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

May 23, 2016

Contents

1 Introduction **2**

2 Vision **2**

3 Background **2**

4 Architecture Requirements **3**
- 4.1 Access channel requirements . . . 3
- 4.2 Quality requirements . . . 3
- 4.3 Integration requirements . . . 3
- 4.4 Architecture constraints . . . 4
  - 4.4.1 Technologies . . . 4
  - 4.4.2 Architectural patterns/frameworks . . . 4

5 Functional requirements and application design **5**
- 5.1 Use case prioritization . . . 5
- 5.2 Use case/Services contracts . . . 5
- 5.3 Required functionality . . . 5
  - 5.3.1 User interaction . . . 5
  - 5.3.2 Admin interaction . . . 5
- 5.4 Process specifications . . . 5
  - 5.4.1 Use-Case Diagrams . . . 6
  - 5.4.2 Activity Diagrams . . . 11
- 5.5 Domain Model . . . 11

6 Open Issues **12**

1 Introduction

The project is called *Arcane Arcade*, which references the esoteric language users will have to use, as well as the gamification approach to try and make it as fun as possible.

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 Background

There are many models in which to ascertain whether a prospective employee has the required skills or aptitude to be a valuable software developer. Online assessments are easy to execute, but lack the insight provided by manual and proprietary testing of how the candidate reasons. Manual and proprietary testing on the other hand is time-consuming and expensive to execute. In addition, company proprietary tests become stale and are leakined into the industry reducing the value tests may have.

BBD has thus opted to use an esoteric programming language in order to test the skills and aptitude of prospective employees, regardless of their programming experience or preferred programming language. A mobile and online platform is thus required to test prospective employees while keeping the tests fun by using gamification principles. The tests should also indicate in which 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.

4 Architecture Requirements

4.1 Access channel requirements

The system requires two interfaces:

- Web interface
- Mobile app (Could also be a mobile friendly website)

4.2 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.
- **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.
- **Reliability:** The system should be reliable in all functions, especially when compiling the user's code in order to avoid wrong results.
- **Maintainability:** Maintainability is amongst the most important quality requirements 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 add new functionality to the system. This means that care should be given to modularity.
- **Cost:** Since some users will be accessing the application via mobile internet, care should be given to keep the required bandwidth to a minimum.

4.3 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.

4.4 Architecture constraints

4.4.1 Technologies

- HTML and JavaScript for the front end functionality for both the browser and game application
- Java 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.

4.4.2 Architectural patterns/frameworks

4.4.2.1 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** the application.

- **The three tiers in a three-tier architecture 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.

4.4.2.2 Authentication Enforcer pattern This pattern will help us improve security of the system while keeping the scalability and performance well maintained since the Authentica tion Enforcer pattern provides a consistent and structured way to handle authentication and verication of requests across actions within Web-tier components and also supports MVC architecture without duplicating the code.

5 Functional requirements and application design

5.1 Use case prioritization

The Use Case Prioritisation is specified for each use case in the next section.

5.2 Use case/Services contracts

Use Case Prioritisation and Service Contracts are described below.

5.3 Required functionality

5.3.1 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.

5.3.2 Admin interaction

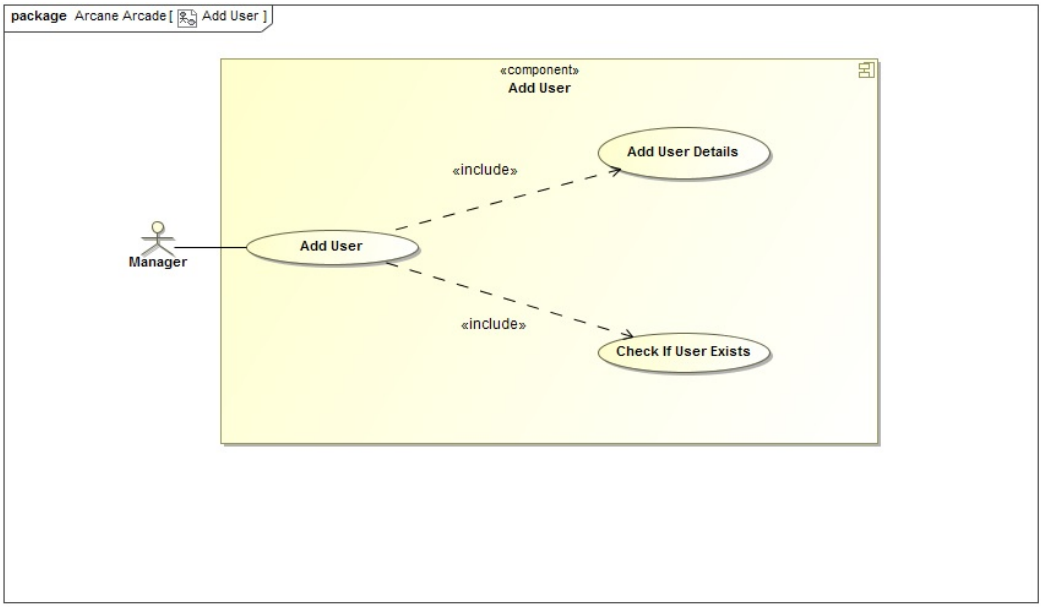
- The administrator may add or remove users from the system.
- The admin may change esolang keywords.
- The admin may add, remove or change challenges.

5.4 Process specifications

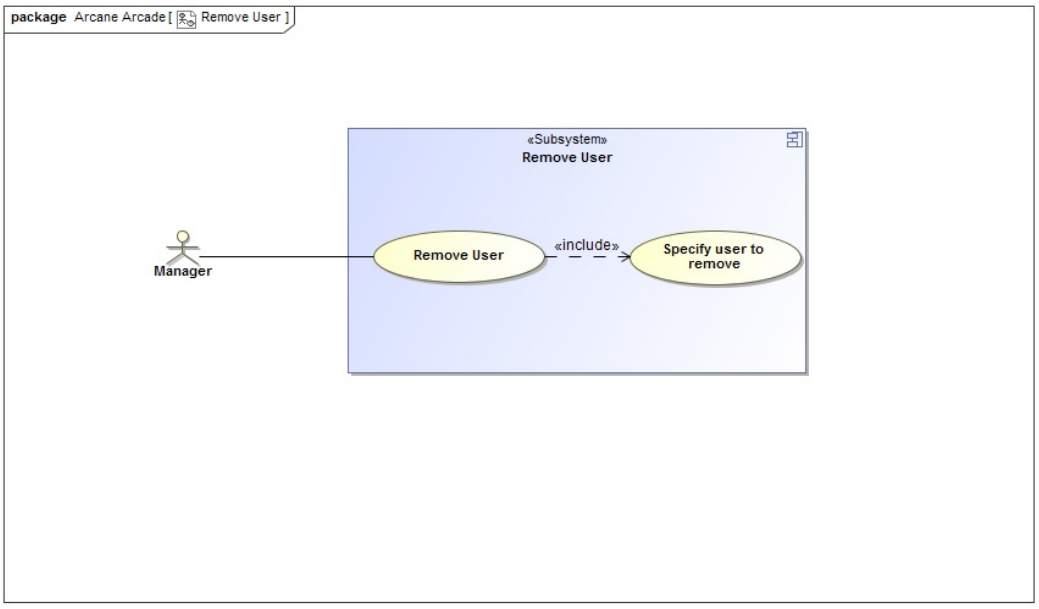
5.4.0.1

5.4.1 Use-Case Diagrams

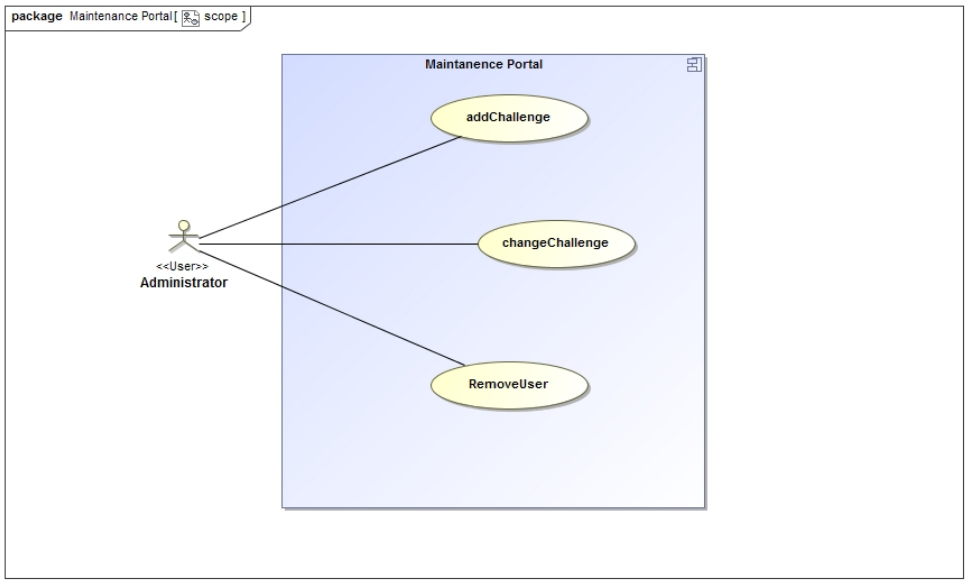
The add user usecase.



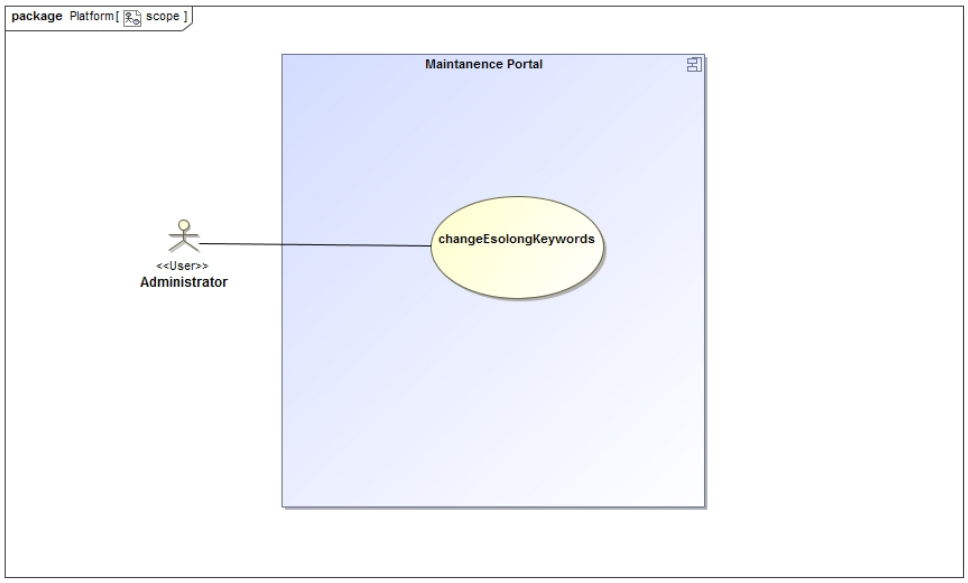
The remove user usecase.



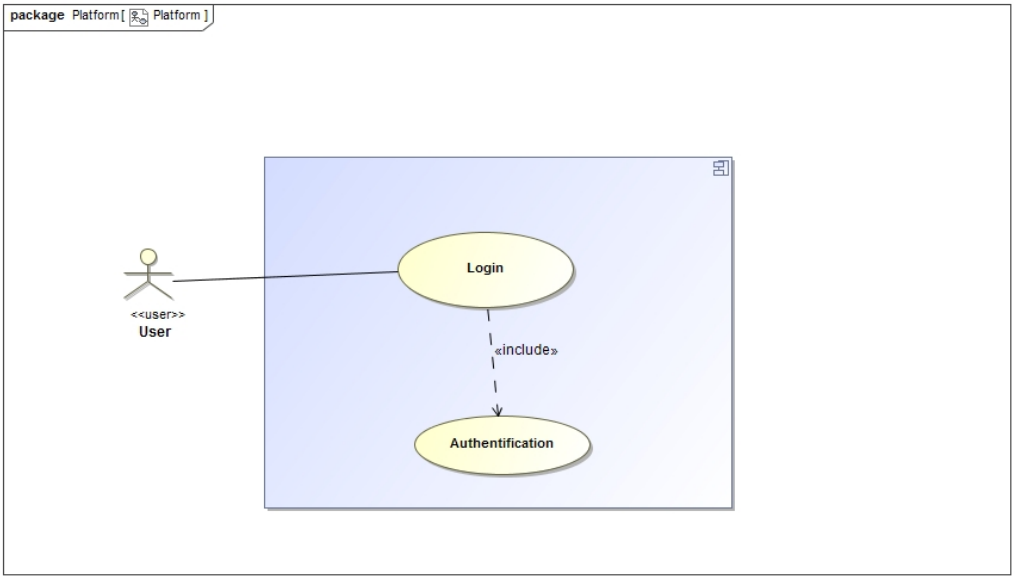
The manage challenges usecase.



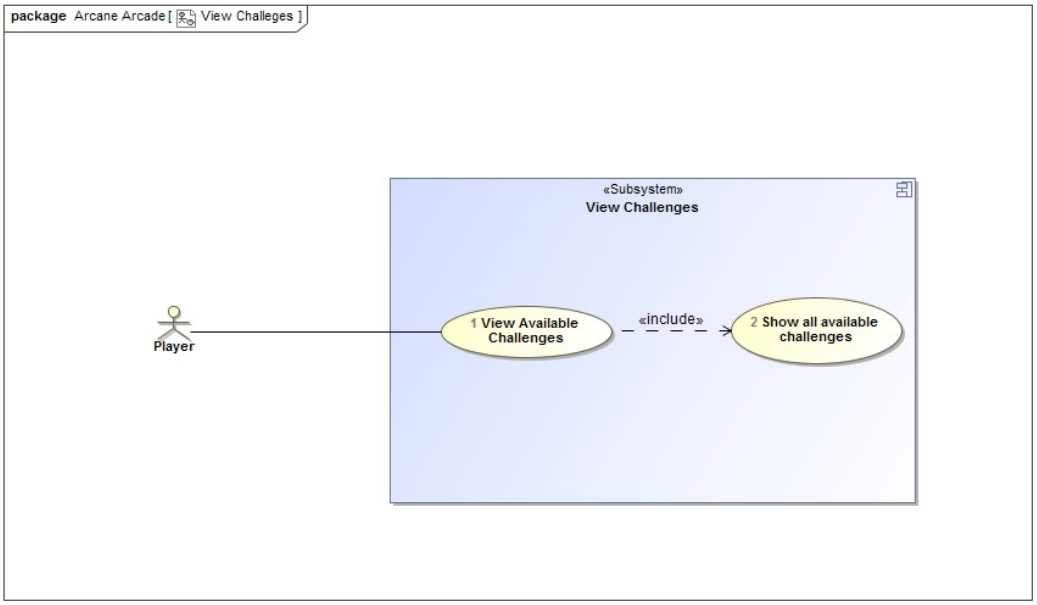
The change keywords usecases.



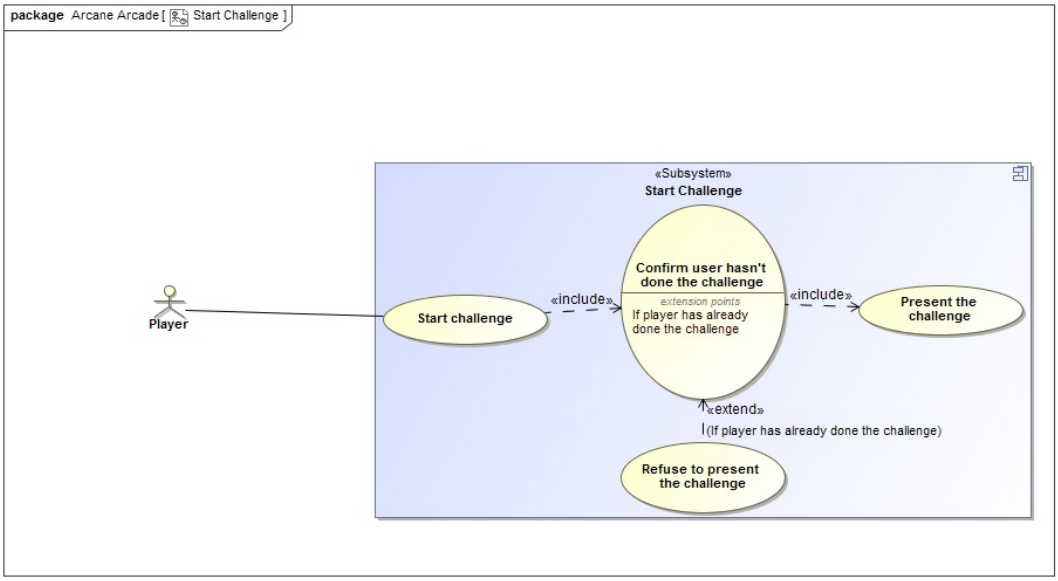
The login usecase.



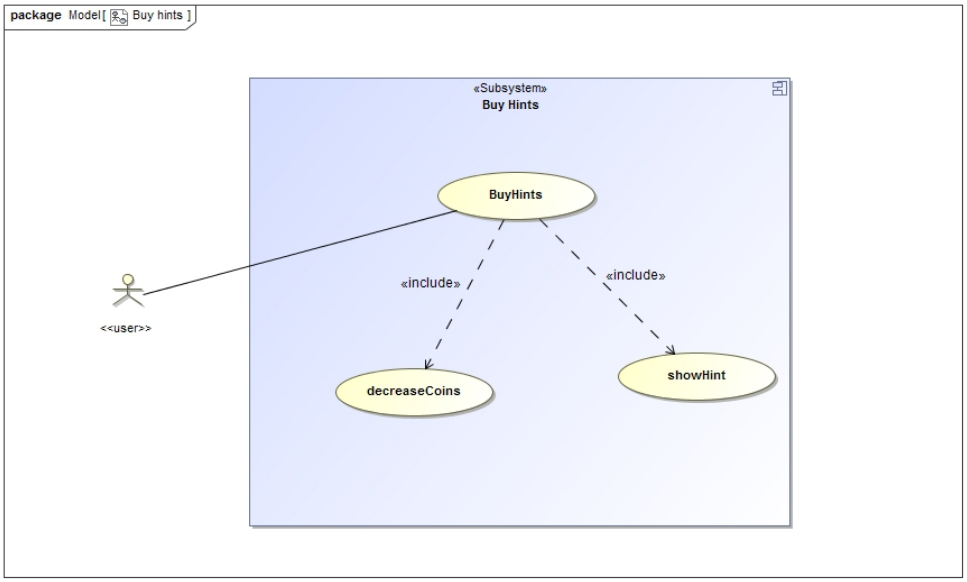
The view challenges usecase.



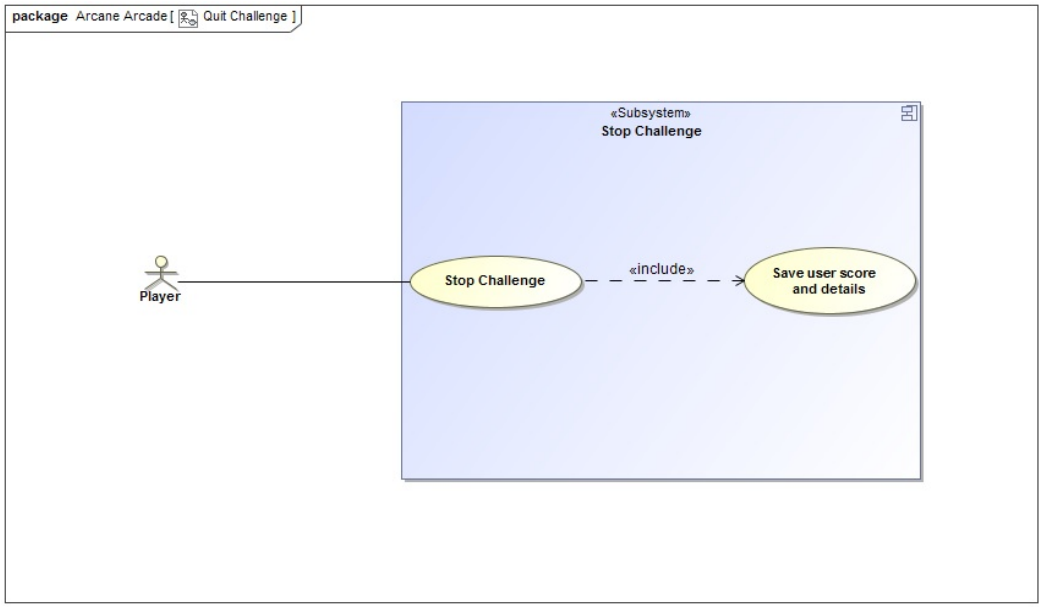
The start challenge usecase.



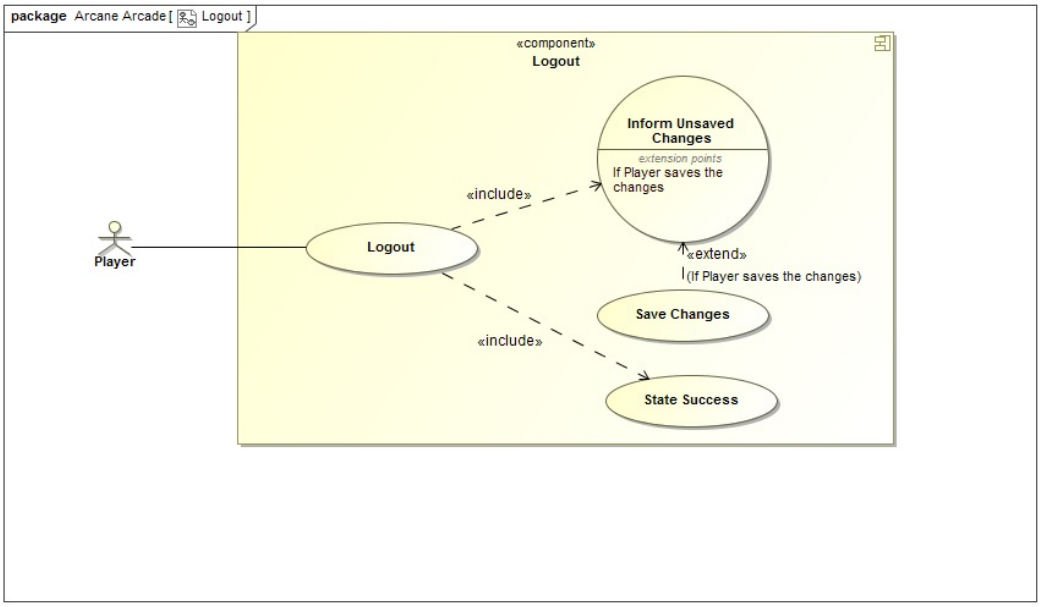
The buy hints usecase.



The quit challenge usecase.

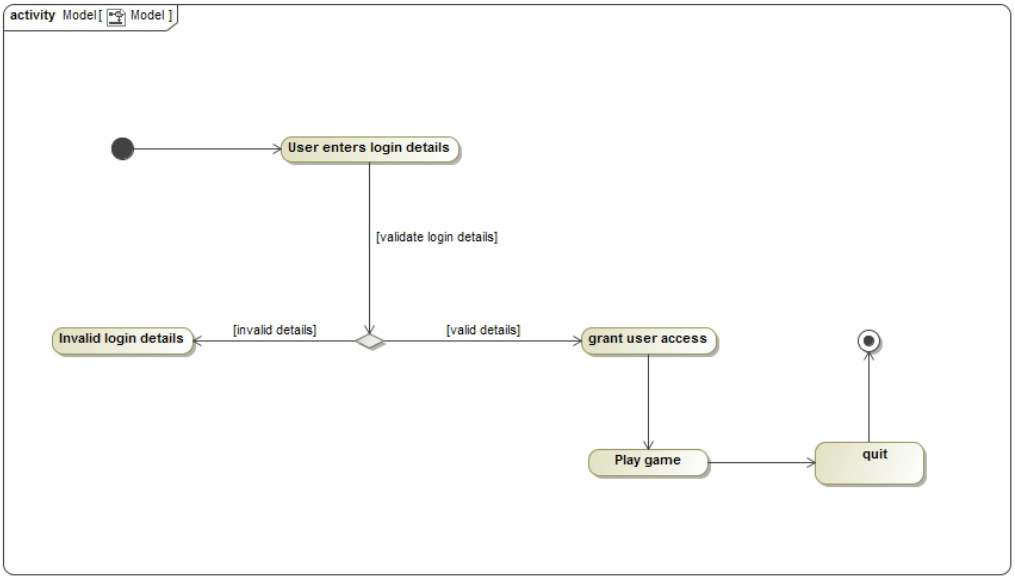


The logout usecase.

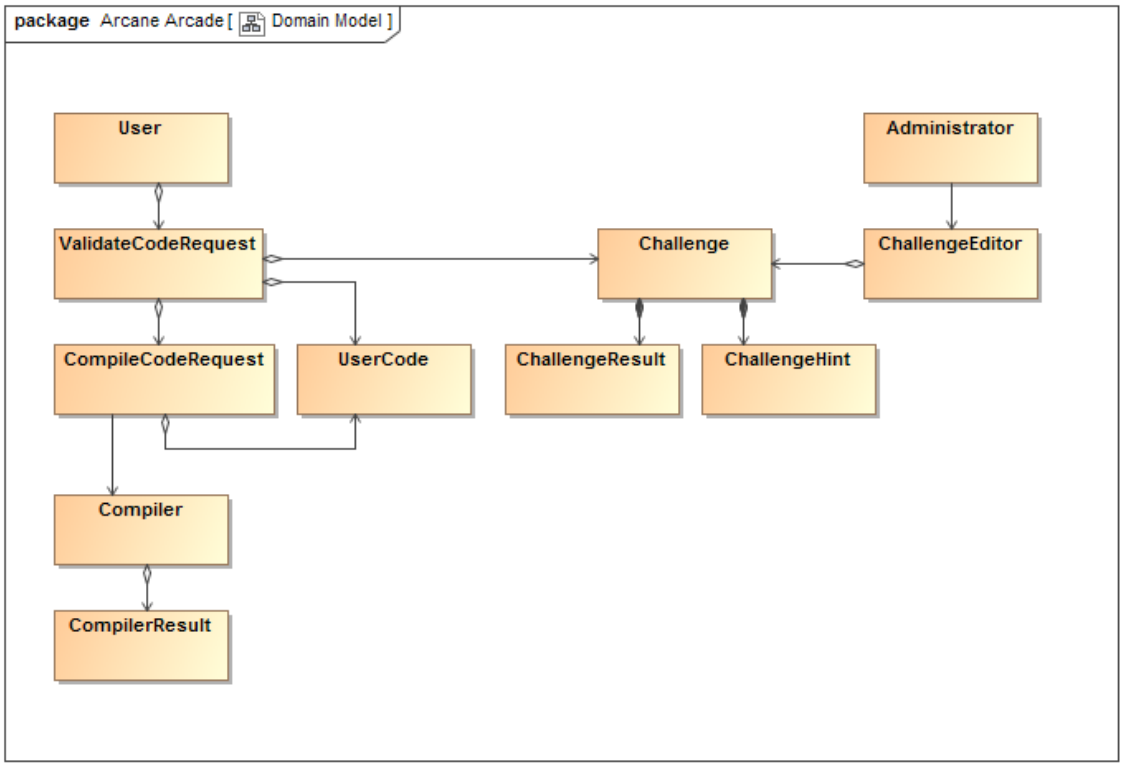


5.4.2 Activity Diagrams

The Login activity diagram.



5.5 Domain Model



6 Open Issues