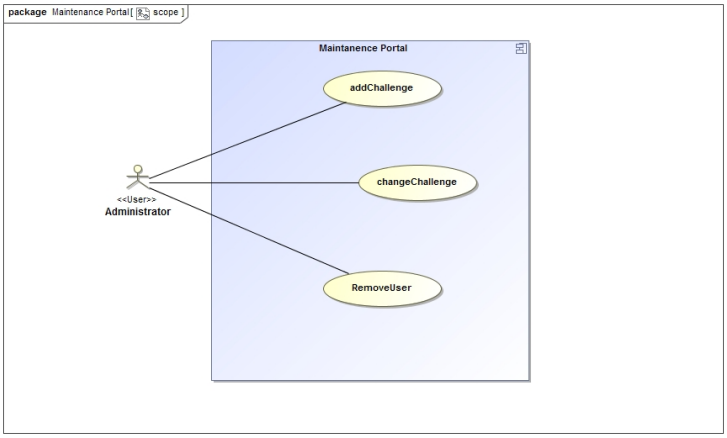
- Pre-Conditions
  1. The user is succesfully logged into the system
- Post-Conditions
  1. The user will be logged out of the system
- Service Contract

The logout use case diagram

- Description
  This use case will be used by the REST clients, specifically mobile app(tablet app), to logout of the system.

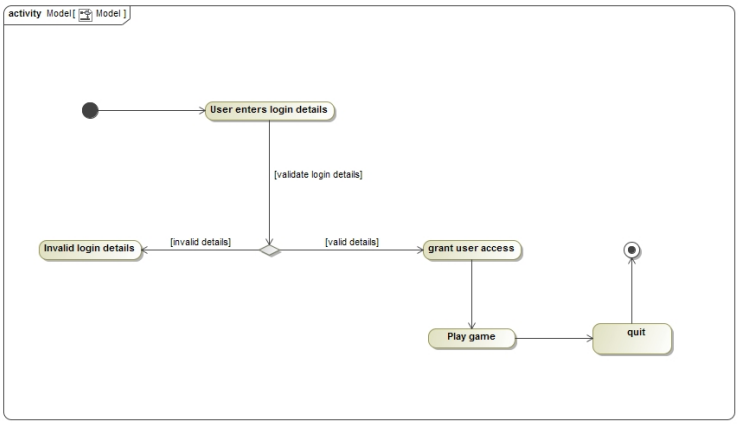


The manage challenges usecase.

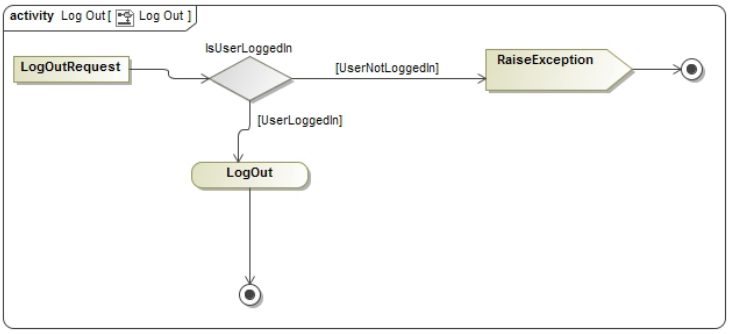


3.4 Activity Diagrams

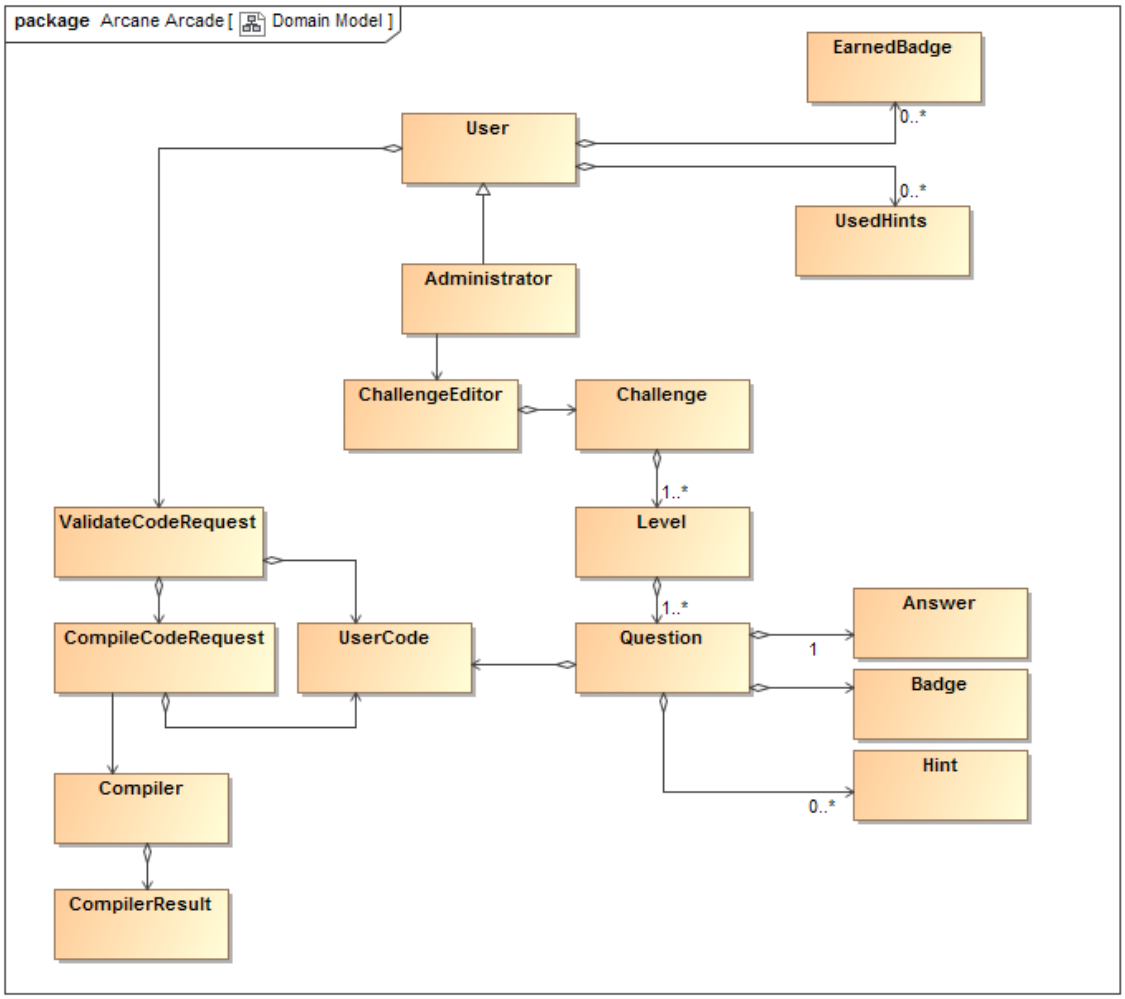
The Login activity diagram.



The LogOut activity diagram.



3.5 Domain Model



4 Open Issues