|  |
| --- |
| MIDDLE EAST TECHNICAL UNIVERSITY  INFORMATICS INSTITUTE  SM504 – TEAM SOFTWARE PROJECT |
| Ridder |
| Software Design Description (SDD)  - Team D - |
| **Version 1.1** |

Contents

[Approval 3](#_Toc420091613)

[1. Introduction 4](#_Toc420091614)

[1.1. Purpose 4](#_Toc420091615)

[1.2. Scope 4](#_Toc420091616)

[1.3. Context 4](#_Toc420091617)

[1.4. Summary 5](#_Toc420091618)

[2. References 6](#_Toc420091619)

[3. Glossary 6](#_Toc420091620)

[4. System Design 6](#_Toc420091621)

[4.1.1. Identified Stakeholders and Design Concerns 7](#_Toc420091627)

[4.1.2. Logical ViewPoint 9](#_Toc420091628)

[4.1.3. Logical View 9](#_Toc420091629)

[4.1.4. Interface ViewPoint 11](#_Toc420091630)

[4.1.5. Interface View 11](#_Toc420091631)

[4.1.6. Interaction ViewPoint 19](#_Toc420091632)

[4.1.7. Interaction View 20](#_Toc420091633)

[4.1.8. Design Rationale 25](#_Toc420091634)

[4.1.9. Traceability Matrix 26](#_Toc420091635)

List of Figures

[Figure 1 System Components 7](#_Toc419930640)

[Figure 2 Server Components 10](#_Toc419930641)

[Figure 3 Client Components 11](#_Toc419930642)

[Figure 4 Interface between MobileClient and MobileAppService 12](#_Toc419930643)

[Figure 5 Interface between AdminClient and AdminService 15](#_Toc419930644)

[Figure 6 Interface between Crawler and Feedly API 18](#_Toc419930645)

[Figure 7 Interface between MobileClient and Pocket API 19](#_Toc419930646)

[Figure 8 Crawling the content sources with using feedly api 20](#_Toc419930647)

[Figure 9 Register Client 20](#_Toc419930648)

[Figure 10 Get User Categories 21](#_Toc419930649)

[Figure 11 Save User Category Preferences 21](#_Toc419930650)

[Figure 12 Get Entries 21](#_Toc419930651)

[Figure 13 Mark Entry 22](#_Toc419930652)

[Figure 14 Get Categories 22](#_Toc419930653)

[Figure 15 Save Categories 22](#_Toc419930654)

[Figure 16 Get Sources under a Category 23](#_Toc419930655)

[Figure 17 Save Sources 23](#_Toc419930656)

[Figure 18 First Run 23](#_Toc419930657)

[Figure 19 SwipeRight (Add to pocket) 24](#_Toc419930658)

[Figure 20 SwipeLeft (Not Interested) 24](#_Toc419930659)

[Figure 21 Get Entries 25](#_Toc419930660)

List of Tables

[Table 1 Glossary 6](#_Toc420090299)

TABLE 2 STAKEHOLDERS AND CONCERNS……………………………………………………………………………………….….8

[Table 3 Traceability Matrix 27](#_Toc420090300)

**Document control**

|  |  |  |
| --- | --- | --- |
| Version Number | Date | Description |
| 1.0 | May 12nd, 2015 | Baseline SDD |
| 1.1 | May 20th, 2015 | Updated SDD |

# Approval

This document has been reviewed and approved as the Software Design Document of the Ridder Project.

Approved by:

Date:

Approval Signature:

Approved version:

# Introduction

This document, which is related to Software Design Description (SDD) of Ridder, provides necessary definitions to conceptualize and further formalize the design of the software, whose requirements and functionalities were summarized in Software Requirements Specifications (SRS) document prepared by Team D for Ridder. The aim of this document is to provide guidance to a design which could easily be implemented by any programmer reading this report.

The document complies with the IEEE standards (IEEE Std 1016 – 2009).

## Purpose

The purpose of this Software Design Description document is to describe how the software will be structured to satisfy the requirements of the Ridder, mentioned in Software Requirements Specifications (SRS) document prepared by Team D for Ridder. This document explains the design details of the system.

This SDD document is prepared for providing information to both backend and mobile developers in Ridder project.

## Scope

This SDD document will define the high level design and technology decisions for Ridder.

The document defines and describes the design stakeholders, design concerns, related views for said design stakeholders, use of each view, visualization of each view, the architectural constraints of the system, the functional requirements with a significant impact on the architecture and use-case realization, design rationale and a traceability matrix mapping use cases to design views.

## Context

This SDD document consists of four main parts and the last part of them which is related to System Design is divided into 9 sections in order to provide a complete and understandable perception about the design decisions to the stakeholders. The first main part defines the general purpose and scope of the document. In the following two main parts, the references that have been used during the preparation of this SDD document and the glossary presenting the definitions of terms are located.

In the forth main part, firstly the stakeholders and their design concerns are defined along with the selected design viewpoint. After this section, the viewpoints and their related views are defined. The design viewpoints and views, which are according to object orient desing methodology are supported by visual drawings in UML language v2.0. In the last two sections of the forth main part design rationale and a traceability matrix mapping the use cases to design views are located.

## Summary

This SDD document defines the high level design decisions for Ridder project where the Object Oriented Design methodology is applied. More specifically, the document identifies the stakeholders for Ridder project and their design concerns. Considering these stakeholders, three viewpoints are selected. The selected viewpoints, which are mentioned in forth main part of the document are Logical Viewpoint, Interface Viewpoint and Interaction Viewpoint.

For Logical viewpoint, Server component and client component views are described.

For Interface viewpoint, MobileClient\_MobileApp Service Interface view, AdminClient\_AdminService Interface view, Crawler\_FeedlyAPI Interface view and MobileClient\_Pocket Interface view are described.

For Interaction Viewpoint, Crawler\_FeedlyAPI Interaction view, MobileClient\_MobileAppService Interactions view, AdminClient\_AdminService Interactions view and Mobile Client Interactions views are described.

For visualizing the views, UML v2.0 design language is used and while taking design decisions, high cohesion-low coupling principle has been applied wherever applicable.

# References

[1] IEEE Std 1016-2009, IEEE Standard of Information Technology – Systems Design – Software Design Descriptions

[2] Pocket, <https://getpocket.com/>

[3] Feedly API, <http://developer.feedly.com/>

[4] Ridder, Software Requirements Specification Document v1.1, Team D

# Glossary

|  |  |
| --- | --- |
| **Terms** | **Definitions** |
| GUI | Graphical User Interface |
| SDD | Software Design Description |
| SRS | Software Requirements Specification |
| IEEE | Institute of Electrical and Electronics Engineers |
| UML | Unified Modeling Language |
| DMBS | Database Management System |
| API | Application Programming Interface |
| JSON | Javascript Object Notation |
| PHP | Hypertext Preprocessor |

Table Glossary

# System Design

The system is composed of the following components. Note that external systems are not depicted in the figure.



Figure System Components

The main components are:

* Client Components
  + Mobile Client Component
  + Admin Client Component
* Server Components
  + Mobile Application Service
  + Admin Service
  + Crawler

The following sections describe the detailed design of these components with different viewpoints and views. Mainly, client components and server components are described in two separate components.



### Identified Stakeholders and Design Concerns

The overall stakeholders for Ridder project include Project Manager, Product Owner, Sponsors, Backend Developers and Mobile Application Developers, Admin and the Mobile Users of Ridder. The sponsors for Ridder project are Onur Demirörs and Ahmet Çoşkunçay while the Project Manager is Ali Kayhan and Product Owner is Ömer Doğan. Mobile Application Developers in Ridder Project are Önder Altıntaş and Ali Kayhan whereas the Backend Developers are Mert Akıncı and Ömer Doğan. Admin and Mobile Users in Ridder project are the people who will use the Ridder and will mostly have concerns about stability and usability for the server application, mobile application and overall service developed in the Ridder project.

The design stakeholders which should be identified in this SDD document are mainly Mobile Application Developers and Backend Developers but Sponsors, Admin, Mobile Users, Project Manager and Product Owner has also concerns affecting the design decisions. Main concerns related to stakeholders can be seen in table below.

|  |  |
| --- | --- |
| **Stakeholder** | **Concerns** |
| Sponsors | Requirements traceability  Progress tracking |
| Project Manager | Requirements traceability  Schedule Estimation  Progress tracking |
| Admin | Consistency with use cases  Performance, reliability |
| Mobile User | Consistenct with use cases  Performance, realiability |
| Product Owner | Requirements traceability  Progress tracking |
| Mobile Application Developer | Sufficient detail for design  Sufficient detail for mobile application interfaces  Compatibility with existing services |
| Backend Developer | Sufficient detail for design  Sufficient detail for server application interfaces  Compatibility with existing services |

**Table 2 – Design Stakeholders and Concerns**

Considering these concerns, the viewpoints are selected. For sufficient detail for design concerns of Mobile Application Developers and Backend Developers, Logical viewpoint has been selected while Interface Viewpoint has been selected for detail concerns of Mobile Applications and Backend Developers respectively for mobile application and server application. Another reason for Interface viewpoint selection was the compatibility concerns of Mobile Application Developers and Backend Developers. Interface viewpoint also helps for the concern of Requirements traceability of Sponsors, Project Manager and Product Owner.

For the concerns on non-developer stakeholders, Interaction Viewpoint is selected so that they will be able to have an insight about the Ridder and its functioning.

For performance and realiability concerns of the Mobile Users and Admin, high cohesion-low coupling principle has been applied for the views mentioned under Logical ViewPoint.

### Logical ViewPoint

Logical viewpoint of the system describes the classes and the relationship of the classes in a static way. The design elements used in the logical view are classes, associations and dependencies between classes and the multiplicity of the associations. Logical ViewPoint aims to satisfy developer needs to start coding the application. The developers are expected to utilize this initial design and begin constructing the software. The logical view may be required to be fine tuned as the software is being constructed.

UML version 2.0 is used to create the logical view. Logical view is a static class diagram in UML.

### Logical View

#### Server Components

The following diagram shows the logical view of the server components. It depicts the details of Mobile Application Service, Admin Service and Crawler.

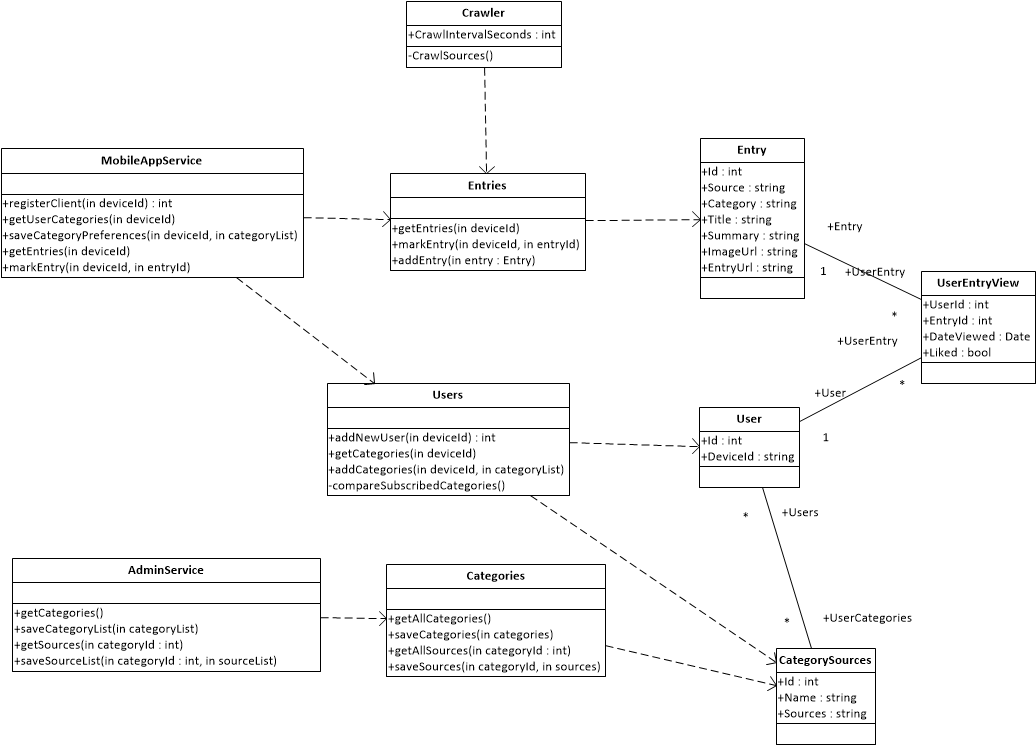


Figure Server Components

MobileAppService is the service interface for the Mobile Client. AdminService is the service interface for the Admin Client. The details of the service methods provided by these services are described in the Interface View.

#### Client Component

The following diagram shows the logical view of the client components. It depicts the details of Mobile Client Component.



Figure Client Components

### **Interface ViewPoint**

Interface viewpoint of the system describes the interaction details of the components with other components. Interface viewpoint is the contract between the serverside developers and the client side developers. Each party provides the information about what the related component exposes as the interface to the other component. The other stakeholders may also utilize this view to have an overview about the decomposition of the system and the interaction of the system components.

UML version 2.0 is used to depict the interface view diagrams. Interface view is a component diagram in UML together with the textual explanations of each interface. Each interface definition includes the methods provided by that interface, the communication protocol, input format and parameters and output format and parameters.

### **Interface View**

#### MobileClient\_MobileAppService Interface

Mobile Client communicates with the server through MobileAppService REST interface. The following diagram shows the interface between the two components.



Figure Interface between MobileClient and MobileAppService

This interface provides the following methods:

###### Register Client

This method allows the mobile application to register the mobile device to the Ridder application. It is used one time for each device. If a request with an existing deviceId is received, this request is ignored by the service.

**Method**

POST /users/register/:deviceId

**Input**

The input “deviceId” is provided with the URL. This is the identifier of the mobile device.

**Response**

{

"userID": 123

}

###### Get User Categories

This method allows the mobile application to get all the categories together with the information whether the category is subscribed by the user.

**Method**

POST /users/categories/:deviceId

**Input**

The input “deviceId” is provided with the URL. This is the identifier of the mobile device.

**Response**

**[**

{

"categoryID": 123,

"categoryName": “Tech”,

"userSubscribed": “true”,

},

{

"categoryID": 456,

"categoryName": “Business”,

"userSubscribed": “false”,

}

**]**

The response includes all the categories defined in the system. Each category item includes an indicator whether the category is subscribed by the user or not.

###### Save User Category Preferences

This method allows the mobile application to subscribe and unsubscribe to categories in the system.

**Method**

POST /users/categories/:deviceId

**Input**

The input “deviceId” is provided with the URL. This is the identifier of the mobile device.

The list of categoryIDs to which the user wants to subscribe is provided as the list of subscribed categories. This list does not contain unsubscribed categories.

The categoryID values should be of type integer.

**[**

123,

456

**]**

**Response**

No specific response is returned other than the HTTP 200 Status Code.

###### Get Entries

This method allows the mobile application to get entries based on the subscribed categories of the user.

**Method**

POST /users/entries/:deviceId

**Input**

The input “deviceId” is provided with the URL. This is the identifier of the mobile device.

**Response**

**[**

{

"entryId": 98765,

"Title": “Google’s self driving cars have been in 11 accidents”,

"Summary": “The viability of self-driving cars will likely be debated and scoffed at for a long time, but at least we know they’re not distracted by their cellphones. In an article on Backchannel, Google revealed details about how its cars work and some of the mishaps they’ve gotten into...”,

“ImageURL”: <http://cdn1.tnwcdn.com/wp-content/blogs.dir/1/files/2014/04/GoogleCar-520x245.jpg>,

“URL”: “http://thenextweb.com/google/2015/05/11/googles-self-driving-cars-have-been-in-11-accidents-because-humans-are-dumb/”

},

....

**]**

The response includes at most 20 entries. Mobile application should send another request to get more entries.

###### Mark Entry as Interested/Not Interested

This method allows the mobile application to mark an entry with the interest of the user. An entry can be marked as “Interested” or “Not Interested” by the user, so that Ridder processes the entries for statistical purposes and for deciding to show or not to show the entry to the same user again.

**Method**

POST /users/markentry/:deviceId

**Input**

The input “deviceId” is provided with the URL. This is the identifier of the mobile device.

The entryIDs together with the interest flag is provided as the input.

**{**

“entryId”: 98765,

“interested”: “false”

**}**

**Response**

No specific response is returned other than the HTTP 200 Status Code.

#### AdminClient\_AdminService Interface

Admin Client communicates with the server through AdminService REST interface. The following diagram shows the interface between the two components.



Figure Interface between AdminClient and AdminService

This interface provides the following methods:

###### Get Categories

This method allows the admin client to get all the categories defined in the system.

**Method**

POST /admin/categories

**Input**

There is no input parameter for this method.

**Response**

**[**

{

"categoryID": 123,

"categoryName": “Tech”

},

{

"categoryID": 456,

"categoryName": “Business”

}

**]**

###### Save Categories

This method allows the admin client to create, update and delete categories in the system.

**Method**

POST /admin/categories

**Input**

It is the list of categories to be added, updated or deleted. The input is processed based on the following rules:

\* If the category information contains categoryID and categoryName, this item is processed as an update request. The name of the category in the system having this categoryID is updated.

If a category cannot be found in the system having the provided categoryID, the method returns an error.

If a category can be found in the system having the provided categoryID, but another category exists with the provided categoryName, then the method returns an error.

\* If the category information contains only categoryName, this item is processed as an insert request. A new category is created in the system.

If another category exists with the provided categoryName, then the method returns an error.

\* If the category information contains categoryID and delete, this item is processed as a delete request. The category is deleted from the system.

If a category cannot be found in the system having the provided categoryID, the method returns an error.

**[**

{

"categoryName": “Gaming”

},

{

"categoryID": 123,

"categoryName": “Technology”

},

{

"categoryID": 456,

“delete”: “true”

}

**]**

In the sample request above,

* A new category with the name “Gaming” will be created,
* The name of the category with the categoryID 123 will be updated to “Technology”,
* Category with the categoryID 456 will be deleted.

**Response**

No specific response is returned other than the HTTP 200 Status Code.

###### Get Sources

This method allows the admin client to get all the sources under a category defined in the system.

**Method**

POST /admin/sources/:categoryId

**Input**

The input “categoryId” is provided with the URL. This is the identifier of the category.

**Response**

**[**

"feed/http://feeds.arstechnica.com/arstechnica/science",

"feed/http://feeds.engadget.com/weblogsinc/engadget"

**]**

###### Save Sources

This method allows the admin client to add/update and delete sources under a category defined in the system.

**Method**

POST /admin/sources/:categoryId

**Input**

The input “categoryId” is provided with the URL. This is the identifier of the category under which the sources will be added/updated/deleted.

This method accepts an input with the list of sources for a specific category. The provided list with this method is accepted as the whole list. Any source which is not provided will be deleted from the system. Any source which does not exist in the system will be added a new source under category.

**[**

"feed/http://feeds.arstechnica.com/arstechnica/science",

"feed/http://feeds.engadget.com/weblogsinc/engadget"

**]**

**Response**

No specific response is returned other than the HTTP 200 Status Code.

#### Crawler\_FeedlyAPI Interface

The system communicates with Feedly through Feedly API REST interface. The following diagram shows the interface between the crawler component and the Feedly API.



Figure Interface between Crawler and Feedly API

Feedly API is the interface provider and the Crawler component of the system is the consumer. Therefore, the description of this interface can be found at [3].

#### MobileClient\_Pocket Interface

The system communicates with Pocket through Pocket API REST interface. The following diagram shows the interface between the crawler component and the Feedly API.



Figure Interface between MobileClient and Pocket API

PocketAPI is the interface provider and the Ridder Mobile Client is the consumer. Therefore, the description of this interface can be found at [2].

### **Interaction ViewPoint**

Interaction viewpoint of the system describes the objects and the communication of the objects in a dynamic way. The design elements used in the interaction view are object instances of the classes on a sequence diagram and the request and response messages. Interaction ViewPoint aims to satisfy developer needs to start coding the application. The developers are expected to utilize this initial design and begin constructing the software. The interaction view may be required to be fine tuned as the software is being constructed.

UML version 2.0 is used to create the interaction view. Interaction view is a dynamic sequence diagram in UML.

### **Interaction View**

#### Crawling the content sources via Feedly API



Figure Crawling the content sources with using feedly api

feedlyAPI object in the figure is an instance of the API provided by Feedly, which is a feed processor cloud based service. The details of this API can be found at [3].

#### Mobile Client – Mobile App Service Interactions



Figure Register Client



Figure Get User Categories



Figure Save User Category Preferences



Figure Get Entries



Figure Mark Entry

#### Admin Client – Admin Service Interactions



Figure Get Categories



Figure Save Categories



Figure Get Sources under a Category



Figure Save Sources

#### Mobile Client Interactions



Figure First Run



Figure SwipeRight (Add to pocket)



Figure SwipeLeft (Not Interested)



Figure Get Entries

### **Design Rationale**

In this section, a list of major design decisions is provided with the justification of why these decisions have been taken.

#### Ridder does not directly process the feeds from original feed sources. It uses 3rd party feed APIs, such as Feedly.

* The purpose of the Ridder application is not to process the feed sources, but provide articles to the end users based on the interests of the users. Feedly and similar applications are already specialized in processing the feed sources.

#### Mobile Client Application does not directly consume the 3rd party feed APIs, such as Feedly.

* Currently, Feedly is the only source of entries for Ridder. But, in later releases, it is possible to add additional sources and use other 3rd party APIs. Therefore, Mobile Client Application would be unaware of the 3rd party APIs.

#### Ridder does not redirect client requests to Feedly. Instead, it has a Crawler component to collect and store entry information from Feedly.

* When the Mobile Client Application accesses the list of articles from Feedly, Ridder does have no idea about which entries are read by the user, which entries the user is interested in and so on. Although direct consumption would still allow the user to save the entries in the Pocket account, this would invalidate some Ridder functionalities.

### **Traceability Matrix**

|  |  |
| --- | --- |
| **Use Case** | **View** |
| UC-MBL-01 | MobileClient\_MobileAppService Interface (Save User Category Preferences) |
| UC-MBL-02 | MobileClient\_MobileAppService Interface (Get User Categories, Get Entries) |
| UC-MBL-03 | MobileClient\_Pocket Interface |
| UC-MBL-04 | MobileClient\_Pocket Interface |
| UC-MBL-05 | MobileClient\_MobileAppService Interface (Mark Entry as Interested/Not Interested) |
| UC-MBL-06 | MobileClient\_MobileAppService Interface (Mark Entry as Interested/Not Interested) |
| UC-ADM-01 | AdminClient\_AdminService Interface |
| UC-ADM-02-1 | AdminClient\_AdminService Interface (Get Categories) |
| UC-MBL-02-2 | AdminClient\_AdminService Interface (Save Categories) |
| UC-ADM-02-3 | AdminClient\_AdminService Interface (Save Categories) |
| UC-ADM-02-4 | AdminClient\_AdminService Interface (Save Categories) |
| UC-ADM-03-1 | AdminClient\_AdminService Interface (Get Sources) |
| UC-ADM-03-2 | AdminClient\_AdminService Interface (Save Sources) |
| UC-ADM-03-3 | AdminClient\_AdminService Interface (Save Sources) |
| UC-ADM-03-4 | AdminClient\_AdminService Interface (Save Sources) |
| UC-ADM-04 | AdminClient\_AdminService Interface |
| UC-ADM-05 | AdminClient\_AdminService Interface |
| UC-ADM-06 | AdminClient\_AdminService Interface |

Table 3 Traceability Matrix