

Assignment:

Architekturdokumentation

**Student:** Jacob Müller

**Email:** Jacobmueller11@gmail.de

**Date of birth:** 10.02.1995

**Matriculation number:** 800913

**Academic institution:** Knowledge Foundation @ Reutlingen University

**Study program:** Master of Science Professional Software Engineering

Contents

[Introduction 2](#_Toc71969206)

[Architectural Pattern 2](#_Toc71969207)

[Layered Architecture 3](#_Toc71969208)

[Architectural Decisions 5](#_Toc71969209)

[Structure of Corona Documentation 7](#_Toc71969210)

[References 9](#_Toc71969211)

# Introduction

This assignment is created for the Software Architecture class of the Professional Software Engineering Masters Class of the Knowledge Foundation Reutlingen. The goal of this Assignment is to conduct an analysis of the architecture of the Corona Warn App (CWA), by analyzing four different aspects of the CWA.

The report will proceed as follows. First, the CWA is analyzed based on the architectural pattern used. Next the layered architecture of the verification server is analyzed and documented. Afterwards three distinct architectural decisions are analyzed. Ultimately the overall documentation is evaluated based on industry documentation standards.

# Architectural Pattern

1) Model View Controller (MVC)

For the UI architecture, the **MVC** pattern is applied. The pattern divides the application into three parts. The model contains the information about the domain model, the View contains the presentation model and the Controller contains the behavioral elements. There is no specific problem stated in the documentation, that is supposed to be solved by the application of the MVC[[1]](#footnote-1). Most likely the explanation is omitted due to the prevailing importance of MVC and its subsequent implementation within most web services. Therefore the general advantages are described.

It is an industry standard which is used in most web application due to the named advantages. MVC increases the maintainability, scalability and enabling customization by separating the application in three parts. [[2]](#footnote-2) All points are relevant to the CWA as it needs to be scalable to fit peek needs for covid information. Maintainability is also crucial for the CWA as it needs to function correctly with moderate amount of effort. Alternatives to the MVC could be a monolith architecture, which would lead in higher maintenance efforts and less efficient development. By using the pattern, various architectural principles are supported. By dividing into the three modules, a separation of concern is realized. Further, it supports the single responsibility principle, as all different parts have a specific responsibility. The distinction also supports information hiding, as not all possible information within each MVC entity is shared among the others. The Model layer is also in charge of the contained data and supports data sovereignty. Ultimately it supports standardization, as most web application support this architectural pattern and enables layering.

2) Client-Server Design

Within the solution architecture documentation a graph is shown with the key architectural elements (Figure 1 high architecture overview).[[3]](#footnote-3) Within this overview a Corona warnapp Server is shown. Within the droid CWA app documentation, the architectural element is called the mobile client. This shows the presences of a traditional **Client-Server design** pattern. There is no specific problem stated, that is solved by this design. Most likely, the client-server pattern is used due to its many advantages. The Client Server pattern reduces the redundant data duplication, as the data is stored on the server side, it makes sharing of resources more efficient and it increases the maintainability by splitting the process of an application among multiple machines. On top of this it enables better security measures, as authorization checks can be implemented centrally.[[4]](#footnote-4) Similarly to the MVC, also the Client Server pattern has not been thoroughly justified. Most likely it is also due to the wide applicability and status as industry standard. Next to the described advantages of the client server pattern, it comes with certain drawbacks. It can lead to overloaded server. However, given the cloud deployment, these service are most likely scalable. Further, critical server can have failover problems. If a server gets down, it might create harmful impact on the whole application, due to its central importance.

By using the Client-server design, the following architecture principles are supported. It supports the single responsibility and separation of concern principles, as the client and server functionalities are divided and own a specific purpose. Further, it is a common standard nowadays and ultimately keeps things simple.

3) Microservice architecture

The third architectural pattern described is the **Microservices** architecture. Although the term Microservice has not been explicitly named in the documentation, the subsequent architecture displayed in the overview and the referenced services are implementing a Microservices architecture. A Microservice architecture describes an application setup, in which services are implemented, deployed and tested independently.[[5]](#footnote-5)

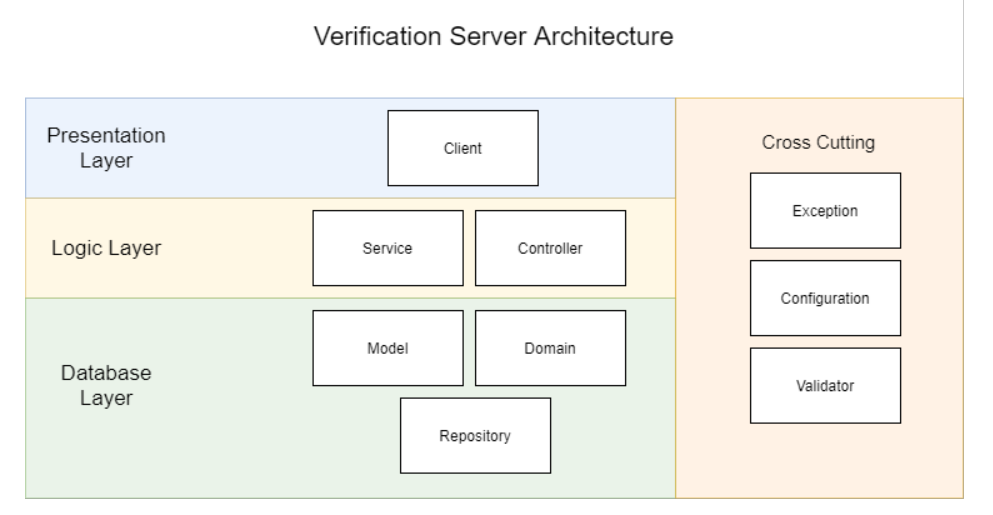
It is not documented why this approach is being implemented. However, the cloud native setup combined with the other advantages make the Microservices architecture a reasonable option. Contrary to the Microservice architecture, a monolith could have been built, but the disadvantages of the monolith would overweigth the disadvantages of Microservices. The disadvantages of Microservices are complex setup, the need for an automated deployment and the difficulty in debugging.[[6]](#footnote-6)

The Microservice uses the following principles. It separates concerns and hides information in between the independent services. Further it minimizes redundancies and allows for convenient layering.

# Layered Architecture

In order to derive the layered architecture from the verification server, the different packages within the repository are clustered into the different layers in the figure below.

Figure 1



The responsibilities of the different layers is described in the figure below.

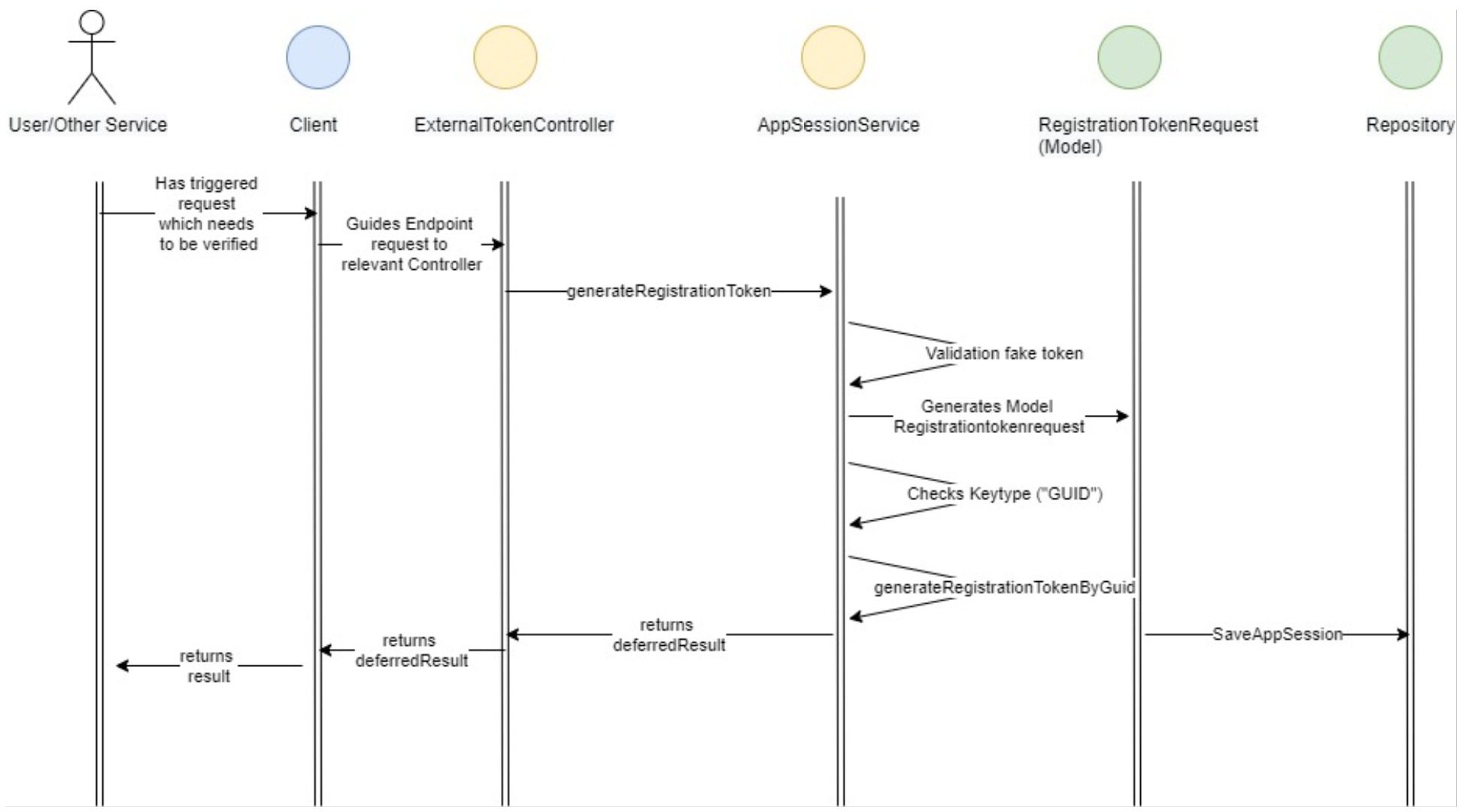
Figure 2 three tier architecture [[7]](#footnote-7)

|  |  |
| --- | --- |
| Layer | Responsibilities |
| Presentation | * Representing the interface to the user. In this case it symbolizes the entrypoint of other services within the CWA to this service, * Is available via the Client Package |
| Logic | * Contains the business logic * Uses the data inserted in the presentation layer * Triggers CRUD operations on the Database layer * Is implemented in the Service and Controller layer |
| Database | * All persistent information is stored and managed here * Is implemented in the Model, Domain and Repository Package |

The Cross Cutting block includes packages that are not directly linked to one specific layer and are therefore displayed separately on all layers.

In the picture below is an exemplary Uml sequencediagramm with an exemplary run-through of the verification server. On the top the responsible package is named, below are the different sequential steps with regards to the verification process. In this example, there is an request to make a registration with a GUID key. It starts with a user or another service making a request to the client (Client-Server architecture), which subsequently redirects the request to the controller class. In this case the example guides towards the externalTokenRequest controller. This Controller redirects to the appSessionService via the generateRegistrationToken. In the Service various Model entities as e.g. Registrationtokenrequest are used. Also various validations take place in the service. Ultimately a registrationToken (by Guid) is created and the AppSession is saved to the DB via the repository package (Adapter). The result (deferredResult) is then returned to the initial request made to the client. As shown by the color code, a MVC architecture can be observed (only with an Endpoint as View instead of a GUI).

Figure 3



# Architectural Decisions

In this chapter three architectural decisions of the CWA are analyzed and compared to the documentation requirements in the ISO 42010 standard. Given that there was no log of architectural decisions found in the different architectural documents, decisions described within the main architecture documentation files are analyzed.

The first architectural decision is found in the solution architecture document and explains the **necessity of the Verification server**.[[8]](#footnote-8) In order to understand the decision the documentation in the verification-server/docs/architecture-overview.md is analyzed.

The table below displays the contained information from the CWA documentation in comparison to the requirements according to the ISO norm 42010.

Figure 4

|  |  |
| --- | --- |
| **ISO norm** | **CWA Documentation** |
| Evidence of consideration of alternatives | * Not available |
| Record key decisions | * Validation of upload requests of users * Provide information about status of Sars-Cov-2 tests * Obtain proof document for health authorities |
| Contains information items | * No unique identifier or owner for the decision * no statement of the decision * no correspondences or linkages concerns to which it pertains * Only narrow correspondences or linkages to affected AD elements * No rationale linked to the decision * No forces and constraints on the decision * No assumptions influencing the decision * No considered alternatives and their potential consequences |

There has been **no documented alternative** to the Verification server decision.

Regarding the **recording of the key decision**, many relevant points have only briefly been stated. For instance the reasoning for the component is threefold. The verification server is necessary because it validates upload requests of users, provides information about the covid status and gives proof documents to health authorities. However, the points are only briefly been mentioned and further evaluation have been omitted. For instance, the commercial implications have been neglected as well as the project management relevancies. On top of this the implication for the key stakeholders is not outlined in at all.

Next to this, it also misses various **information items**. It does not contain a unique identifier for the decision nor does the decision have an owner. There is no detailed constraint overview resulting from this decision, nor are alternatives described. Next to the overview graph of the architecture, the verification server documentation does not link other Architectural modules to the documentation. Ultimately, the documentation is describing the functionality of the verification server, but does not explain the decision for the architecture.

The next architectual decision analysed is the **decision to integrate a CWA Server**. In order to understand the decision the documentation in the cwa-server/docs/architecture-overview.md is analyzed.

|  |  |
| --- | --- |
| ISO norm | CWA Documentation |
| Evidence of consideration of alternatives | * Not available |
| Record key decisions | * Enable users to take part in exposure notification framework * Compliance to the specification |
| Contains information items | * No unique identifier or owner for the decision * no statement of the decision * no correspondences or linkages concerns to which it pertains * YES: correspondences or linkages to affected AD elements * No rationale linked to the decision * No forces and constraints on the decision * No assumptions influencing the decision * No considered alternatives and their potential consequences |

This documentation has a similar level of detail as the documentation of the prior architecture decision. It also does **not contain considerations of an alternative architecture** to the CWA-server. It also does only gives a high level justification for the architectural decision. The information items are similarly narrow as in the prior architecture, with the exception, that there are various cross references to other architecture files in this documentation.

The next architectual decision analysed is the decision how to **integrate the CWA mobile client**. In order to understand the decision the documentation in the cwa-app-android/docs/architecture-overview.md is analyzed.

|  |  |
| --- | --- |
| ISO norm | CWA Documentation |
| Evidence of consideration of alternatives | * Not available |
| Record key decisions | * Exposure tracing management * Exposure risk calcualtion * Test result access * Diagnosis key submission |
| Contains information items | * No unique identifier or owner for the decision * no statement of the decision * no correspondences or linkages concerns to which it pertains * YES: correspondences or linkages to affected AD elements * No rationale linked to the decision * No forces and constraints on the decision * No assumptions influencing the decision * No considered alternatives and their potential consequences |

Similar to the two architecture decisions described above, this documentation also neglects information about the decision in various aspects. It does not contains information about potential alternatives and omits similar information regarding the decision as the two examples above. Due to the similarity in depth to the other decisions, the differences are analyzed. Contrary to the other decisions, this documentation has a higher level of detail in explaining the necessity for the component by illustrating the four key reason on a detailed level. It also has an extended section with key references to external used libraries.

Conclusively, all three architectural decisions lack the list of alternatives and information on the decision. In order to evaluate the architectural decisions, the list of alternatives would have been helpful to proof considerations of other potential solutions. It might have been omitted to keep the architecture documentation small and not overcomplicate the reading. In order to derive alternatives, the lead architecture or any team member, which was part of the architectural steering could have provided information. An example for an alternative to the verification server could be a verification within the CWA-server and not externally in a verification server.

# Structure of Corona Documentation

The structure of the CWA is being displayed within the Repository on github. The following observations were made with regards to the structure of the CWA documentation:

One of the first observations made, when analyzing the documentation of the CWA, is the **general toolset** and features used for displaying the documentation. In general WIKIs like confluence are suitable and convenient for documentation. However the CWA documentation is part of the repository in a folder with redundant markdown and pdf files. This makes the navigation and understanding of the documentation overwhelming and complex at first sight. The ease of navigation is further diminished, as there is additional documentation markdowns outside of the documentation folder. There are specific architectural documentation file within the distinct services in other folders. For example, within the cwa-server/docs is additional architectural documentation in the architecture.md file. This combination of centrally and decentrally stored architectural information hinder both, transparent insights and efficient information retrieving. Further, the decentrally stored architecture documentation have not been referenced in the solution architecture documentation. For example it has been stated that “The documents will be linked here, as soon as they are available”. Therefore the documentation is outdated, as the documents are already available. On the positive side however, there is cross-references available to the other documentation files in the same folder, enabling convenient technical navigation through the documents. A migration to a traditional WIKI system could enhance the transparency and readability of the documentation.

In order to evaluate the general structure of the architectural documentation, the solution architecture markdown file is analyzed. The architecture is analyzed based on the ISO norm 42010 and a estimation is to be made, whether the documentation fulfills the standard. Therefore the mandatory acceptance criteria are evaluated.

The figure below compares the structure of the documentation with the structure of the ISO norm 42010 on a high level:

Figure 5 ISO42010 Fulfillment Overview

|  |  |  |
| --- | --- | --- |
| Mandatory ISO42010 requirement | | Presence in CWA Documentation |
| Name of Architecture | | “CORONA-WARN-APP SOLUTION ARCHITECTURE” |
| System of Interest | | “CWA” |
| Supplementary information | | Miscellaneous |
| Architecture evaluations | | N.a. |
| Rationale for key decisions | | Available in introduction |
| Stakeholder description & categorization | | Implicit referenced and not described |
| Concerns of CWA | | available |
| Concern Stakeholder tracing | | N.a. |
| Viewpoint - concern match | Implicit | |
| Viewpoint - rational match | Implicit | |
| Viewpoint - stakeholder match | N.a. | |
| Sources | | available |
| known inconsistenciers | Not mentioned | |

In order to display the shortcomings, the missing mandatory documentation points are analyzed:

In order to satisfy the ISO standard, **architecture evaluations** must be included into the documentation. Given that there is no evaluation available, the CWA solution architecture either has not been evaluated, or the evaluation has not been documented. Given the scope of the CWA, most likely architecture evaluations took place and were omitted. In order to improve the quality of the documentation, architecture documentation could be integrated by the leading architectural committee.

In the introduction, the **rational** for the key architectural decisions is stated. However, the structure of the introduction is not conscious. For example, the paragraphing in the introduction section seems to only partly follow a structural logic.[[9]](#footnote-9)

Further, **a stakeholder description** is a required part of an architectural documentation. This has only implicitly been done in the documentation. The stakeholder User has been used 56 times in the solution architecture documentation whilst not been described in detail. The developer has only once been referenced. Therefore the documentation is not fulfilling the stakeholder description requirement. In order to get a better understanding for the motivation of specific architectural decisions, an explanatory paragraph could be inserted to introduce the key stakeholder.

The specific **Viewpoints** documentation in accordance with the ISO standard has partly been included. For example there is a an excessive documentation on the security architecture addressing the data privacy and security concerns. This Viewpoint is realized into a dataflow view in figure 3 in the solution architecture documentation. The explicit connection to certain stakeholders however has been omitted in the security section as well.

**Sources** to external parties and third party libraries is present and is cross-referenced within the documentation.

No **inconsistencies** are mentioned in the documentation, however no inconsistencies have been encountered during the high level analysis of the CWA documentation.

Due to the difficult navigation through the documentation, it can be not granted that all items marked not available are non-existent. However, if the item cannot be found after close analysis, most likely other consumer of the architecture would not be able to find it either. Therefore it can be seen as missing.

# References

Literature:

Morales-Chaparro, R., et al. "MVC web design patterns and rich internet applications." *Proceedings of the Jornadas de Ingenierıa del Software y Bases de Datos* (2007): 39-46.

Oluwatosin, Haroon Shakirat. "Client-server model." *IOSRJ Comput. Eng* 16.1 (2014): 2278-8727.

Thönes, Johannes. "Microservices." *IEEE software* 32.1 (2015): 116-116.

1. https://github.com/corona-warn-app/cwa-app-android/blob/main/docs/architecture-overview.md [↑](#footnote-ref-1)
2. Morales-Chaparro, 40 [↑](#footnote-ref-2)
3. https://github.com/corona-warn-app/cwa-documentation/blob/master/solution\_architecture.md [↑](#footnote-ref-3)
4. Oluwatosin, 67 [↑](#footnote-ref-4)
5. Thönes, 116 [↑](#footnote-ref-5)
6. https://www.theserverside.com/answer/What-are-some-of-the-disadvantages-of-microservices [↑](#footnote-ref-6)
7. https://www.ibm.com/cloud/learn/three-tier-architecture [↑](#footnote-ref-7)
8. https://github.com/corona-warn-app/cwa-documentation/blob/master/solution\_architecture.md [↑](#footnote-ref-8)
9. https://github.com/corona-warn-app/cwa-documentation/blob/master/solution\_architecture.md [↑](#footnote-ref-9)