



UNIVERSITEIT VAN PRETORIA  
UNIVERSITY OF PRETORIA  
YUNIBESITHI YA PRETORIA

---

COS 301

DEPARTMENT OF COMPUTER SCIENCE

---

# Main Project 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

<b>1</b>	<b>Vision</b>	<b>3</b>
<b>2</b>	<b>Background</b>	<b>4</b>
<b>3</b>	<b>Software Architecture</b>	<b>5</b>
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
<b>4</b>	<b>Functional requirements and application design</b>	<b>6</b>
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
<b>5</b>	<b>Open Issues</b>	<b>7</b>

# 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

#### 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 section to change independently
- Full decoupling
  - Model from both view and controller
- Simplification
- Reuse
- Maintainability
- Improved Testability

### 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