PROJECT STORM
TEAM NAME

Renaldo van Dyk 12204359
Andreas du Preez 12207871
Sean Hill 12221458
Shaun Meintjes 133310896
Johann Dian Marx 12105202

GitHub link

Contents

1 Vision and Scope

1.1 History and Background

1.1.1 A Brief History of Project STORM

In 2013 lecturers of the department of Computer Science, which resides at the University of Pretoria, approached honors students of the module "Educational Software Development" to develop a team shuffling system. The system was going to be used by the lecturers of the module "Software Engineerin" to determine teams for the "Rocking the boat" exercise of the "Software Engineering" module using a set of lecturer defined criteria to select the teams with. An incomplete requirements specification document was designed and the project was halted.

1.1.2 Project Background

The lecturers of the "Software Engineering" module sought the need for such a shuffling tool, this time approaching students of the "Software Development" module. The requirements specification previously developed will be used as a starting point. This document will be stripped down to a "basic system" requirements specification, not completely discarding functionality specified in the previous documentation but adding relevant functionality as the development life-cycle persists.

1.2 Project Scope

The complete system should enable users to build teams, from a list of students, by selecting a set of criteria. This will aid the users in such a way that the users do not have to build the teams manually, which may require a lot of time. The users can spend their time rather on analyzing the results of each "Rocking the Boat" round to change the criteria for the next round more effectively.

2 Application requirements and design

2.1 Modular Design

The system is to be a modular system which allows for:

- Only a subset of modules to be deployed. Minimally the system will require the core modules to be deployed.
- Further modules to be added at a later stage.

To this end there should be:

- Minimal dependencies between modules, and
- No dependencies of core modules on any add-on modules.

Modular design allows that each module encapsulates information that is not available to the rest of the program. This information hiding reduces the cost of subsequent design changes when future functionality is added to STORM. For example, if at a later stage functionality is added to allow for personality tests to be completed within STORM and results are automatically pulled in, a new module can be added without affecting other modules.

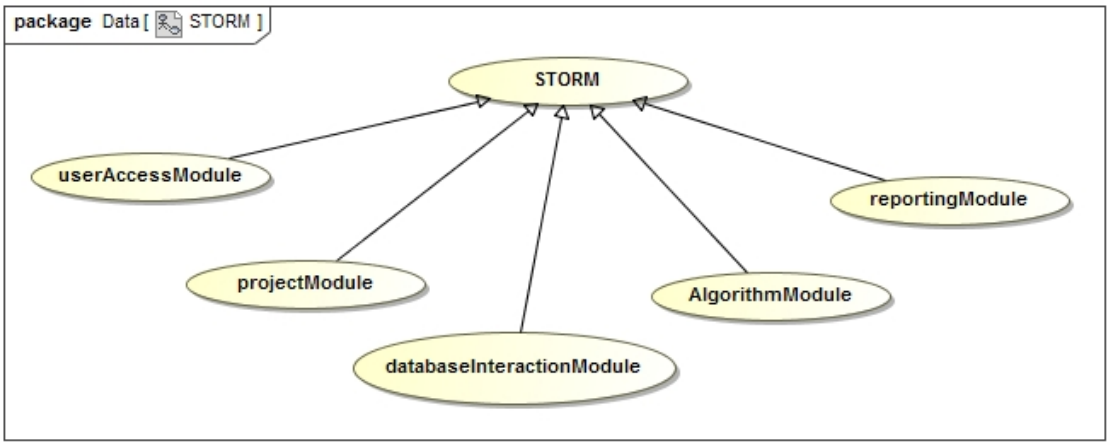

Figure 1: High level overview of STORM

2.2 User Access Module

This module deals with the STORM user access, specifically signing up, logging in, logging out and user authentication.

2.2.1 Use-cases

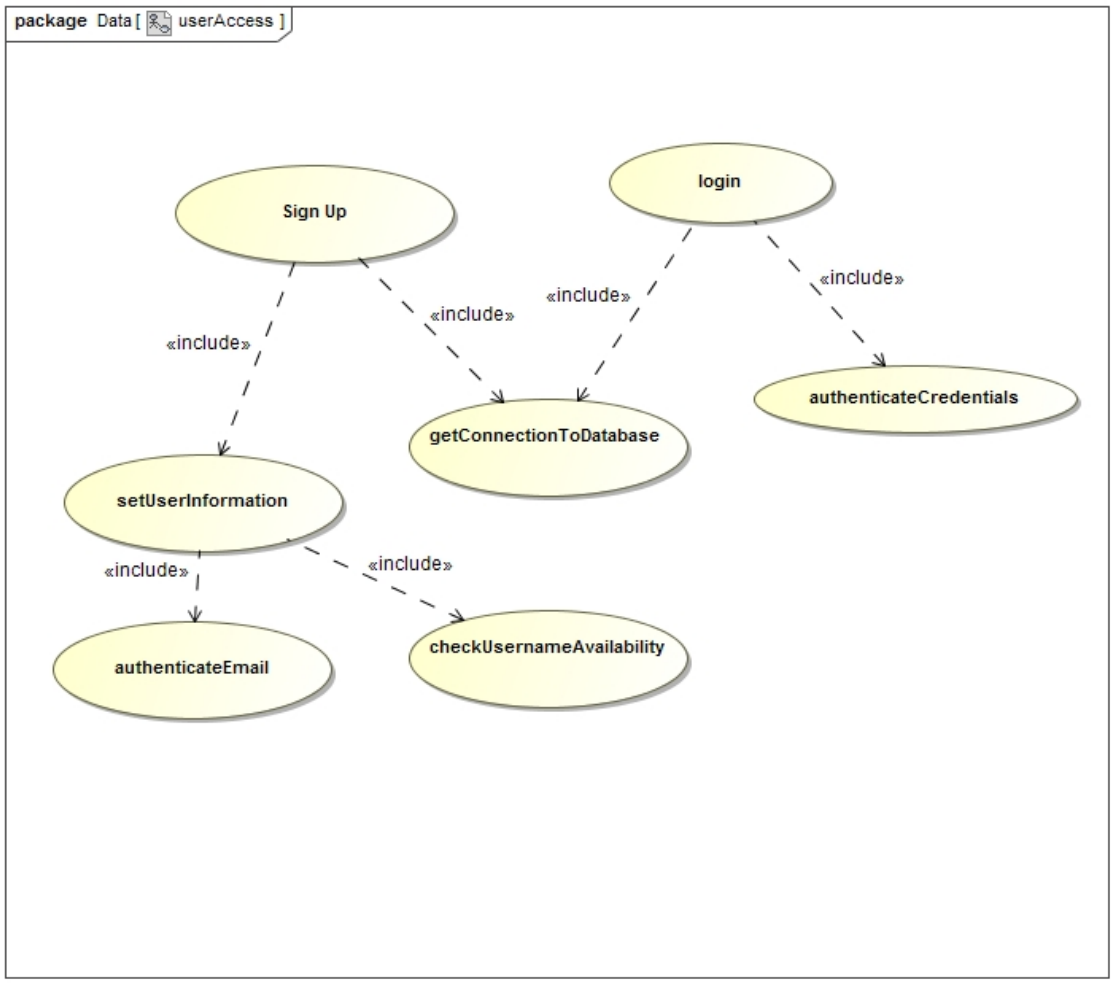

Figure 2: User access module use case

1. Sign Up
   Priority: Critical.

Pre-condition: Client must have a valid e-mail address.

Post-condition: Client has a STORM profile.

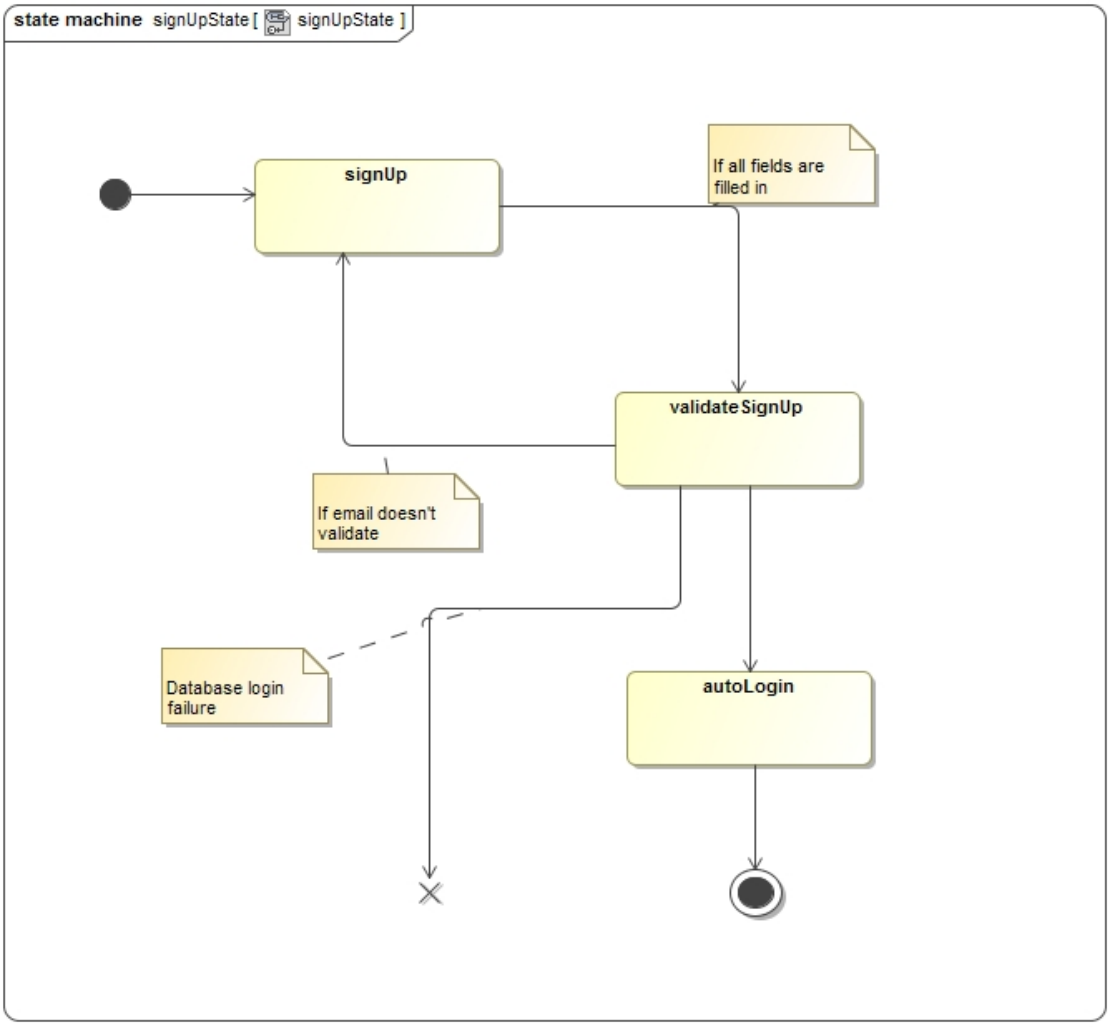

Figure 3: SignUp state diagram

2. Login

   Priority: Critical.

   Pre-condition: Client must have a STORM profile.

   Post-condition: Client can now use STORM functionality.

3. Log out

   Priority: Critical.

   Pre-condition: Client must be logged into STORM.

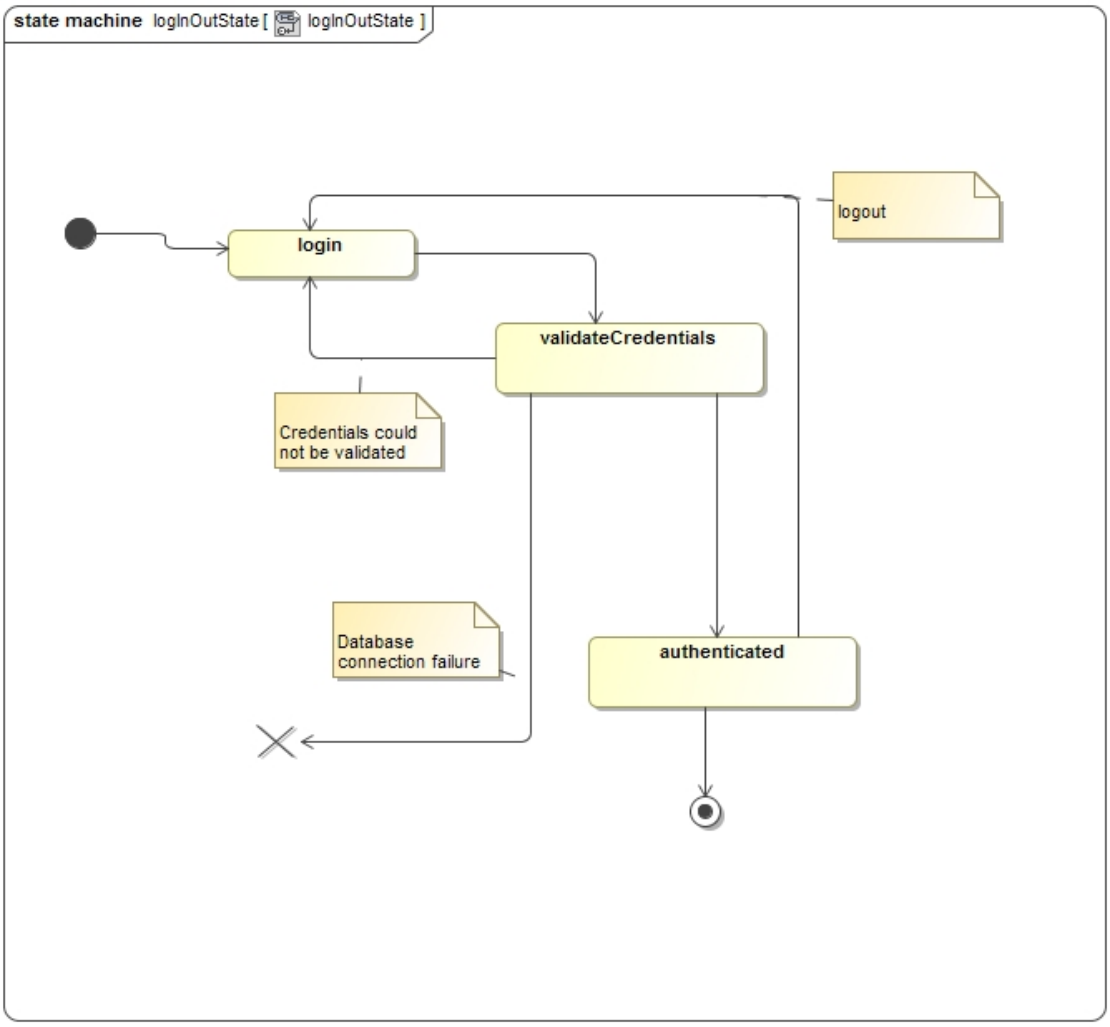

Figure 4: Login/logout state diagram

2.3 Project Module

This module deals with the configuration of projects. Initially a skeleton project will be set up with basic criteria and additional criteria can be added at a later stage as needed.

2.3.1 Use-cases

The project module provides services to create and manipulate projects.

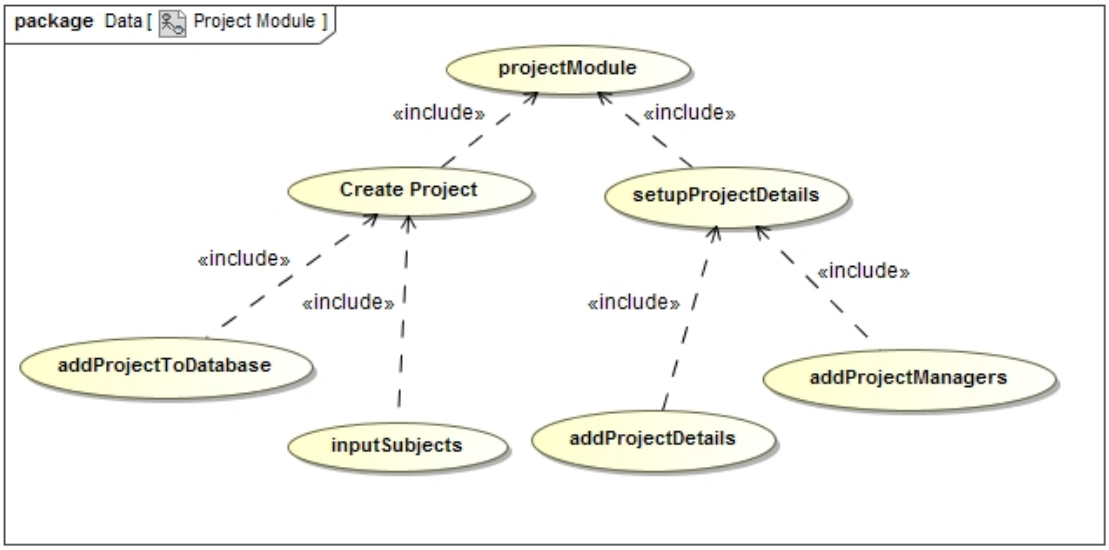

Figure 5: Project Module Use Case

1. addProjectDetails

   Priority: Critical.

   Pre-condition: Client must be logged in.

   Post-condition: Client must have a STORM profile.

   Post-condition: Project skeleton is created in database.

2. addProjectManagers

   Priority: Important.

   Pre-condition: Client must have a STORM profile.

Pre-condition: Manager to be added must have a STORM profile.

Pre-condition: Client must have created a STORM project.

Post-condition: Manager has permission to collaborate on the project.

3. inputSubjects

   Priority: Critical.

   Pre-condition: Client must have a STORM profile.

   Pre-condition: Client must have created a STORM project skeleton.

   Pre-condition: A .csv file should exist and should be selected.

   Pre-condition: The .csv should have subjects in it.

   Post-condition: Project database is updated with a list of subjects.

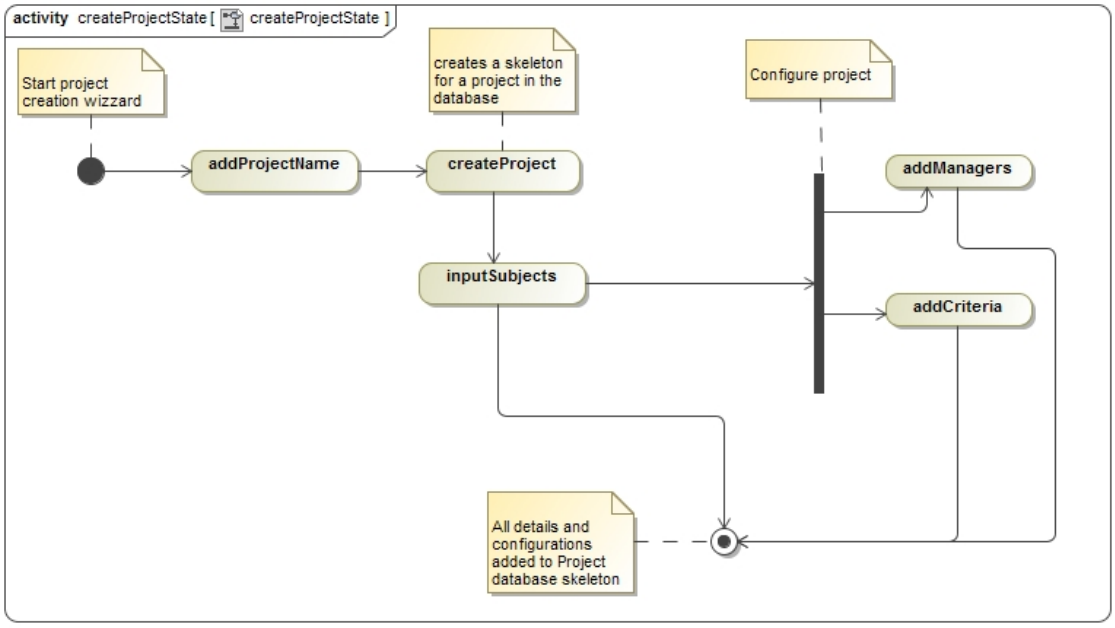

Figure 6: Create project sate diagram

2.4 Database Interaction Module

This module deals with the creation and interaction with the database.

2.4.1 Use-cases

After a project has been created, a database collection is created and associated with it. The project can then request and update the database and persist as the shuffling criteria grows.

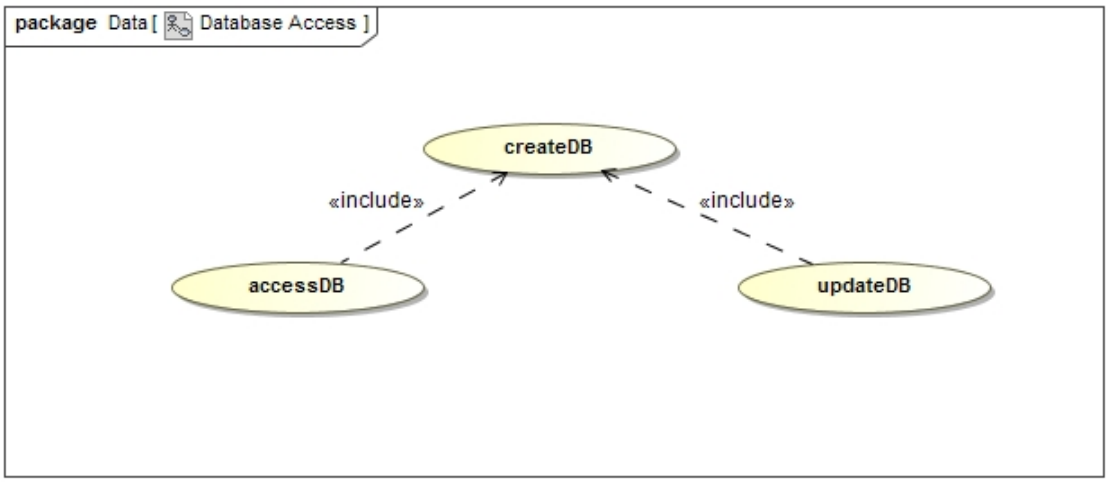

Figure 7: Database access use case

1. createDB

   Priority: Critical.

   Pre-condition: The databse should not exist yet.

   Pre-condition: Client must have a STORM profile.

   Post-condition: The database is created.

2. accessDB

   Priority: Critical.

   Pre-condition: The database should exist.

   Pre-condition: The user should have a STORM profile.

   Pre-condition: The user should be authorized to access the database.

   Post-condition: The database is accessed and queried.

3. updateDB

   Priority: Critical.

   Pre-condition: The database should exist.

   Pre-condition: The user should have a STORM profile.

   Pre-condition: The user should be authorized to update the database.

   Post-condition: The database is updated.

2.5 Algorithm Module

This module deals with the main process behind STORM, it is the shuffling algorithm.

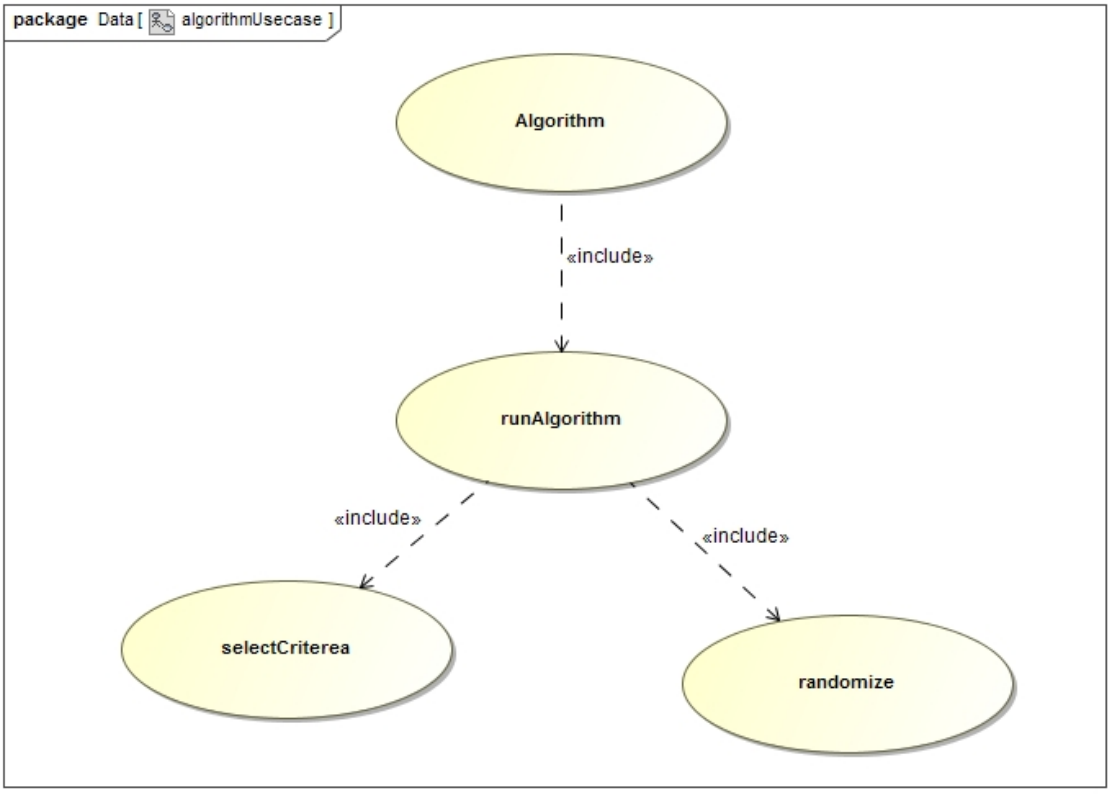

Figure 8: Algorithm module use case

2.5.1 Use-cases

The algorithm module adds the functionality required to shuffle subjects into teams.

1. Randomize

   Priority: Critical.

   Pre-condition: Project must have users.

   Pre-condition: Team size or number of teams must be specified.

   Post-condition: Random teams are built.

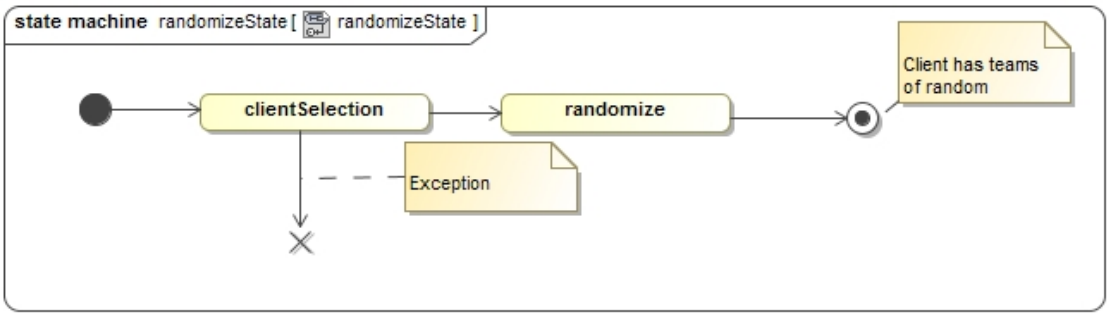

Figure 9: Randomize state diagram

To be completed in future iterations

2.6 Reporting Module

To be completed in future iterations

3 Architectural Requirements

3.1 Access and Integration Requirements

3.1.1 Human Access Channels

The system will be accessible by human users through the following channels:

- From a web browser through a rich web interface. The system must be accessible from any of the standards-compliant web browsers including all recent versions of Mozilla Firefox, Google Chrome, Apple Safari and Microsoft Internet Explorer.
- From mobile devices if time allows it.

3.1.2 System Access Channels

- Other systems should be able to access services offered by STORM. Perhaps direct feed of teams into other systems.
- Various technologies will be used such as:
  - Node.js
  - Express.js
  - EJS
  - Mongoose
  - SMTP

3.2 Quality Requirements

The quality requirements are the requirements around the quality attributes of the system and the services it provides. Quality requirements relevant to project STORM are listed below in order of priority.

3.2.1 Usability

Usability is one of the most important quality attributes. Usability tests should be performed to check whether:

- users find any aspects of the system cumbersome, non-intuitive or frustrating.
- the user has a positive experience and finds the functionality easy to use and learn.

3.2.2 Scalability

The system must implement a very generic optimization algorithm in order to be used by different parties in different contexts. In addition to different environments the system should be able to optimize groups for an extremely large number of subjects.

3.2.3 Performance

- The optimization algorithm should be able to sort a maximum of 10 000 subjects into groups and respond within 10 seconds.
- Reporting queries should respond within 15 seconds.

  **The above figures does not include the network round-trip which is outside the control of the system.*

3.2.4 Maintainability

Amongst the most important quality requirements for the system is maintainability which includes flexibility and extensibility. It should be easy to maintain the system in the future. To this end

- future developers should be able to easily understand the system,
- the technologies chosen for the system and be reasonably expected to be available for a long time,
- and developers should be able to easily and relatively quickly
  - change aspects of the functionality the system provides, and
  - add new functionality to the system.

3.2.5 Reliability

The system should provide by default a reasonable level of reliability and should be deployable within configurations which provide a high level of availability, supporting

- fail-over safety of all components and
- a deployment without single points of failure.

Hot deployment of new or changing functionality is not required for this system.

3.2.6 Deployability

The system must be deployable

- on Linux servers,
- and in environments using different databases for persistence of the STORM data.

3.2.7 Security

Security is a fairly important aspect of the system in order to protect sensitive project and subject data. The system should also allow users that are logged on to the system, to assign collaborators to a project.

3.2.8 Testability

All services offered by the system must be testable through

1. unit tests, testing components in isolation using mock objects, and
2. integration tests where components are integrated within the actual environment.

In either case, these functional tests should verify that

- the service is provided if all pre-conditions are met (i.e. that no exception is raised except if one of the pre-conditions for the service is not met), and

- that all post-conditions hold true once the service has been provided.

In addition to functional testing, the quality requirements like scalability, usability, auditability, performance and so on should also be tested.

3.3 Architectural Responsibilities

The architectural responsibilities for STORM includes the responsibilities of providing an infrastructure for

- a web access channel,
- hosting and providing the execution environment for the services/business logic of the system,
- persisting and providing access to domain objects,
- logging,
- and delivering reports readable by 3rd party applications.

3.4 Architecture Constraints

The choice of architecture is largely unconstrained and the development team has the freedom to choose the architecture and technologies best suited to fulfill the non-functional requirements for the system subject to:

1. The architecture being deployable on a server, such as the Linux server of the University of Pretoria.
2. The architecture is constrained to using web technology.