The non-existent average individual

Blaauw, Frank Johan

Based on:

Blaauw, F. J., & Emerencia, A. (2015). A Service-Oriented Architecture for Web Applications in e-Mental Health: Two Case Studies. In *2015 IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA)* (pp. 131–138).

Chapter 4

Architecture and Infrastructure of HowNutsAreTheDutch and Leefplezier

The primary source of medical data for the specific domain of e-mental health are online questionnaires. Most applications in this domain focus on viewing, filling out, and managing these questionnaires (Danziger, 1990). Many distinct applications have been developed in recent years, most of them starting from scratch and running in isolation. In this chapter, we design and propose a generic architecture for e-mental health applications based on the HowNutsAreTheDutch (HND) and Leefplezier platforms described in Chapter 3. These platforms are used on a large scale with over 13 000 users combined. By abstracting functionalities into reusable interfaces, we can maximize data interoperability while minimizing application-specific code to facilitate rapid development of e-mental health applications. We set forth to answer the question of whether it is possible to design one such generic Web application, to identify problems that would have to be tackled, and to highlight topics of debate.

Applications designed for research projects tend to focus primarily on collecting data and not on having a clean architecture or incorporating reusable components. This neglect can be attributed to the fact that the success of a research project is usually not measured by the quality of the application, but rather by the data that is collected. Furthermore, since many such research projects have predefined budgets and time limits, application development that supports reuse tends to be an afterthought. The result is that, rather than incorporating reusable components, these projects create one-off applications, each starting from scratch and learning the same lessons, and each being costly to develop.

In light of these concerns, we believe that incorporating external service components and reusable application logic in a generic architecture will reduce development time and cost compared to those of one-off projects. In this chapter, we

propose a generic architecture for e-mental health applications based on two case studies. This architecture takes advantage of service-oriented architecture (SOA) and service-oriented computing (SOC) paradigms for its core functionality. We explain the architecture for both case studies, and we give an overview of the pros and cons of the decisions made during the design process.

4.1 Service-Oriented Architectures in E-mental Health

The importance of accessible and affordable care for mental disorders cannot be overstated. For instance, approximately $41.2\,\%$ of the Dutch population experiences at least one Diagnostic and Statistical Manual of Mental Disorders (DSM) disorder in their lifespan (Bijl et al., 1998). The symptoms experienced range from feeling tired and having a lack of interest, to experiencing strong feelings of depression. Such episodes can have a great influence on the well-being one experiences.

Although numerous papers on e-mental health research platforms exist, only a few of them shed light on the platform's architecture. Griffiths and Christensen (2006) give an overview of several on-line mental health assessments and Internet based treatments. The authors conclude that the e-mental health assessments appear to provide a promising means for mental health self management. Donker et al. (2013) review several mobile applications used to deliver mental health assessments. In their research they conclude that mobile phones are well suited to give basic advice on mental health.

SOA and SOC are two relatively new concepts in the field of e-mental health and health research. Research platforms are often basic and do not consider any elaborate (service-oriented) architecture. Only a few studies describe the use of these concepts. Kazi and Deters (2013a, 2013b) describe an architecture for a diary study that measures pain which applies various SOC principles. They found that the combination of Web technology and a representational state transfer (REST) architecture allows for the development of a reliable and secure health information system. Some research exists with regards to safely storing health data in a SOA environment. Fan et al. (2011) describe DACAR, a secure data storage by means of a single point of contact (SPOC). DACAR uses a SOA to support integration of eHealth services.

The described works show that SOA and SOC are viable concepts because of the requirements they can fulfill in eHealth platforms, both functional (such as authentication / authorization and storage of medical data) and non-functional (such as interoperability and privacy). However, the SOA and SOC concepts have not seen widespread adoption for e-mental health applications focusing specifically on questionnaires. As a result, these applications are created as isolated, one-off solutions.

Our e-mental health applications, HND and Leefplezier, are both partly based on SOA concepts, which we will analyze in this chapter. For each of these applications, we analyze the degree of SOC applied and motivate the design decisions. Furthermore, we present a generic architecture based on these case studies. The first of the two case studies (HND) was intended as a one-off project, the second case study, however, was designed to use more reusable components (Leefplezier). Analyzing the differences and commonalities between these case studies gave us insight in which elements could be important in a generic e-mental health platform.

4.2 Two Case Studies

In Chapter 3, we described some of the background of the research from the HND and Leefplezier studies. Here we describe and analyze the architecture of these platforms. Both platforms enable e-mental health research on a moderate to large scale. HND was designed to obtain insight in the psychological well-being of the Dutch population. Leefplezier focuses on well-being of elderly people in the Netherlands. An overview of the architecture and degree of SOC of both applications is provided in Section 4.2.1 and Section 4.2.2. To provide some basic context we give a short introduction on both platforms and their backgrounds. For a more elaborate description see Chapter 3. A comparison of both architectures is performed to derive a generic architecture. Section 4.2.3 describes the technical details of both platforms.

4.2.1 The Architecture of HowNutsAreTheDutch

HND is a Web application created as the platform for a national Dutch mental health research project (Blaauw, van der Krieke, Bos, et al., 2014; van der Krieke, Jeronimus, et al., 2016). In the HND application, people can fill out various psychological questionnaires and automatically retrieve feedback on these questionnaires. This feedback is a comparison with the average Dutch population combined with personalized diagrams and figures, depending on the questionnaire.

As is described in the previous chapter, HND offers two types of studies: (i) a cross-sectional study and (ii) an individually repeated ecological momentary assessment (EMA) study (referred to as *diary study*). In the cross-sectional study, participants can fill out a number of questionnaires from a predefined set of questionnaires. The diary study focuses on within-person variance by letting the participants fill out the same questionnaire multiple times. In the diary study, participants are measured for thirty consecutive days by filling out a questionnaire, consisting of 43 questions, three times per day (van der Krieke, Blaauw, et al., 2016; van der Krieke, Jeronimus, et al., 2016). The participants in this study are notified via SMS when they should fill

out their next questionnaire. The SMS contains the link to the Web application, on which the questionnaire can be filled out. After filling out a cross-sectional questionnaire or after participating in the diary study, people get automatically generated feedback.

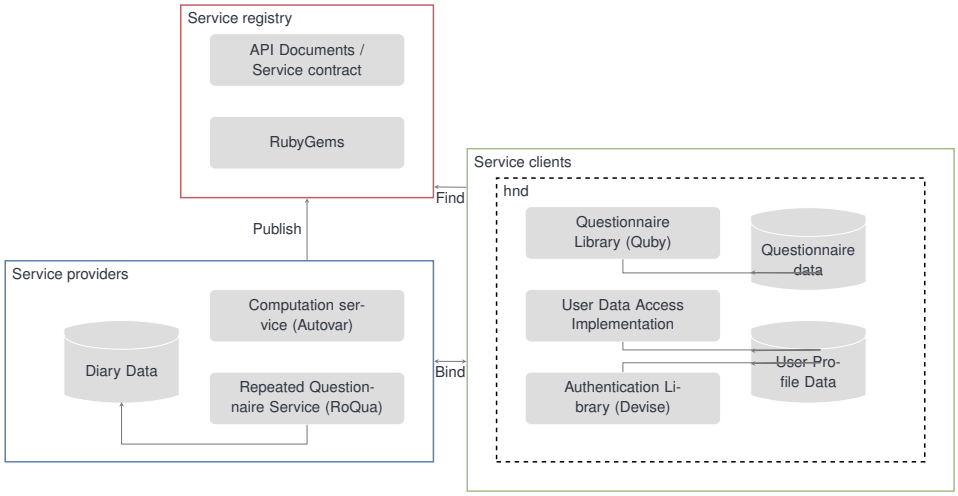

Figure 4.1: General overview of the HowNutsAreTheDutch architecture.

Figure 4.1 gives an overall overview of the HND architecture. The architecture consists of an application built upon various local libraries and several external services. Most functionality in HND is built into the application itself. Authentication and questionnaire conduction is implemented using two off-the-shelf libraries. The questionnaire data is stored in a database separate from the authentication information and user specific data (demographical information and email addresses) in order to limit the risk of a security breach affecting both types of data. The diary study uses two service-based components. Firstly, a service is used to perform the scheduling of the questionnaires. This service schedules the notifications and does a callback to HND whenever a participant should be notified. Secondly, when participants have completed their diary studies, personal feedback is automatically calculated. In order to calculate this feedback, an existing service is used (Autovar; Emerencia et al., 2016). Autovar calculates a vector autoregression (VAR) model and returns a JavaScript object notation (JSON) containing results to HND. Autovar calculations can be started using a Web service. The feedback based on the results returned by Autovar is rendered using client-side JavaScript.

4.2.2 The Architecture of Leefplezier

Leefplezier focuses on enhancing, sustaining, and providing feedback on the well-being of elderly people (Blaauw, van der Krieke, de Jonge, & Aiello, 2014; Jeronimus et al., 2017). Although the goals and procedures of the Leefplezier project are similar to those used in HND (see Chapter 3), their architectures have some important differences. Compared to HND (depicted in Figure 4.1), the Leefplezier architecture

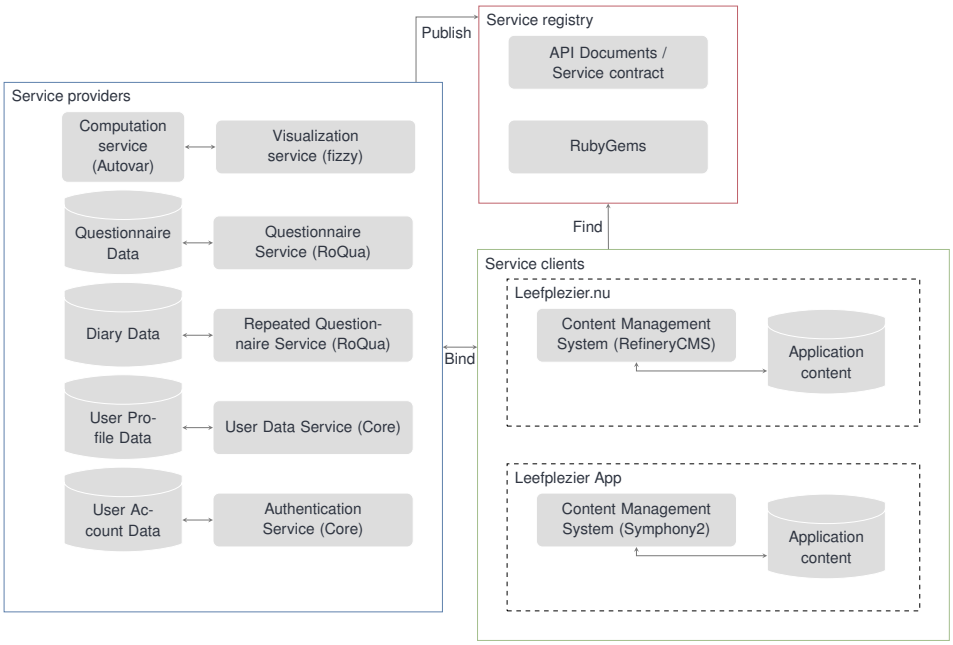

Figure 4.2: General overview of the Leefplezier architecture.

(depicted in Figure 4.2) depends more on external service-based components. Both the cross-sectional part and the diary study part are implemented using SOC. Authentication of users is provided by a single sign-on (SSO) approach. Leefplezier participants authenticate themselves to a third-party application and are provided access Leefplezier via keyed-hash message authentication code (HMAC). HMAC uses a cryptographic hash function in order to verify the authenticity of a message and thereby authenticating a participant. Storage of demographical data is provided by a separate service. The calculation of diary study results is performed using the Autovar service. However, in Leefplezier, the Autovar service (as described in Section 4.2.1) is wrapped in a visualization service. This visualization service takes care of running Autovar and also renders the feedback of the questionnaires. The only

part customized for Leefplezier is the content management system (CMS) and the business logic to connect the services.

4.2.3 Technical Overview

The case studies share several commonalities. For example, communicating and authenticating with external services and their encompassing frameworks. The communication with external services is performed using a REST architecture. A REST architecture uses the hypertext transfer protocol (HTTP) for accessing the external services (e.g., GET, PUT, POST, and DELETE methods). These method invocations are sent over a secure socket layer (SSL) connection to ensure privacy and security. Binding to these services is secured by using server-to-server authentication. Three methods of authentication are used (depending on the service): (i) HTTP basic authentication, (ii) open authorization (OAuth) and (iii) HMAC. HTTP basic authentication and OAuth are used for the general communication with the service providers. HMAC is used to allow external applications to access Leefplezier, such as the SSO service.

Both case studies are implemented in the open-source Web framework Ruby on Rails. Ruby on Rails has numerous plug-ins available (called *Gems* in Ruby parlance). These Gems are self-contained packages providing functionality that can be used by the application including them. HND and Leefplezier use Gems in order to provide a clean interface to the SOC services. The HTTP communication to the services is abstracted by the Gems, by providing a simple application programming interface (API) to the developers. Gems can be distributed via standardized services, such as *RubyGems*[1]. RubyGems is in this case used as a service registry. An overview of these separate components from a classical SOC perspective is shown in Figure 4.3.

The components of both architectures that have not yet been discussed, including the applications used to provide some of the functionality (e.g., Quby and RoQua), are elaborated in the next section.

4.3 Comparison

Both HND and Leefplezier are used for collecting mental health data and as such share several commonalities in their architectures. However, there are some noteworthy dissimilarities, mainly for the following two reasons. Firstly, the applications were built in succession. We developed HND prior to developing Leefplezier.

[1] Website: `https://rubygems.org`

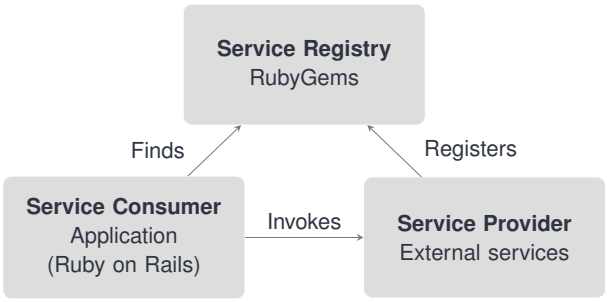

Figure 4.3: High-level service-oriented architecture using Ruby on Rails.

The experience of building and maintaining HND has influenced our decisions in designing the Leefplezier architecture. Secondly, the operating environments for the two applications were different. HND focused on sampling a larger percentage of the Dutch population. In order to find a large group of participants for HND, seeking publicity for HND was done using several newspaper articles, magazine articles, radio interviews, and other media attention (see Chapter 3 for more details). When media attention was sought to announce new features of the application, this imposed strict deadlines on the development of HND and could cause a sudden increase in the number of participants. The important requirements for HND were therefore speed of development and scalability, often at the expense of maintainability and reusability. Leefplezier, on the other hand, was aimed at a smaller target audience, that is, elderly people associated with certain care organizations. For Leefplezier, scalability is therefore not a primary concern. The lack of reusable code from the HND project forced us to create Leefplezier from scratch and to reevaluate the design priorities. For Leefplezier, two of the main requirements were established to be reusability and maintainability, by relying more on off-the-shelf components and existing services.

Table 4.1 shows the degree to which SOC concepts were applied in the case studies. 'Custom' describes the components that were custom built for the target application. SOC describes the components used via external services. The remainder of this section explores, discusses, and compares the impact that the different requirements have on the architectural design of the applications.

4.3.1 Data Security

Often e-mental health platforms can contain delicate and personal information that should not be publicly available. Keeping this information safe is a primary re-

quirement for any e-mental health platform. An important factor for achieving data security is the data storage location. One can choose to store the data locally or outsource storage to an external service. In the latter case, one could opt for choosing an external service that is specialized in storing medical data.

In HND, the questionnaire data is stored locally. However, to reduce the impact of a security breach, the personal information (i.e., demographical data, email addresses, and authentication details) is stored separately from the questionnaire data. For both types of data a different database management system is used and the data is stored on physically different servers.

Leefplezier uses a specialized medical service known as RoQua to store its data. RoQua is a company that offers different services for storing personal data and for storing anonymous questionnaire data. Authentication to Leefplezier is provided by a SSO service. By using an external SSO provider, Leefplezier is not responsible for securely storing the login information.

4.3.2 Conducting Questionnaires

In HND, we used an existing application for administering questionnaires, named Quby[2]. Quby is an application designed for administering questionnaires in the field of mental health. Currently, the Quby application only supports the mode where we are responsible for hosting the data. Hosting the data in a self-managed database did give us more flexibility since we had direct access to the data. The trade-off for this flexibility is having to assume responsibility for securely storing the questionnaire data.

[2]Source available at `https://github.com/roqua/quby_engine`.

Table 4.1: The degree of SOC applied in the case studies.

Topic	HowNutsAreTheDutch	Leefplezier
Authentication	Custom	SOC
Data management	Custom	SOC
Questionnaires	Custom	SOC
Diary questionnaires	SOC	SOC
Content management	Custom	CMS
Result calculation	SOC	SOC
Result visualization	Custom	SOC
Newsletter list management	Custom	SOC

The Leefplezier platform uses Quby as a service. The Quby used in Leefplezier is hosted by an organization specialized in conducting medical questionnaires, namely the company mentioned Section 4.3.1; RoQua. RoQua offers a platform that hosts Quby called RoQua-ROM. RoQua was used for all interactions with the hosted Quby, by means of REST Web services. These Web services were exposed natively through a Ruby Gem.

Although the service-oriented approach of Leefplezier for conducting questionnaires absolves us from the responsibility of having to worry about storing questionnaire data, it is not without caveats. The main issues with the Leefplezier approach are development time and flexibility. Implementing questionnaire functionality using an external Web service could save development time. However, since the available services were created to be generic, they might not exactly provide the functionality needed by the platform to be implemented, and some additions might be required in order for the system to be usable. This was the case for the Leefplezier application, and such additions added up to a significant part of the time spent on the development of the Leefplezier.

4.3.3 Feedback Generation

The two case studies attempt to incentivize participants to fill out questionnaires by providing them with individual and useful feedback based on their questionnaire data. In particular, the diary study allows for detailed analysis of the changes in behavior of a person over time.

The feedback for the cross-sectional study phases is relatively straightforward. We present the participants with a graph of their score compared to the maximum obtainable score and compared to the score of the average of the other people in the study. These graphs are calculated synchronously as they do not require lengthy statistical analysis.

In contrast, the diary study feedback is complex. We perform time series analysis using VAR models to elucidate causal relationships between symptoms measured in the questionnaires. In order to fit a VAR model automatically, we use Autovar (Emerencia et al., 2016). Autovar is written in the statistical programming language R, so its code cannot execute directly on the Ruby platforms. We therefore interface with Autovar over a RESTful interface, using the OpenCPU platform (Ooms, 2014). The OpenCPU approach is used for both the HND and Leefplezier applications and provides the added benefit of separating the statistical calculations from the application logic in a way that facilitates the reusability and scalability of the Web applications.

4.3.4 Feedback Visualization

Various graph types are used to visualize the questionnaire results (e.g., bar graphs, line graphs, scatter plots, pie charts, etc.). The graphs showing this feedback are generated using the *Highcharts* and data-driven documents (D3) libraries. These are JavaScript libraries for rendering graphics. HND uses both libraries to visualize the feedback graphics. Since the only way to access HND is by using the Web interface, these front-end JavaScript can be implemented directly into the application.

For the Leefplezier application, however, it is insufficient to only use client-side rendering. In Leefplezier, participants should also be able to view their feedback on a mobile application. For this feedback to be generated in a generic and abstract way, we developed *Fizzy*, a reusable service provider that generates graphs based on questionnaire data.

4.3.5 Content Management

Not all contents of a Web application are static. Some pages may require regular additions, removals, or other updates of text and media. Rather than editing the hypertext markup language (HTML) files on the server by hand, applications frequently facilitate editing their contents as an integral part of the functionality of the Web application using a CMS. A CMS is designed to be easy to use for people without a technical background. The main advantage of using a CMS is the fact that multiple users are able to edit the contents of the application as it is running. Without a CMS, any changes in the contents of the application falls to the responsibility of the application developers. The main disadvantage of using a CMS is that it incurs a higher up-front cost from the development team as the application needs to be adapted to integrate the CMS. Nevertheless, in most situations the application developers do not remain available for the length of the project to maintain the system for minor textual changes. In these cases a CMS is inevitable.

HND was implemented without the use of a CMS. All changes that were to be made to the application were given to the development team, who directly edit the HTML source of the pages. The decision for not having a CMS was made during the development phase, in order to save time on development and have high flexibility. In our experience, having a CMS, at least for the most frequently changing pages of the website, is worth the extra up-front cost. It would have been more time efficient if HND was implemented with a CMS, compared to the current approach.

The Leefplezier does integrate a CMS. The application uses *Refinery*CMS[3]. Our experience with having to maintain the HND website and being tasked with adding

[3]Source available at `https://github.com/refinery/refinerycms`.

content throughout the year influenced our opinion in that it is good practice not only to separate contents from design but also to allow those contents to be updated directly by the users that create that content.

4.4 Requirements of E-mental Health Applications

From our two case studies, we extracted several architectural commonalities and heuristics. Furthermore, we identified three main non-functional requirements that are applicable to e-mental health applications in general: (i) *data security and patient privacy*, (ii) *maintainability of the e-mental health platform*, and (iii) *availability and reliability of the platform for data collection*.

4.4.1 Data Security and Patient Privacy

Security and privacy are two requirements inextricably tied to any application that deals with personal health information. Law and regulations dictate the use of numerous security standards for collecting and storing medical information. SOC can help achieving a secure and privacy aware system by restricting data access for service clients. The service provider could be the only service retaining the data, being a SPOC for data retrieval. The provider can be used as a security authority and restrict access to the data by only exposing specific services to specific service clients (Fan et al., 2012). The separation of concerns in SOC also makes a system inherently more secure. A separation of, for example, personal data from patient health data reduces the chance of compromising all data, if one of the service providers were to contain a security breach. Some organizations have policies stating that data needs to be stored in-house. Such policies make the use of generic SOC solutions impossible. The last important aspect is the reuse of the service its security standards. In order to store medical data, a system should have various certifications (e.g., ISO 27001 or NEN 7510 in the Netherlands) as imposed by law and regulations. Reusing a system that has these certifications is more time efficient.

4.4.2 Maintainability of the E-mental Health Platform

Researchers in e-mental health projects are mostly concerned with starting the data collection as soon as possible and would prefer to spend little time and effort in having to develop and maintain the application. The concerns of these researchers can be met by combining off-the-shelf components rather than directing a one-off monolithic solution for which they are solely and wholly responsible. This work-

flow is accelerated by using SOC because it can reduce the lead time for developing a new project while outsourcing most of the software maintenance.

The distribution of responsibilities imposed by SOC also increases maintainability. For example, by separating the concerns of application logic from those of the statistical data analysis, software developers and researchers can adapt the parts of the project in their field of expertise. Moreover, because of the dynamic binding between the SOC components, external services can be updated without needing to redeploy the application.

4.4.3 Availability and Reliability for Data Collection

The punctual and time-dependent nature of many medical studies imposes high demands on system availability and reliability. These demands are magnified when the system features a patient-facing front-end. Especially in studies where patient compliance is an area of concern, additional technical difficulties that may discourage participation can be detrimental or even prohibitive to performing a study successfully. For instance, when data are collected in real-time, or relatively often (e.g., in a diary study), one requires a platform to be reliably available. Availability concerns can be mitigated by SOC since separation of concerns can help increasing the availability of critical parts of the system. These parts can be replicated decreasing the chance of a failure to occur.

4.5 Proposed Architecture

Based on the analysis of the architectures of HND and Leefplezier, we propose a generic architecture for e-mental health applications, shown in Figure 4.4. In this figure, the colors indicate the likelihood of an individual application having to write application-specific code in the layer (green is very likely, blue is likely, and red is very unlikely). Based on the lessons learned from both platforms, this architecture encompasses the elements common to applications in e-mental health research.

The architecture set out in Figure 4.4 consists of five layers. From top to bottom, the layers decrease in volatility and in interactivity with the user. For example, while the presentation layer consists of mostly user-facing code, the lower layers contain the back-end of the application (hence the decrease in user interactivity). The decrease in volatility is because the top layers are deemed application specific, with layouts and graphical user interfaces that may vary from project to project, while the lower layers contain functionality that is more generic, more complex, unlikely to change frequently, and often accessed through service-oriented mechanisms.

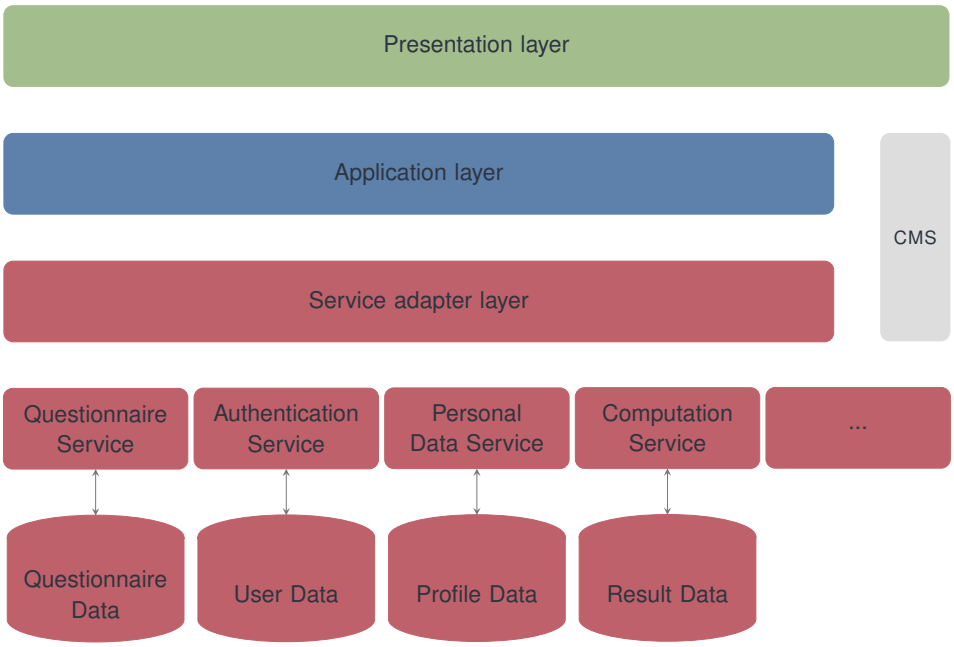

Figure 4.4: A layered overview of a generic architecture for e-mental health applications.

The topmost layer is the *presentation layer*. The presentation layer houses the user-interface (e.g., layout templates and themes) of the application. We envision this layer to be the most application specific and to feature little code reuse, as each application is likely to want to use its own theme and looks. Figure 4.4 shows a connection between this layer and the CMS, as certain assets may be modified through a CMS, for example.

The second layer is the *application layer*. The application layer features the application logic as well as other code that lets the presentation layer interact with the other layers. Examples of application-specific components may include the specifications for which pages are accessible to users, the diary study protocol, and which plots to include on a results page. Examples of reusable components in this layer may include, for example, code to generate certain result plot types or account management pages.

The *service adapter layer* serves to abstract the application from the service-specific code for interfacing with the back-end services. This layer functions as an adapter by providing the application with a non-changing interface to interchangeable external services. This layer may host service-specific code to support certain services or service types. This code should be restricted to this layer and only needs to change

when the external components change, not when the application changes. We envision a scenario where these adapter definitions are managed by a service registry.

The fourth layer is the *service layer*, which contains the service providers. We have identified four primary services as the common subset of services used by applications in e-mental health: (i) questionnaire services, (ii) authentication services, (iii) personal data services, and (iv) computation services. These services should operate agnostic of the application and are often external to the application. The additional flexibility provided by the service adapter layer warrants that external services may be used as interchangeable parts, allowing one to switch to a different questionnaire manager without having to rewrite any application code (for this particular example, migrating questionnaire data is a separate issue but an issue nonetheless).

The fifth layer is the *data layer*, which contains the data collected by the research application. For this layer we firstly propose to (physically) separate user identifiable information (i.e., user account information and demographics) from the other data, to the best extent possible. The reason is that a data leak is more likely to happen in either one of these services than to both at the same time. Secondly, the process of selecting external services should weigh the availability of features for exporting data that meet the research requirements.

A separate component in the architecture is the CMS. In the proposed architecture one should take into account which contents of the application are likely to change, and if there are such contents, to provide a means for regular users to manage this content themselves. In practice, this often translates to integrating a CMS. Because, in theory, any form of an administrative interface could also be used to manage, e.g., which questionnaires are selected for a study or which external services should be used, the CMS layer may have tie-ins to the application layer and service adapter layer. With the CMS being such an integral part of the application, it is often difficult to abstract the CMS from the rest of the application in a way that facilitates, for example, interchangeability. The architecture presented in Figure 4.4 satisfies the requirements listed in Section 4.4. A short explanation for each requirement is listed below.

Data security and patient privacy: the generic architecture absolves the application owner from being solely responsible for data security by assigning responsibility to each of the external services for managing their share of the data. Security could further be increased by using data storage services from providers that specialize in information security or by using encrypted data storage systems.

Maintainability of the e-mental health platform: applications following the proposed architectural guidelines are maintainable because they (i) separate application-specific code from reusable code (e.g., the layout is separated from the back-

end logic), (ii) separate code by functionality and responsibility (e.g., only the computation service is responsible for statistical computations, and for nothing else), and (iii) separate static content from dynamic content, allowing direct user edits for the latter category (i.e., by integrating a CMS).

Availability and reliability for data collection: although our architecture does not directly influence the availability and reliability of the platform, it was designed to allow for redundant availability between any of its individual components, providing a high level of reliability.

4.6 Discussion and Concluding Remarks

We analyzed the architecture of both HND and Leefplezier with a particular focus on SOC. From this analysis, we distilled a generic architecture that could be used for other online e-mental health platforms. Despite the fact that we performed our analysis only on these two platforms, we believe that the architecture presented and the caveats described for implementing service-oriented architectures can serve as blueprint for developing e-mental health research applications in general. In fact, we recently released a new e-mental health diary study platform[4] in part based on the principles and heuristics described in this chapter.

One of the major advantages of our proposed architecture is the loose coupling of the service components, provided by SOC. Loose coupling allows for easy reuse of the services when implemented in a generic way. In our proposed architecture, the service adapter layer enforces the loose coupling between the application and the services it uses. Reuse of existing systems can save development time for successive studies and can eventually save costs. For medical data, one advantage of SOC could be to outsource the responsibility for keeping data secure.

However, the increase in security and maintainability comes with a decrease in flexibility, performance, and testability. Flexibility is reduced as the service client has no direct influence on the functionality exposed by the service provider. In many cases, the service client might not have access to run detailed queries on the data. Performance is reduced due to the extra layer of abstraction (i.e., the API of the service provider) between data and application. Data requests that invoke an external service induce latency and bandwidth overhead compared to direct database connections. Especially in procedures that invoke multiple repeated queries, the performance penalty can be significant. Testability is reduced as traditional testing approaches may be less suitable for applications that use SOC (Canfora & Di Penta, 2009). This may result in applications being unable to validate the functionality of

[4]Website (in Dutch): `https://nemesisdagboekonderzoek.nl`.

external services, having to trust that their descriptions are accurate and up-to-date. Even in cases where testing external services is theoretically possible, this may be too slow and costly for practical use, especially for services that charge on a per-request basis.

As e-mental health applications frequently have a significant amount of overlap in terms of their main functionality (i.e., recording questionnaires and providing feedback), using a SOA fosters the reuse of these components and meets the most important requirements of e-mental health platforms. The use of specialized services can help application owners to focus on performing research without being burdened with the implementation details of every single aspect of the application. If more applications followed these guidelines, then perhaps the interfaces could be standardized, as other formats in medical informatics have. Components could be reused not only between projects within the same hospital, but between different hospitals worldwide.