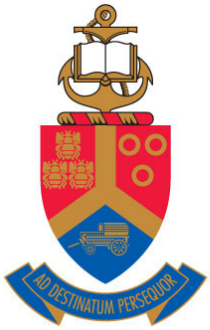

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

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 25, 2016

IMPAKD link

For further references see gitHub. May 25, 2016

Contents

1 Vision

The Property Investor Optimiser project is objective is to evaluate whether a certain rental propery is worth buying. It does this by calculating the Return of Investment (ROI) of a property, which can be compared with another property's ROI, to assist a user to optimise their investment strategy according to their portfolio.

The project will assist the user by helping to answer the following questions:

- Given a certain bond (interest rate, deposit as a percentage of property value), rental (occupancy rate,agent commission, rental amount) and environmental conditions (Interest rate, inflation) what is the ROI?
- When is it better to pay a higher or lower deposit for a bond?
- Between two rental scenarios which provides the greater ROI?
- Is it better to try and pay off the bond as fast as possible by paying in extra capital?
- How does purchasing another property influence a users ROI and at which point would this be a good idea?
- At which point does it make sense to buy another property?
- How much tax will the user have to pay?

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 databse 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 implememted in "Ember.js" which utilises JavaScript, HTML and "Handlebar.js".

3.1.4 Architectural constraints

- User
  - Has to be registered and his/her details in the system inorder to login and be able to use the system
- Time
  - If a user is logged in and remains inactive for more than 30mins the user will have to login again before they can use the system again

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 intercations 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 memebers 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

- Django
  - Our System is going to be web based, so Django is used for web development back end of the system. The structural pattern we are going to use is MVC and Django implements MVC

- JavaEE
  - JavaEE contains most of the frameworks and technologies we need to develope and deploy our system.

Technologies

- *HTML5 :* It will be used to create the front end of the system
- *Javascript :* front end to verify the log in details for each user it will keep track of the user logged in.
- *JPA:* It will be used to manage the relational data in JavaEE
- *EJBs:* It will be used to manage concurrency control in the system
- *Ember.js:* To implement the MVC patten
- *Django:* To implement the MVC Patten
- *CSS:* It will be used for the styling of the web page
- *Bootsrap:* It will be used for the styling of the web page
- *Web browsers:* Any web browser that supports HTML 5

3.5 Access and integration channels

- This is a stand-alone application and therefore will not use other applications for all the required functionality.
- Plug-ins and APIs will be included, and will therefore be integrated with the main application to add specialised functionality.

3.6 Technologies

4 Functional requirements and application design

4.1 Use case prioritization

Critical:

- calculateROI
- getDefaultValues
- setDeafultValues

Important:

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

Nice-to-have:

- updateProfile
- generateReport

4.2 Use case/Services contracts

login

- *Pre-Conditions:*
- *Post-Conditions:*
- *Request and Results Data Structures:*

register

- *Pre-Conditions:*

- *Post-Conditions:*
- *Request and Results Data Structures:*

logout

- *Pre-Conditions:*
- *Post-Conditions:*
- *Request and Results Data Structures:*

viewProfile

- *Pre-Conditions:*
- *Post-Conditions:*
- *Request and Results Data Structures:*

updateProperty

- *Pre-Conditions:*
  - user must be logged in
- *Post-Conditions:*
  - the property page and associated fields must be updated
  - database should be updated
- *Request and Results Data Structures:*

deleteProperty

- *Pre-Conditions:*
  - user must be logged in
  - property must exist
- *Post-Conditions:*
  - property is deleted
  - user must not see deleted property anymore
- *Request and Results Data Structures:*

addProperty

- *Pre-Conditions:*

  - user must be logged in
  - user must be navigated to the addProperty page
  - no duplicate properties
- *Post-Conditions:*
  - property is added
  - user should be able to view addProperty
- *Request and Results Data Structures:*

updateProfile

- *Pre-Conditions:*
- *Post-Conditions:*
- *Request and Results Data Structures:*

compareTwoProperties

- *Pre-Conditions:*
- *Post-Conditions:*
- *Request and Results Data Structures:*

generateReport

- *Pre-Conditions:*
- *Post-Conditions:*
- *Request and Results Data Structures:*

getDefaultValues

- *Pre-Conditions:*
- *Post-Conditions:*
- *Request and Results Data Structures:*

displayGraphs

- *Pre-Conditions:*
- *Post-Conditions:*
- *Request and Results Data Structures:*

displayStatistics

- *Pre-Conditions:*
- *Post-Conditions:*
- *Request and Results Data Structures:*

calculateStatistics

- *Pre-Conditions:*
- *Post-Conditions:*
- *Request and Results Data Structures:*

calucalateROI

- *Pre-Conditions:*
- *Post-Conditions:*
- *Request and Results Data Structures:*

4.3 Required functionality

4.4 Process specifications

4.5 Domain Model

5 Open Issues