



UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS 301

DEPARTMENT OF COMPUTER SCIENCE

Architectural Requirements and Initial Architecture Design Functional Requirements

Group Members:

Student numbers:

Diana Obo

u13134885

Priscilla Madigoe

u13049128

Kudzai Muranga

u13278012

Sandile Khumalo

u12031748

May 24, 2016

IMPAKD LINK

For further references see [gitHub](#). May 24, 2016

Contents

1	Vision	3
2	Background	4
3	Software Architecture	5
3.1	Architecture requirements	5
3.1.1	Architectural scope	5
3.1.2	Quality requirements	5
3.1.3	Integration and access channel requirements	5
3.1.4	Architectural constraints	5
3.2	Architectural patterns or styles	5
3.3	Architectural tactics or strategies	5
3.4	Use of reference architectures and frameworks	5
3.5	Access and integration channels	5
3.6	Technologies	5
4	Functional requirements and application design	6
4.1	Use case prioritization	6
4.2	Use case/Services contracts	6
4.3	Required functionality	6
4.4	Process specifications	6
4.5	Domain Model	6
5	Open Issues	7

1 Vision

2 Background

3 Software Architecture

3.1 Architecture requirements

3.1.1 Architectural scope

3.1.2 Quality requirements

3.1.3 Integration and access channel requirements

- Integration

- Logging into the system is done over a HTTPS POST method.
- The user's login details are kept in an HTTP session so the user does not need to log in everytime he/she makes a request to the server.
- The HTTPS sessions are invalidated when the user terminates his/her session by logging out.
- Communication between the server (back-end) and the webpage (front-end) will be facilitated by the REST method which uses JSON objects and HTTPS methods to send requests and get responses.
- The "Create, Read, Update and Delete" or CRUD actions that will make changes to the database will be logged automatically. This will ensure auditability of the system.

- Human Access Channel

- End-users interact with the Web client to display the required information and do desired actions.

- System Access Channel

- The Web-based component of the system will be implemented in "Ember.js" which utilises JavaScript, HTML and "Handlebar.js".

3.1.4 Architectural constraints

3.2 Architectural patterns or styles

MVC (*Model View Controller*)

Allows the system's states to change and it encapsulates the interactions from the user and transforms these interactions into business logic.

REASON:

- Reduce presentation layers complexity and improves flexibility
 - Separates responsibilities
 - * Provide view onto information - *View*

- * React to user events - ***Controller***
 - * Provide business services and data - ***Model***
- Allows each component to change independently
- Full decoupling
 - Model from both *view* and *controller*
- Simplification
 - Through separation of concerns
- Reuse
 - *Model* components and *View* components
- Maintainability
 - Different components can be used, developed and maintained by different members of a team
 - * *Model* - backened developers
 - * *View* - UI designers
 - * *Controller* - Front-end developers
- Improved Testability
 - Model/business services tested independently of UI
 - UI tested with mock model

3.3 Architectural tactics or strategies

3.4 Use of reference architectures and frameworks

3.5 Access and integration channels

3.6 Technologies

4 Functional requirements and application design

4.1 Use case prioritization

Critical:

- calculateROI
- getDefaultValues
- setDefaultValues

important:

- Register
- Login
- logout
- addProperty
- updateProperty
- deleteProperty
- displayGraphs
- displayStatistics

Nice-to-have:

- updateProfile
- generateReport

4.2 Use case/Services contracts

4.3 Required functionality

4.4 Process specifications

4.5 Domain Model

5 Open Issues