

WIT MSc Communications Software  
Architecture Description (AD)

WIT Design Patterns Assignment AD: Pacemaker Android Application

*<< Note: This Architecture Description (AD) template from the Open Group TOGAF™ 9.1 architecture framework has been adapted to the purposes of the WIT MSc module on IT Architecture Patterns.>>.*

Table of Contents

[A)Problem Description 6](#__RefHeading___Toc2122_729351093)

[A.1PROBLEM SCOPE 6](#__RefHeading___Toc2124_729351093)

[A.1.1PROBLEM SUMMARY 6](#__RefHeading__163_1468066642)

[A.1.2DOMAIN GLOSSARY 9](#__RefHeading___Toc2126_729351093)

[A.2STAKEHOLDERS CONCERNS 9](#__RefHeading__10_1468066642)

[A.3REQUIRED QUALITY PROPERTIES 9](#__RefHeading__18_1468066642)

[B)Solution Description 11](#__RefHeading__42_1468066642)

[B.1ARCHITECTURE VIEWS 11](#__RefHeading___Toc2128_729351093)

[B.2CONTEXT VIEW 12](#__RefHeading__44_1468066642)

[B.2.1VIEW INTENT 12](#__RefHeading__193_1468066642)

[B.2.2VIEW MODELLING ARTIFACTS 13](#__RefHeading___Toc2134_729351093)

[B.2.2.1SYSTEM CONTEXT MODEL 13](#__RefHeading___Toc2136_729351093)

[B.2.2.2SYSTEM FEATURES MODEL 19](#__RefHeading___Toc1991_1386314816)

[B.3BEHAVIORAL VIEW 20](#__RefHeading___Toc1993_1386314816)

[B.3.1VIEW INTENT 20](#__RefHeading__193_14680666421)

[B.3.2VIEW MODELLING ARTIFACTS 20](#__RefHeading___Toc2134_7293510931)

[B.3.2.1DATA PREPARATION MODEL 20](#__RefHeading___Toc2136_7293510931)

[B.3.2.2DASHBOARD RENDERING MODEL 23](#__RefHeading___Toc1995_1386314816)

[B.4INFORMATION VIEW 30](#__RefHeading___Toc1997_1386314816)

[B.4.1VIEW INTENT 30](#__RefHeading__193_146806664211)

[B.4.2VIEW MODELLING ARTIFACTS 34](#__RefHeading___Toc2134_72935109311)

[B.4.2.1BATCH ANALYTICS DATA FLOW MODEL 34](#__RefHeading___Toc2136_72935109311)

[B.4.2.2DASHBOARD RENDERING DATA FLOW MODEL 36](#__RefHeading___Toc1999_1386314816)

[B.5FUNCTIONAL VIEW 38](#__RefHeading___Toc2001_1386314816)

[B.5.1VIEW INTENT 38](#__RefHeading__193_1468066642111)

[B.5.2VIEW MODELLING ARTIFACTS 38](#__RefHeading___Toc2134_729351093111)

[B.5.2.1INDIRECTION LAYERS MODEL 38](#__RefHeading___Toc2136_729351093111)

[B.5.2.2DATA PROVISIONING MODEL 40](#__RefHeading___Toc2003_1386314816)

[B.6ARCHITECTURE PERSPECTIVES 42](#__RefHeading___Toc2149_1386314816)

[B.6.1PERFORMANCE PERSPECTIVE 42](#__RefHeading___Toc2151_1386314816)

[B.6.1.1RECORDING of DESIGN DECISIONS / TRADE-OFFS 42](#__RefHeading___Toc2153_1386314816)

[B.6.2SCALABILITY PERSPECTIVE 42](#__RefHeading___Toc2155_1386314816)

[B.6.2.1RECORDING of DESIGN DECISIONS / TRADE-OFFS 42](#__RefHeading___Toc2157_1386314816)

[B.6.3TESTABILITY PERSPECTIVE 43](#__RefHeading___Toc2159_1386314816)

[B.6.3.1RECORDING of DESIGN DECISIONS / TRADE-OFFS 43](#__RefHeading___Toc2161_1386314816)

[B.6.4FLEXIBILITY PERSPECTIVE 43](#__RefHeading___Toc2163_1386314816)

[B.6.4.1RECORDING of DESIGN DECISIONS / TRADE-OFFS 43](#__RefHeading__110_1666809543)

[B.7VIEW MAPPINGS 43](#__RefHeading___Toc2140_729351093)

[B.7.1For each VIEW MAPPING 43](#__RefHeading___Toc2142_729351093)

[C)References 44](#__RefHeading___Toc2146_729351093)

Document Information

|  |  |  |  |
| --- | --- | --- | --- |
| **Project Name:** | Pacemaker Android Application | | |
| **Prepared By:** | Colm Carew - 20053766 | **Document Version No:** | 1.0 |
| **Title:** | Architecture Description (AD) | **Document Version Date:** | 2016-05-02 |
| **Reviewed By:** | Colm Carew | **Review Date:** | 2016-05-03 |

Distribution List

| **From** | **Date** | **Phone/Fax/Email** |
| --- | --- | --- |
| Colm Carew | 2016-05-02 | N/A |
|  |  |  |

| **To** | **Action\*** | **Due Date** | **Phone/Fax/Email** |
| --- | --- | --- | --- |
| Siobhan Drohan | Review | 2016-05-12 | sdrohan@wit.ie |
| Eamon De Leastar | Review | 2016-05-12 | edealeastar@wit.ie |
| Pierre Peclier | Review | 2016-05-12 | ppeclier@wit.ie |
|  |  |  |  |

\* Action Types: Approve, Review, Inform, File, Action Required, Attend Meeting, Other (please specify)

Document Version History

| **Version Number** | **Version Date** | **Revised By** | **Description** | **Filename** |
| --- | --- | --- | --- | --- |
| **1.0** | May. 02nd | Colm Carew | This document files some of the Architecture Views describing the Android and Play Application Assignment of the Design Patterns module: Pacemaker Android Application. | pacemaker\_architecture.archimate |
|  |  |  |  |  |

Document Overview

The following document shows the Architecture Views of the Pacemaker Android Application for the Design Patterns Module Assignment on Architecture Styles & Patterns into a TOGAF Architecture Description.

# Problem Description

## PROBLEM SCOPE

### PROBLEM SUMMARY

|  |  |
| --- | --- |
| **Project Name** | Pacemaker Android Application |
| **Industry Domain** | Activity Tracking |
| **Problem Description** | A Play Application was created in the Agile Software Development Module. This assessment seeks to create an Android Application which interfaces with the Play Application using the REST API.  The Android Application will mirror the Play Application as closely as possible as well as add some features specific to the Android Application.  The Android Application should:   1. Track and Log activities and basic stats 2. Have progress reports based on completed activities 3. Connect with friends and compare workouts with friends 4. Prescribe workouts based on completed activities   The Android Application should also exhibit and number of code design patterns. |

### DOMAIN GLOSSARY

|  |  |
| --- | --- |
| **User** | A registered Pacemaker User can track their activities, connect with friends, compare workouts and generate prescribes workouts based on their completed activities. The User can log in via colmcarew.com or via the Android Application. Comparing and prescribing workouts are features solely for the Android Application. |
| **Activities** | Activities are exercised based activities which hold data such as date, duration and distance. These figures are then used for prescribing workouts and progress reports. |

## STAKEHOLDERS CONCERNS

|  |  |
| --- | --- |
| **Initiative Sponsors** | Pierre Peclier, Eamon De Leastar, Siobhan Drohan |
| **Project Development Team** | Colm Carew |
| **System Administrator** | Colm Carew |

## REQUIRED QUALITY PROPERTIES

|  |  |
| --- | --- |
| **Performance** | System must have acceptable performance and response time for requests. Proximity of components should remain close to not lose performance due to communication overhead. |
| **Usability** | The Android Application must be simple to understand and use.  The user should be able to navigate through all the Android Application’s features without any impediments. |
| **Executability** | The Play Application should be easily deployable to another server without large issues (true as it is ran from Jar files).  The Play Application should run as cheaply as possibly due to limited resources (true as the Play Application and database are ran from a single Raspberry Pi). |
| **Portability** | The system should be relatively easily built upon for a different platform such as iOS (true due to the heavy implementation of REST) |

# Solution Description

Below are the key characteristics of the solution design.

## ARCHITECTURE VIEWS

|  |  |  |
| --- | --- | --- |
| **VIEW NAME** | **PURPOSE** | **TARGET STAKEHOLDER(s)** |
| **CONTEXT View** | The purpose of this view is to outline the main elements at play in the design of the Pacemaker dashboard:  1.What is the purpose of the system  2.What are the systems features | Data Analyst, CIO |
| **BEHAVIORAL View** | List processing flows at play for the rendering Pacemaker's dashboard. The purpose of this View is to outline:  1.How data gets collected and prepared for storing  2.How analytics data is served to users | CIO  Data Analysts |
| **INFORMATION STRUCTURE View** | Describes how the Pacemaker system stores and manages data.   1. Analytics Data Flow - how the application processes and calculated data 2. Rendering Data Flow – how data is stored and retrieved | Data Analysts  Developers |
| **FUNCTIONAL View** | This specifies the key components of the Pacemaker system and their relationships with one another. | Developers |
| **INFRASTRUCTURE View** | The purpose of this view is to outline the infrastructure behind the design such as servers, databases, applications, how they connect to one another and how the user can connect to the applications. | System Administrators |

## CONTEXT VIEW

### VIEW INTENT

|  |  |
| --- | --- |
| **MODEL NAME** | **PURPOSE** |
| **SYSTEM CONTEXT Model** | To the SCOPE of architecture solution in terms of interdependent / surrounding system components and actors. |
| **SYSTEM FEATURES Model** | To the SCOPE of architecture solution in terms of application capaibilities and . |

### VIEW MODELLING ARTIFACTS

#### SYSTEM CONTEXT MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Context View Type** | To outline what/who will interact with the analytics platform architected. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **System Context Modeling** | This model shows the Achiever Dashboard Analytics system solutions in the center of the model, and its inputs and outputs from/to external factors at the periphery.  All User data candidate for Analytics is collected and correlated from 3 distinct sources of information: PostgreSQL (User Activity / Workouts data), REDIS (User Geo-positioning / Health Tracking data points) and HealthGraph (User Health Profile data).  Resulting datasets (prepared for Analytics Dashboard reporting) are then persisted in a single location which is the authoritative source for reporting, i.e. providing data that can be analyzed under a number of reporting dimensions. |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **Data-driven Style: Data Factory (Data Warehousing)** | User Data Analytics Views are prepared. Data coming from various data sources (relational and non-relational) is going through a number of preparation steps (e.g. cleansing, validation, mapping) with a view to create user-driven analytics datasets.  The Runkeeper Data Warehouse stores and centralizes User Analytics datasets for purposes of reporting.  These datasets are “ready” (suitable) for the Achiever Analytics Dashboard to directly consume information. |
| **Domain Pattern: Shared Database** | User Data Analytics are stored in a shared data repository, accessible from the Achiever Analytics Dashboard. Data can be pulled into the DW or pushed from external applications into the DW.  This architecture is based on a Data Factory Style making use of a PULL mechanism (from data sources), referred as ETL (extract Transform load). |

#### SYSTEM FEATURES MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Context View Type** | To outline what are the main respsonbilities for the components at play. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **System Feature Modeling** | This model shows that the Application is composed of two main components compeltely independent from one another: (1.) Runkeeeper Analytics, (2.) Runkeeper Analytics Dashboard.  (i.e. each module can be deployed and inter-changed completely independentely from one another at runtime without having to redeploy the entire solution).  More specifically, the model shows how features of the Analytics solution are decomposed and allocated to each of the main two components.  These two components share a common database infrastructure component storing Pre-generated (i.e. prepared) User Analytics Views. |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **Data-driven Style: Data Factory (Data Warehousing)** | N/A - Same as above/previous section. |
| **Domain Pattern: Shared Database** | N/A - Same as above/previous section. |

## BEHAVIORAL VIEW

### VIEW INTENT

|  |  |
| --- | --- |
| **MODEL NAME** | **PURPOSE** |
| **Data Preparation** | This model describes the data analytics preparation process. The process is triggered every time a User uploads new data points (e.g. a new activity feed, a modification of heath profile).  The flow described in this model is an asynchronous data processing daemon, observing when new user information is uploaded, then triggers an ETL data job to generate new (i.e. refreshed) User Analytics View in Runkeeper’s Data Warehouse. |
| **Dashboard Rendering** | This model describes the type of tasks the Analytics Dashboard performs to fetch the Analytics Views from the Data Warehouse, and transform the data to render it as Dashboard Views – foe example: a historgram chart, a pie chart, a terrain topology map, a table, other. |

### VIEW MODELLING ARTIFACTS

#### DATA PREPARATION MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Behavioral View Type** | To outline the main flows defining how data gets collected and prepared for purposes of analytics reporting.  To outline Triggers and Events, flow-logic and flow-synchronization. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **Process Flow Modeling** | This model shows how a scheduled batch process monitors periodically for any new feed of data uploaded by an ELITE member. |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **Process Control Pattern - Orchestrated Flow** | An orchestration flow coordinates the preparation of data analytics views. The process can be parallelized but central coordination is required to start processes and synchronize results in the target data warehouse. |

#### DASHBOARD RENDERING MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Behavioral View Type** | To outline the process by which data analytics get fetched and rendered on the analytics reporting dashboard.  To outline intermediate business logic steps manipulating data before it is displayed in the analytics dashboard. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **Process Flow Modeling** | This model shows the sequence of steps collecting pre-generated user analytics views, performing calculations to agregate values, then mapping the results into a data object readily available for consumption by the Web Desktop Dashboard UI. |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **MVC Pattern - Synchronous Request / Response** | A classic Web-server based Model-View-Controller pattern to render dashboard pages based on the data collected & transformed from the data warehouse. |

## INFORMATION VIEW

### VIEW INTENT

|  |  |
| --- | --- |
| **MODEL NAME** | **PURPOSE** |
| **Batch Analytics data flow** | This model outlines the components at play to transform data pulled from three different data sources.  The data flow described in this model is a set of ETL jobs implemented as RDBMS Stored Procedures generating User Analytics Views in Runkeeper’s Data Warehouse. |
| **Dashboard Rendering data flow** | This model presents the propagation of request/response objects throughout application components: triggered by User actions from the Dashboard Web UI, down to the collection of data, to the generation of model views, and rendering of data in the browser / from time of page rendering. |

### VIEW MODELLING ARTIFACTS

#### BATCH ANALYTICS DATA FLOW MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Information Services View Type** | To outline the routes of core application data-in-motion, i.e. how the batch analytics data services takes input to manipulate, transport and ultimately distribute information to target sources.  To outline any data format, timeliness/latency, transactional integrity risks between application components. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **ETL Orchestration Modeling (also loosely referred as Batch Processing).** | To describe the type and roles applications components at play in the Extract phase, the Transform phase, the Load phase of analytics data. |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **Pipes & Filters Pattern** | Implementation of a message processing which is platform-independent, re-usable and extensible to generating future analytics views.  Identify the type of transformation functions and data objects involved in Input/Output of each Pipe/Filter mechanism. |
| **Message Translator Pattern** | Convert message payloads from 3 different sources, using 3 different communications protocols, in 3 different data schemas to one unique data model – i.e. schema of analytics views. |

#### DASHBOARD RENDERING DATA FLOW MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Information Services View Type** | Same as described in section above. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **Data Flow Modeling** | A classic model show the life-cycle, the triggers, and direction of data flows within the Web Dashboard application (only – the analytics preparation component above holds a different and independent process that isn’t related to events or flows of this component). |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **Template View pattern (for Synchronous Web Page Request/Response)**  [https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller] | In this situation, some of the View, most of the Model and Controller logic are implemented on Runkeeper’s JAVA server stack. The Dashboard UI is a combination of “thin” client (in which some decoration is generated by JSP server tags), and Javascript libraries (initialized with JSON objects prepared in the server BEFORE page rendering). When the page is rendered, the JS gets executed and reaches out of the JSON objects to initialize Pie charts, Topology Maps, and other visual components. |

## FUNCTIONAL VIEW

### VIEW INTENT

|  |  |
| --- | --- |
| **MODEL NAME** | **PURPOSE** |
| **Indirection Layers** | To outline the structure of the Dashboard Application (all components included) and identify coupling risks. To describe logical groupings of key components collaborating to achieve a similar function. |
| **Data Provisioning** | To outline what paradigm is adopted to shield application components from the specific of data storage forms/infrastrucutre – i.e. isolating the upper layers from the physical implementation mechanics of lower layers → data persistence model / data abstratcion. |

### VIEW MODELLING ARTIFACTS

#### INDIRECTION LAYERS MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Functional View Type** | To describe the composition relations between components collaborating for the specific purpose of (1,) fetching data (i.e. Data Access Objects - DAO), (2.) aggregating data into Transfer Objects (TO) propagated back up to the Dashboard Client UI. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **Application Structure Modeling** | Used in this situation to identify coupling between layers of components of a same nature. Also used to identify servicing points and data contracts reducing coupling (typically Application Services, API, etc.) |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **Layered Style – Relaxed Variant (includes 2 API service points)** | The variant of the Layared pattern used in the Analytics Dashboard architecture aims to primarily isolate domain model components – but also to eliminate any dependency between data infrastructure concerns and user interface experience, so to prepare for Mobile UI Clients with a minimum of rework. |

#### DATA PROVISIONING MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Functional View Type** | Same as described in section above. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **Application Structure Modeling** | Break down the structure of the domain layer to show the lower level design of data-centric components, with a a specific focus on Data Analytics Access Provider component. |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **3 Object-Relational Data Patterns:**  **- Data Access Object Pattern**  **- Domain Entity Aggregation**  **- Transfer Object Pattern** | Data Access Object to retrieve analytics view datasets from PostgreSQL Data Warehouse.  Domain Entity Aggregation to perform calculation / aggregations on data points retrieved from data analytics views.  Transfer Object to feed Web Client UI JS components (and in the future Mobile Client UI). |

## ARCHITECTURE PERSPECTIVES

### PERFORMANCE PERSPECTIVE

#### RECORDING of DESIGN DECISIONS / TRADE-OFFS

|  |  |  |
| --- | --- | --- |
| .**ID** | **DECISION ITEM** | **DECISION MADE** |
| **#1** | **Need to prevent session time-out for users of the Analytics Dashboard.**  **Response time of Analytics Dashboard must be less than 8 seconds for 5,000 concurrent users.** | **Decision**: A Data Analytics View will not be generated on-demand when an ELITE User connects to a dashboard.  Generating data analytics on-demand is technically feasible but this architectural options doesn’t scale above 2,500 concurrent users given the current infrastructure in place at Runkeeper.  **Trade-off:** Data Analytics will be generated asynchronously by a background process named (Batch Analytics). |
| **#2** | **Some ELITE Users will want to see their stats immediately after upload. At that point, the analytics job will not be yet complete, and the dashboard will not reflect reality.** | According to current usage statistics of simpler Runkeeper dashboards, 95% of Users are not checking their stats immediately after ending an Activity.  However 5% of ELITE Users may want to see their new activity stats immediately reflected after uploading their Health or Activity data into Runkeeper.  **Trade-off**: For these users, an indicator will be displayed on the Dashboard page to notify that their last data upload is in progress. *Eventual Consistency* in this case is a satisfactory option for these Customers. |
| **#3** | **Performance MUST BE independent of processing volume.**  **Throughput of data MUST BE predicatble.** | **Ref to #1** |
| **<#>** |  |  |

### SCALABILITY PERSPECTIVE

#### RECORDING of DESIGN DECISIONS / TRADE-OFFS

|  |  |  |
| --- | --- | --- |
| .**ID** | **DECISION ITEM** | **DECISION MADE** |
| **#1** | **Cost of required processing cycles and memory must be predictable, function of the number of concurrent users.**  **Starting from 5,000 concurrent users, the supporting infrastructure MUST dynamically scale up 100,000 concurrent users by means of infrastructure configuration.** | **Decision**: No net-new scalability strategies will be introduced for the Rendering of the Web Data Analytics Dashboard.  The Dashboard UI is based on a classic Web architecture sourcing data from a shared database, and as such follows the same scalability strategies as any other form of dynamic web-content currently generated by Runkeeper (using HAProxy, Load Balancer and Caching mechanisms).  Runkeeper has the infrastructure and tools in place to deal with Web workload going way beyong 5,000 users.  **Decision:** A new strategy will be defined however to Scale-out the resource-intensive generation of Data Analytics Views.  The Asynchronous Batch Analytics architectural option allows for a better exploitation of server resources (CPU and memory), minimizing the use of the same components and communication paths.  Batch Analytics implementations can be optimzed for repeated processing. Contention can be reduced via replication of ETL jobs.  **Trade-off**: Data processing processing jobs will be distributed over time instead of kick-off on-demand. Some analytics jobs will be prioritized for Users who are logged-in and are waiting to see results immediately. Other jobs will be queued for Users that are not connected. |
| **<#>** |  |  |

### TESTABILITY PERSPECTIVE

#### RECORDING of DESIGN DECISIONS / TRADE-OFFS

|  |  |  |
| --- | --- | --- |
| .**ID** | **DECISION ITEM** | **DECISION MADE** |
| **#1** | **Beyond testing the UI features of the Analytics Dashboard, the architecture MUST allow of load testing so monitoring of service levels of the application can be automated.** | **Decision:** Given above decisions and trade-offs for the Performance and Scalabilty Perspectives, it makes sense to automate Load Testing for Batch Analytics processes since it is the most heavily loaded component of the architecture.  **Decision**: A “load test” batch analytics process creating “dummy” analytics views will be consistently running on Runkeeper’s servers, exercising the server components at all times.  The end-to-end duration of the job will be constently controlled as a Key Performance Indicator of the architecture.  When / if the “load test” job falls below a certain duration window not meeting SLA requirements, a notification will be dispatched to System Admnistrators to adjust server configurations.  **Trade-off**: A new ETL Server will be acquired and act as a distribution node for other data processsing ETL servers.  At a minimum, 4 servers will compose the backend of Runkeeper data analytics (i.e. 1 Main Node, 3 Slaves).  Steps for scaling out further from there-on are, by order of preference: (1.) CPU and Memory additions to ETL server, (2.) Insertion of new node. |
| **<#>** |  |  |

### FLEXIBILITY PERSPECTIVE

#### RECORDING of DESIGN DECISIONS / TRADE-OFFS

|  |  |  |
| --- | --- | --- |
| .**ID** | **DECISION ITEM** | **DECISION MADE** |
| **#1** | **The proposed architecture must be extensible to an Android/IOS version of the Analytics Dashboard with minimum rewrite of the core system logic.** | **Decision**: A Layered Architecture will abstract the presentation layer from how/where User data analytics views are stored.  A data provider component will act as a Facade (gateway) to access Analytics Views from any future Client UI.  **Trade-off**: No Service API will be defined for the first revision of the Analytics Dashboard.  The Web UI Pages will exclusively use Transfer Objects passed via Server-Side Scripting mechanisms (i.e. Web MVC).  The Web UI will not use any Service API. The reason for this is that, since a Facade component acts as a gatway to access Analytics Views, it will be relatively easy to create an API Service sitting on he top of it (i.e. making use of it).  Domain Logic will not change, only the Data Provider, offering Transfer Objects passed via REST to the Mobile UI. |
| **<#>** |  |  |

## VIEW MAPPINGS

### For each VIEW MAPPING

|  |
| --- |
| Not covered in this example AD. |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **<<VIEWPOINT NAME>>** | <<DESCRIPTION>> |

# References

This section to reference external sources of additonal information relevant to the proposed solution design.