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 honours 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 Engineering 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 brought to an end.

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 sytem" requirements specification, not completely discarding functionality specified in the previous documentation but adding relevant functionality as the development lifecycle 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 analysing 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.

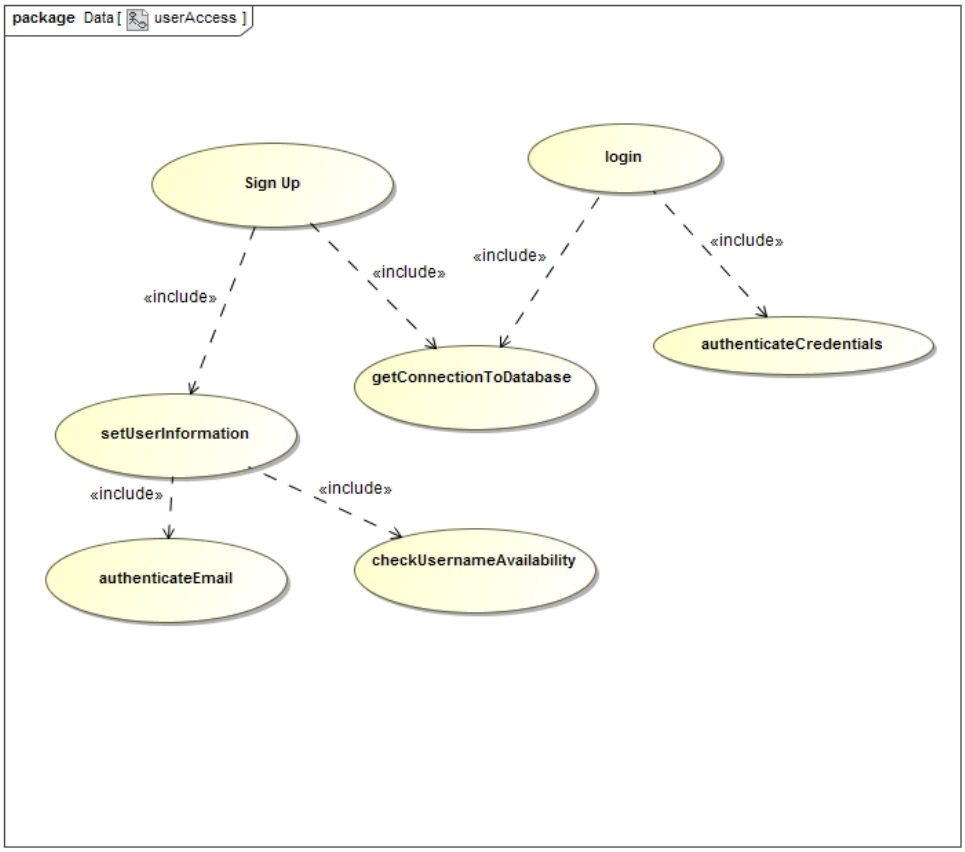
Modular design allows that each module encapsulates information that is not available to the rest of a program. This information hiding reduces the cost of subsequent design changes when future functionality is added to STORM. For example, if at a later stage functionality is added to allow for personality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

2.2 User Access Module

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

2.2.1 Use-cases

The user access module provides services to sign up, login and log out.



1. Sign Up

   Priority: Critical.

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

   Post-condition: Client has a STORM profile.

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.

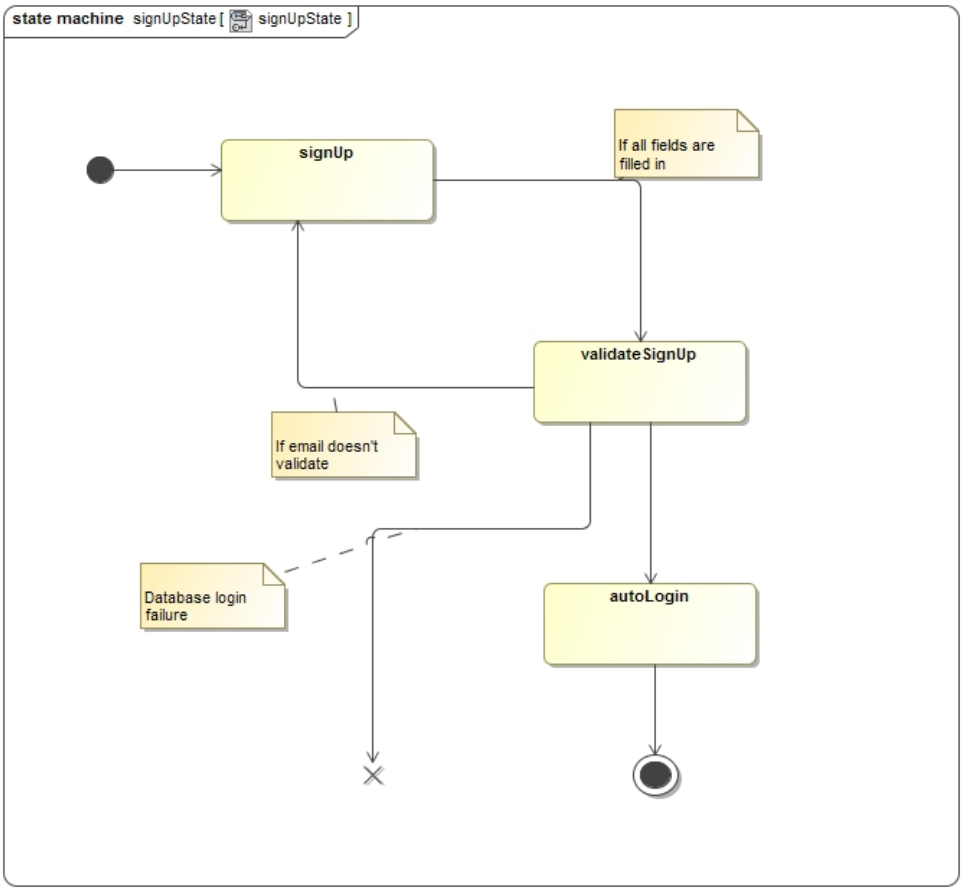


Figure 1: SignUp state diagram

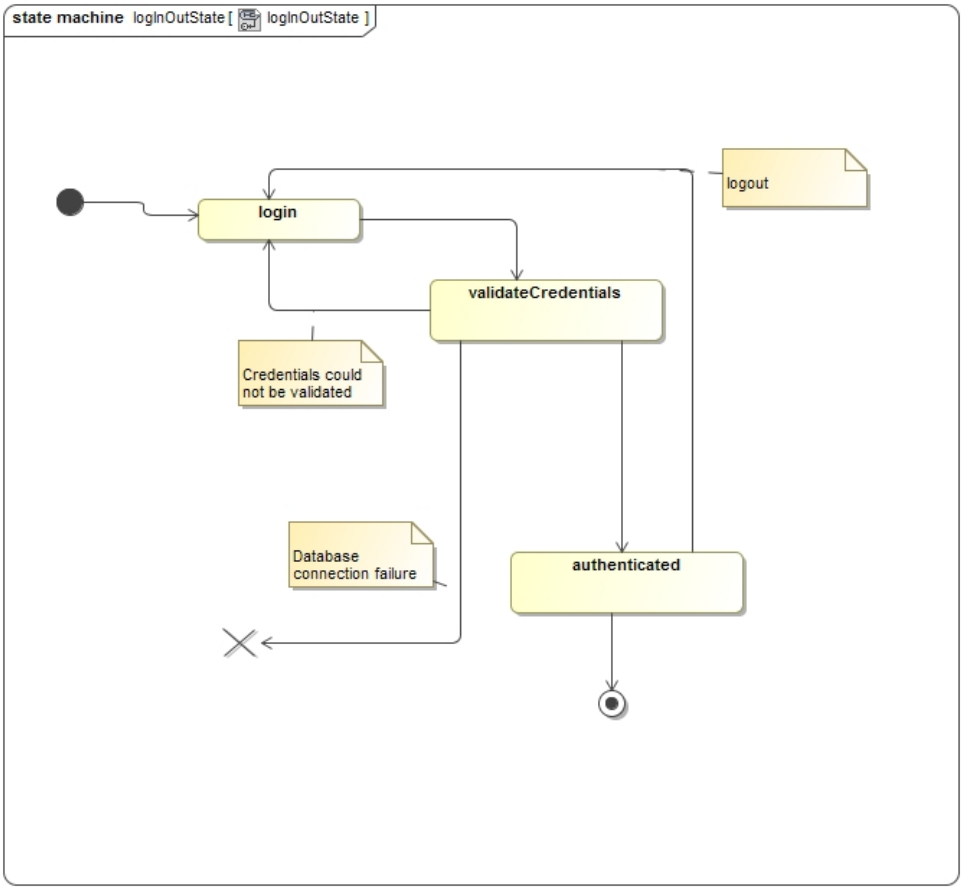


Figure 2: Login/logout state diagram

2.3 Project Module

2.4 Database Access Module

To be completed in future iterations

2.5 Algorithm Module

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:

1. 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.

2. 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.

3.1.3 System Access Channels

STORM will have access to a NoSQL database to save and retrieve all information needed to shuffle teams. It will also allow for importing to the database from at least CSV files, and possibly other types.

3.2 Quality Requirements

3.3 Architectural Responsibilities

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

1. a web access channel,
2. hosting and providing the execution environment for the services/business logic of the system,
3. persisting and providing access to domain objects,
4. logging,
5. 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.