

Contents

[INTRODUCTION 3](#_Toc13558811)

[PURPOSE 3](#_Toc13558812)

[SCOPE 3](#_Toc13558813)

[GLOSSARY 4](#_Toc13558814)

[Definition 4](#_Toc13558815)

[Acronyms 4](#_Toc13558816)

[Abbreviation 4](#_Toc13558817)

[REFERENCE DOCUMENT 4](#_Toc13558818)

[DOCUMENT STRUCTURE 4](#_Toc13558819)

[ARCHITECTURAL DESIGN 4](#_Toc13558820)

[OVERVIEW 4](#_Toc13558821)

[SELECTED ARCHITECTURAL STYLE AND PATTERNS 5](#_Toc13558822)

[Server 6](#_Toc13558823)

[Storage 7](#_Toc13558824)

[Client 7](#_Toc13558825)

[APPLICATION STRUCTURE 10](#_Toc13558826)

[COMPONENT VIEWS 10](#_Toc13558827)

[RUNTIME VIEWS 10](#_Toc13558828)

[ALGORITHMS DESIGN 11](#_Toc13558829)

[USER INTERFACE DESIGN 11](#_Toc13558830)

[REQUIREMENTS TRACEABILITY 11](#_Toc13558831)

[IMPLEMENTATION, INTEGRATION AND TEST PLAN 11](#_Toc13558832)

# INTRODUCTION

## PURPOSE

The purpose of the Design Document (DD) is to provide a description of the design of a system fully enough to allow for software development to proceed with an understanding of what is to be built and how it is expected to be built. The Software Design Document provides information necessary to provide description of the details for the software and system to be built

It is a technical document that exposes techniques, decisions, implementations, components and the architecture that we decided to adopt in the **QDocs** application system development. In this introduction chapter we provide a description of the application and for what it was thought to be used, some definition and the structure of the entire document.

The technical aspects of the **QDocs** implementation start in the second chapter, titled << Architectural Design >>.

## SCOPE

The **QDocs** app is a file-storage mobile application which help users to organize, visualize and, above all, find in a faster way their personal stored files. In fact, there are many files that users use rarely and after some time they no longer remember where they were placed. **QDocs** allows user to basically associate a stored file to a specific and unique QR-code that can be printed and placed in a site of the “real world” that is related to that file. When the QR-code is scanned, the related file is instantly shown on the smartphone. In order to show how this application can be used in real word we provide some usage examples:

1. << A student takes notes on his notebook and he wants to extend his notes with some digital articles or some digital book pages. He can upload the articles on his **QDocs** storage and print the generated QR-code on the personal notebook. When he must study and want to read the articles, he can easily scan the QR-code with the application scanner and immediately the file appears on the smartphone. >>
2. << Since nowadays more and more electronic tools are sold with digital instruction booklet, **QDocs** can become very useful for keeping associate the tool with its instruction booklet. A simple example can be the following, suppose to buy a camera and suppose that after long time you don’t remember what the functionality of a specific button is, very often you don’t remember where the booklet was stored. So to overcome this problem you can store the instruction booklet on the **QDocs** storage and print its QR-code on the camera and then whenever you have to access this document you can easily scan its QR-code (that you know is placed on the camera) and instantly read the document. >>

Obviously, the application can be used as any other cloud-storage application like Dropbox, Google Drive and so on since it allows users to manage their own files (e.g. uploading and/or deleting files), you can also create directories for a clearer storage procedure. The representation of the images is directly managed by the application itself, without using external application, the same for the audio. You can also save your online files on the internal storage of the device such that you whenever you have to open that file you have not to download it each time.

In order to use the application, you must register to it and create your own account, all the accounts are separated such that you cannot access other user’s files.

## GLOSSARY

### Definition

* **QDocs:** Application
* **Cloud**: Physical storage located somewhere that can be accessed through a network connection.
* **Client/User:** Person that uses the **QDocs** application.
* **Firebase:** Mobile development platform.
* **Internal Storage:** This refers to the internal storage of the device that is using the application.

### Acronyms

* **DD:** Design Document.
* **API**: Application Programming Interface.
* **DB:** Database.
* **UML:** Unified Modeling Language.
* **MVC:** Model View Controller pattern.

### Abbreviation

## REFERENCE DOCUMENT

## DOCUMENT STRUCTURE

# ARCHITECTURAL DESIGN

## OVERVIEW

The goal of this chapter is to analyse and describe the architecture implemented in the **QDocs** application. This chapter is structured as follow:

* SELECTED ARCHITECTURAL STYLE AND PATTERNS: this first paragraph introduces the general architecture used for developing the application, analysing all parts involved and their connection/communication.
* APPLICATION STRUCTURE: in this part we focus the attention on the graphic representation of the whole architecture, with different levels of granularity, through UML diagrams.
* COMPONENT VIEWS: this paragraph provides a more detailed analysis on the class structure of the mobile application, providing all necessary class diagrams and highlighting all the classes’ interactions.
* RUNTIME VIEWS: this paragraph provides a more detailed analysis on the interactions-flows for the more important action that can be performed in the application, such as login, registration, scanning, etc.
* ALGORITHMS DESIGN: this paragraph provides a description of the more significant algorithms implemented in the developing of the application, such as the update list algorithm, etc.
* USER INTERFACE DESIGN: the goal here is to provide the whole lifecycle of the application’s screens through apposite diagrams
* REQUIREMENTS TRACEABILITY: this paragraph provides all **QDocs’s** requirements analysing in detail which are the activities that are in charge to guarantee them.
* IMPLEMENTATION, INTEGRATION AND TEST PLAN: here all information about how the testing is performed are provided.

## SELECTED ARCHITECTURAL STYLE AND PATTERNS

[[ describe patterns used such as MVC, firebase event listener ]]

[[ abstract pattern (storage adapter) ]]

The **QDocs** application was developed using the 3-tier client-server architecture [Figure 1].

“3-tier architecture is a client-server architecture in which the functional process logic, data access, computer data storage and user interface are developed and maintained as independent modules on separate platforms.”

* PRESENTATION TIER: Occupies the top level and displays information related to services that the system can provide. This tier only communicates with the application tier through internet request/response. The Mobile application **QDocs** is in this tier.
* APPLICATION TIER: Also called the middle tier, logic tier, business logic or logic tier. This tier is pulled from the presentation tier. It controls application functionality by performing detailed processing, such as calculations, logical decisions, data and model manipulation. This is basically the server application that provides APIs used by the presentation tier software. It communicates with both presentation tier and data tier
* DATA TIER: Houses storage servers where information is stored and retrieved. Data in this tier is kept independent of application servers or business logic. For this tier we used the services offered by Firebase: Realtime Database and Cloud Storage.

Immagine che contiene screenshot

Descrizione generata automaticamente

Figure 1: General 3-tier architecture

In the specific case of **QDocs** application the Presentation Tier corresponds to the Android mobile application, the Application Tier to the Firebase server and the Data Tier to the Realtime Database and Storage.

The following sub-paragraphs will provide more details about how each tier was implemented for the **QDocs** application.

### Server

The server represents the main backend logic, it allows multiple users to interact with it simultaneously and keeping them separated. From the technical point of view, the software was developed using the Firebase development platform providing a set of APIs that hides to the client all the backend logic.

The following list provides all the APIs included in the server:

* AUTHENTICATION API: the server can directly handle the authentication of the users or forward the authentication mechanism to other external services like Facebook and Google, in both cases provides a set of APIs that allows users to make request for registering, logging-in, etc.

It stores all user’s information such as usernames, passwords, personal image and so on.

* CLOUD STORAGE API: since in the server a Cloud Storage service is included, it provides a set of APIs that allows users to interact with their own cloud filed such as upload, download, create new folders and so on.
* REALTIME DATABASE API: since making request directly to the storage may require too much time, the idea was to provide also a Realtime Database service that is queried by the client whenever it want to retrieve static information about the stored files in the cloud, so without updating anything. For this reason, the server provides a set of APIs used for listening on database information.
* CLOUD FUNCTIONS API: backend logic used to react on files upload/delete operation and keeping the Realtime DB updated.
* LOGGING: It logs all the interaction between users and the server.

[[ Firebase image ]]

### Storage

The storage layer is provided by the Firebase infrastructure (Google cloud) and in our case is basically composed by two main parts:

* CLOUD STORAGE: this is basically the cloud location where all the users’ files are stored, this is provided by the Google cloud.

[[ storage structure image ]]

* REALTIME DATABASE: not relational database provided by Firebase that is directly associated to the cloud storage and kept congruent with it, it allows a faster access to the stored data for read-only operation.

[[ database structure image, tree ]]

### Client

The client is represented by the Mobile Application itself [**QDocs**], it is a Native mobile app implemented in Java for the Android platform.

[[ QDocs logo ]]

From the architectural point of view the app was developed following the MVC pattern (Figure 2), as you can see this pattern is composed by 3 main parts:

* MODEL: The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.

In this specific case contains the model of files, directories, users and so on, and the main logic used for some processes like authentication, downloading, uploading and so on, that are kept separated from the part responsible of manage the graphic of the screens.

* VIEW: The view means presentation of the model in a particular format. In this case represents the screen appearance coded in xml files. An xml file is static representation of the app’s screen or of a simple component, that can be reused more times.

For each screen of this app there is at least one xml file representing it, in some case more than one file are used for a clearer representation and reuse.

* CONTROLLER: The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model. These elements are represented by all the Activity/Fragment objects where each of them is responsible to inflate a specific xml file.



Figure 2: General Model-View-Controller pattern

Differently from the general MVC pattern, here, there is not so great distinction between view and controllers because in this case it is the controller itself that listen to user’s action/event and react accordingly. In this case the view is basically the static part inflated when the screen is loaded in the memory and all the changes that may occur to the screen are directly handled by the controllers themselves. So for this reason you can see the true pattern implemented in **QDocs** application in [[ FIGURE ]] .

[[ MVC applied to QDocs ]]

#### Controllers

In this specific context the controllers are represented by Activity/Fragment objects. Each controller is a class responsible of creating the screen associated to it, the invocation of lifecycle-methods is directly controlled by the OS through a callback model. In the following [Figure 3] and [Figure 4] you can see the lifecycle of Activity and Fragment objects respectively.

The following diagrams show the important state paths of an Activity/Fragment. The square rectangles represent callback methods you can implement to perform operations when the Activity/Fragment moves between states. The coloured ovals are major states the Activity/Fragment can be in.

Immagine che contiene testo, mappa

Descrizione generata automaticamente

Figure 3: Activity lifecycle

Immagine che contiene screenshot

Descrizione generata automaticamente

Figure 4: Fragment lifecycle

## APPLICATION STRUCTURE

## COMPONENT VIEWS

[[ Class diagrams in details (methods) and their interactions ]]

## RUNTIME VIEWS

[[ Sequence diagrams (more important) showing interaction among activities (and server)]]

# ALGORITHMS DESIGN

[[ provide examples of algorithm implemented ]]

# USER INTERFACE DESIGN

[[ provide whole mobile app lifecycle ]]

# REQUIREMENTS TRACEABILITY

[[ use case diagram ]]

[[ describe which activity is associated to each requirement ]]

# IMPLEMENTATION, INTEGRATION AND TEST PLAN

[[ describe how tests are performed ]]