UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

ARCHITECTURAL REQUIREMENTS
iCrawler
Emilio Mumba

The 5 Concurrent Nodes

Khathutshelo Shaun Matidza
Sylvester Sandile Mpangane
Thabang Michael Letageng
Aluwani Augustine Simetsi
Matthew Nel

Department of Computer Science, University of Pretoria

July 2015

Contents

1 Introduction

This document contains the software (application) architectural requirements specification which is the infrastructure upon which the application will be developed.

The following requirements will be addressed:

- Quality requirements
- Architectural Patterns or Styles
- Access and Integration Channels
- Technologies

2 Android Application Requirements

2.1 Architectural Requirements

2.1.1 Critical Quality Requirements

Auditability

Description

Auditability refers to the ability to account for a system's usage by the user and be able to monitor events and create logs on what are the user's actions within the system.

Justification

iCrawler's main functionality is to monitor the user's device activities and report all logs of that particular device.

Mechanism

1. Strategy:

   Auditability can be achieved by:

   - Resource Monitoring: A tactic that registers all the events and resources that the application uses; a log of these resource usage is then created.

2. Architectural Pattern(s):

   - To monitor the application events and resources a design pattern such as the *Observer* pattern can be implemented by the application that can register all events and resource usage.

Security

Description

Security is the degree of resistance to, or protection from, harm. It applies to any vulnerable and valuable asset, such as a person, dwelling, community, nation, or organization.

Justification

Security is a critical aspect of any system that deals with critical and confidential data, the strategies used for security provides mechanisms to protect data from unauthorized access and modification. For our mobile monitoring application this means the following:

- No one (person or program) should be able to delete the data collected on device.
- When the data is transferred over the network, data should be protected from being intercepted or hacked.

Mechanism

1. Strategy:

   Security can be achieved by:

   - Authentication: This strategy is used to identify and confirm a user's identity.
   - Encryption: Data is converted to a secure format that cannot be easily read by unauthorized individuals.

2. Architectural Pattern(s):

   - Layering: This pattern decouples the system by dividing it into components (layers) that communicate with each other through message requests and responses, this prevents direct requests to critical data from the application to the database.

2.1.2 Important Quality Requirements

Maintainability

Description

This is the ease with which a product can be maintained in order to isolate defects or their cause, maximize a product's useful life, meet new requirements, maximize efficiency, reliability, and safety.

Justification

A modular design decouples a system into components that are easy to maintain and makes the system more adaptable. Decoupling the system will also ensure that it is easy to do unit testing and integration testing.

Mechanism

1. Strategy:

   Maintainability can be achieved by:

   - Decoupling: This strategy breaks the system into manageable components to achieve a proper structure and maintainable system.

2. Architectural Patterns(s):

   - Microkernel: The *Microkernel* pattern improves maintainability because it separates high level services from low level services that is, it divides the systems into components that are maintainable and also allow for components to be easily removed or added to the system.

Performance

Description

Performance is a measure of a system's responsiveness when executing some action.

Justification

iCrawler needs to use the devices resources efficiently to increase the performance of the application which is an important quality requirement that will also make the system more reliable and increase throughput of the application.

Mechanism

1. Strategy:

   Performance can be achieved by:

   - Dynamic code optimization: This strategy focuses on the code design, quality and efficiency to improve performance on the system.

2. Architectural Patterns(s):

   - The best way to achieve performance is to manage the system's resources efficiently and use design patterns to optimize code and improve efficiency.

2.1.3 Nice-To-Have Quality Requirements

Testability

Description

Testability is a measure of how well a system allows one to test if a certain criteria is met by the system. This makes fault detection in the system easy and also the faults can be isolated in a timely manner.

Justification

It is vital that every component that is deployed on the system can be tested using unit testing and also integration testing so that faults can be detected as soon as possible and be fixed.

Mechanism

1. Strategy:

    - White-box: This tactic is mainly used for unit testing and requires the knowledge of the internal structure to create test cases for the application components.
    - Black-box: This tactic is mainly used for integration testing, it examines the functionality of the application against the specification and simplifies system components testing when plugged in a modular system.

2. Architectural Patterns(s):

    - Model View Controller: This pattern promotes separation of concern in a system by decoupling the system into components that can be tested independently.

2.1.4 Architectural Patterns or Styles

Model View Controller (MVC)

Description

Separates the applications concerns by separating the following responsibilities:

- Model: Provides business services and data.
- View: Provides a view for information.
- Controller: Reacts to user events.

Justification

This pattern separate the system's concerns into components that can evolve independently which reduces the application's complexity. The iCrawler mobile monitoring application is intended to have low complexity, to be a modular system and it also needs to be maintainable which can all be achieved best by using the MVC pattern.

Benefits

- Simplification
- Improve maintainability
- Improve reuse
- Improve testability

2.1.5 Layered Architecture

Description

Partitions the application's concerns into a stacked group of layers.

Justification

It will provide a high level of abstraction which will allow the application components to vary; this decouples the system, reduce complexity and improve the systems performance which is the objective for the mobile monitoring application.

Benefits

- Improve cohesion
- Reduce complexity
- Improve maintainability
- Loose coupling
- Improve testability
- Improve reuse

2.2 Access and Integration Channels

2.2.1 Access Channels

Human Access Channel

Human access channels addresses all the different ways in which a human can interact with the Mobile Monitoring Application.

- Mobile device: The application only runs on android mobile devices, only minimal user-interaction is presented to the user as the application runs on the background.

2.2.2 Integration Channels

Channels

The mobile monitoring application will need to access a MySQL database to store user information and data logs collected from the device.

Protocols

The protocols the application will use are the following:

- HTTPS: This protocol will be used for security to ensure that a secure connection is maintained between the device and server and also that transported data cannot be easily intercepted.

- SMTP: Notifications for a user who forgets his/her password will use this protocol to allow that user to recover their credentials.

2.3 Technologies

2.3.1 Platform and IDE

- Android Device(s)
- Android Studio IDE

2.3.2 Programming Languages

- Java

2.3.3 Frameworks

- JUnit

2.3.4 Databases

- MySQL Relational Database

2.3.5 Web services

- REST

2.3.6 Others

- AJAX
- JSON

3 Web Application (Dashboard) Requirements

3.1 Architectural Requirements

3.1.1 Critical Quality Requirements

Scalability

Description

Scalability refers to the ability of a system to easily accommodate and handle a large amount of work at a single instance

Justification

The iCrawler mobile monitoring application dashboard needs to be able to handle as many concurrent users as possible without breaking.

Mechanism

1. Strategy:

   Scalability can be achieved by:

   - Clustering: Ensures that resources are not strained by running or maintaining many instances of the application over a cluster of servers.
   - Caching: Will reduce database workload by maintaining database query results on the application within a user session to avoid querying the database every time.

Security

Description

Security is a critical aspect of any system that deals with critical and confidential data, the strategies used for security provides mechanisms to protect data from unauthorized access and modification; this means the following for our application dashboard:

- Only authorized individuals may have access to the relevant data.
- Users should strictly be restricted by access levels

Justification

Security is a high priority feature for any system that deals with critical and confidential data, this is the case with the iCrawler application. User collected data needs to be protected from being tampered with and accessed by unauthorized individuals to maintain its integrity and confidentiality.

Mechanism

1. Strategy:

   Security can be achieved by:

   - Authentication: The strategy is used to identify and confirm a user's identity.
   - Encryption: Data is converted to a secure format that cannot be easily read by unauthorized individuals.

2. Architectural Pattern(s):

   - Layering: This pattern decouples the system by dividing it into components (layers) that communicate with each other through message requests and responses, this control the access of the user level layer from directly make request to lower layers that provides critical data.

3.1.2 Important Quality Requirements

Maintainability

Description

This is the ease with which a product can be maintained in order to isolate defects or their cause, maximize a product's useful life, meet new requirements, maximize efficiency, reliability, and safety.

Justification

The dashboard application must be decoupled into components that are easy to maintain and make the system more adaptable.

Mechanism

1. Strategy:

   Maintainability ca be achieved by:

   - Readability: The dashboard code should be easy to read for whoever is maintaining it. This means that it should be indented properly and make use of comments.

2. Architectural Patterns(s):

   - Microkernel: The *Microkernel* pattern improves maintainability because is separates high level services from low level services that is, it divides the systems into components that are maintainable and also allow for components to be easily removed or added to the system.

Performance

Description

Performance is a measure of a system responsiveness when executing some action.

Justification

The dashboard application needs to use resources efficiently to increase its performance and provide quality experience to the user.

Mechanism

1. Strategy:

   Performance can be achieved by:

   - Dynamic code optimization: This strategy focuses on the code design, quality and efficiency to improve performance on the dashboard system.

3.1.3 Nice-To-Have Quality Requirements

Testability

Description

Testability is a measure of how well a system allows one to test if a certain criteria is met by the system. This makes fault detection in the system easy and also the faults can be isolated in a timely manner.

Justification

It is vital that every component that is deployed on the dashboard system can be tested in some way so that faults can be detected as soon as possible and be fixed.

Mechanism

1. Strategy:

    - Userbility testing: This is a technique used in user-centered interaction design to evaluate a product by testing it on users. This can be seen as an irreplaceable usability practice, since it gives direct input on how real users use the system.

2. Architectural Patterns(s):

    - Model View Controller: This pattern promotes separation of concern in a system by decoupling the system into components that can be tested independently.

3.2 Architectural Patterns or Styles

3.2.1 Model View Controller (MVC)

Description

Separates the applications concerns by separating the following responsibilities:

- Model: Provide business services and data.
- View: Provide view for information.
- Controller: Reacts to user events.

Justification

This pattern separate the system's concerns into components that can evolve independently which reduce the dashboard systems complexity.

Benefits

- Simplification
- Improve maintainability
- Improve reuse
- Improve testability

3.2.2 Layered Architecture

Description

Partitions the dashboard system's concerns into a stacked group of layers.

Justification

Provides high level of abstraction which allows the dashboard components to vary, this decouples the system,reduces its complexity and improve the systems performance which is also the objective for the iCrawler mobile monitoring application dashboard

Benefits

- Improve cohesion
- Reduce complexity
- Improve maintainability
- Loose coupling
- Improve testability
- Improve reuse

3.3 Access and Integration Channels

3.3.1 Access Channels

Human Access Channel

Human access channels addresses all the different ways in which a human can interact with the iCrawler dashboard system.

The dashboard is accessible through a web browser but it is restricted to only the following:

- Desktop Computer
- Tablet
- Laptop

3.3.2 Integration Channels

Channels

The dashboard will need to access a MySQL database to read user information and data logs from the device.

Protocols

The dashboard system will use the following protocols:

- HTTPS: This protocol will be used for security to ensure that a secure connection is maintained between the application and server and also that transported data cannot be easily intercepted.
- SMTP: Notifications for a user who has forgotten his/her password will use this protocol to allow the user to recover their login information.

3.4 Technologies

3.4.1 Platform

- JavaEE

3.4.2 APIs

- JAX-RS 2.0
- JPA (Java Persistence API)
- JTA (Java Transition API)

3.4.3 Persistence Provider

- Hibernate

3.4.4 Application Server

- GlassFish

3.4.5 Programming Languages

- Java

3.4.6 Frameworks

- JUnit

3.4.7 Dependency Injector

- CDI (Context and Dependency Injector)

3.4.8 Dependency Management

- Apache Maven

3.4.9 Databases

- MySQL Relational Database

3.4.10 Web services

- REST

3.4.11 Others

- JSON
- Java Server Faces
- Servlets