On Architecture of BodyInNumbers Exercise and Wellness Health Strategy Framework

Petr Brůha
Department of Computer Science and Engineering , Faculty of Applied Sciences, University of West Bohemia
Univerzitní 8
Pilsen, Czech Republic
+420 377 632 412, 30100
pbruha@kiv.zcu.cz

Roman Mouček
Department of Computer Science and Engineering , Faculty of Applied Sciences, University of West Bohemia
Univerzitní 8
Pilsen, Czech Republic
30100
moucek@kiv.zcu.cz

P. Volf, L. Šimečková, O. Šťáva
Department of Computer Science and Engineering , Faculty of Applied Sciences, University of West Bohemia
Univerzitní 8
Pilsen, Czech Republic
30100
{volfpe,simeckol, ostava }@students.zcu.cz

ABSTRACT

They are many risk factors decreasing overall human physical and cognitive performance and increasing incidence of chronic diseases. It is very beneficial for any society to map, discuss and cope with these factors. This can be supported and evaluated by designing, developing, testing and using suitable self-management health systems. One of these systems is the BodyInNumbers exercise and wellness health strategy framework that allows experimenters to collect various heterogeneous health related data in a highly organized and efficient way. Thanks to its success and daily use, new requirements related to better security, scalability and maintainability of its architecture have emerged. The aim of this work is to present advances and changes in the architecture of the BodyInNumbers health strategy framework mainly focusing on new definition of user roles, optimization of the system deployment, and orchestration of the system components. As a proof of concept, a Kubernetes cluster prototype has been used to demonstrate the improved architectural solution.

CCS Concepts

• **Applied computing → Information systems → Computer systems organization→Software and its engineering→Security and privacy**

Keywords

health information systems; software architecture; body in numbers software system; health related data; brain data; physical performance; cognitive performance; data security

1. INTRODUCTION

Overeating, extreme drinking, smoking and physical and cognitive inactivity are well-established risk factors decreasing human physical and cognitive performance and increasing incidence of chronic diseases that present an enormous burden on society [1][2].

An essential question is how we can both effectively and efficiently contribute to the prevention of chronic diseases and how we can help people who have suffered from them or who may suffer from the min the future. There are people who need support to manage risk factors, and people who require motivation to prevent these factors in the first place, while others are truly healthy and need to stay that way. (Bustamante et al., 2012 [3]) claim that chronic disease management is highly complex because multiple interventions are required to improve clinical outcomes. From the patient's perspective, their main problems deal with self-management without support and their feeling that they are isolated between clinical visits.

A strategy for providing continuous self-management support is the use of communication technologies [3]. So-called health coaching then offers various but targeted ways to better manage one's own health and communication and IT technologies support this with software applications and wearable electronics for acquiring health-related data, such as smartphones, tablets, IoT-enabled mobile devices, smartwatches, and trackers.

This paper copes with both strategically conceptual and technical/technological aspects in building successful self-management health systems. More specifically, it deals with the architectural improvement of the already existing BodyInNumbers exercise and wellness health strategy framework (here-after called the BodyInNumbers software system or BodyInNumbers) [2] to achieve its security, scalability and maintainability.

BodyInNumbers is a modular software system primarily designed to collect data and metadata related to users' lifestyle, sports activities, physical and cognitive condition, and diet [2]. These data and metadata are then generally suitable for further analysis of lifestyle and human cognitive and physical performance. The system contains, for example, a well-annotated collection of data from electric activity of the human brain [4].

During the BodyInNumbers system operation its architecture, as originally designed and implemented, became partly obsolete. It did not follow requirements experienced during the daily use of the system (typically troubles with its security, scalability and maintainability) and did not include some of the current generally applicable design and development paradigms and techniques.

ICMHI 2020, August 14–16, 2020, Kamakura City, Japan

ACM ISBN 978-1-4503-7776-8/20/08…$15.00
https://doi.org/10.1145/3418094.3418102

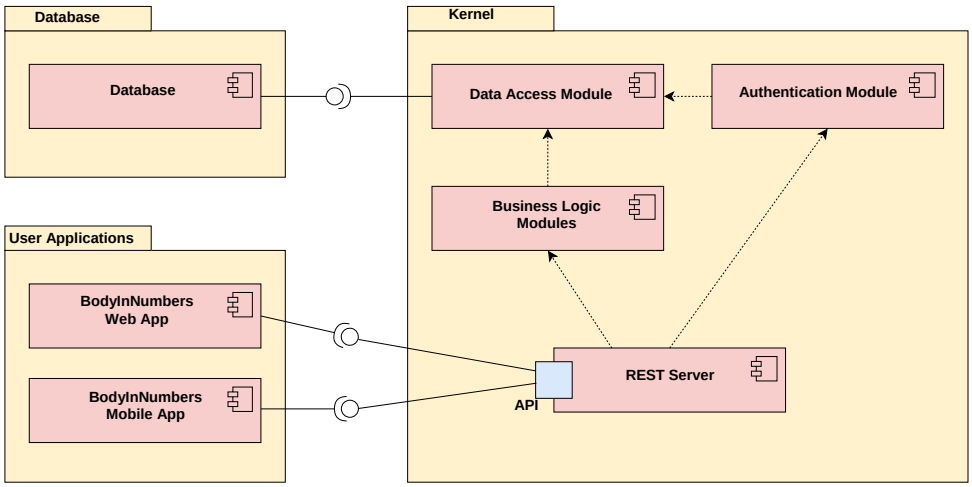

Figure 1. The component model of the BodyInNumbers system. It consists of three subsystems: kernel, database, and user applications. These subsystems consist of software components.

The advances we have achieved in building the BodyInNumbers system architecture together with the underlying architectural and technical concepts are presented in this paper.

The paper is organized as follows. First, new trends and approaches in software architectures and software engineering which are generally suitable for health related applications as well as the original architecture of the BodyInNumbers system are introduced. Then the architecture of the BodyInNumbers system is revisited and new design and technical/technological solutions are explained in more de-tail. As a proof of concept, a Kubernetes cluster prototype has been used to demonstrate the advances of the resulting solution.

2. TRENDS IN SOFTWARE ARCHITECTURES

During the last decade new approaches in software engineering and software architectures have emerged. These approaches consider to model not just documentation artifacts, but also central artifacts, al-lowing the creation or automatic execution of soft-ware systems starting from those models. These proposals have been classified generically as Model-Driven Engineering (MDE) and share common concepts and terms that need to be abstracted, discussed and understood [5].

Moreover, as development techniques, paradigms and platforms evolve far more quickly than domain applications, software modernization and migration are constant challenges to software engineers [6]. Among these, docker-based containerization techniques, container orchestration platforms, micro-service architectures, cloud computing, and artificial intelligence (AI) technologies play an essential role. Some of them have been used to improve the BodyInNumbers system architecture and are briefly introduced below.

A container is a piece of software that contains and packages everything that software needs to run. Apart from virtual machines (VMs), containers are lightweight, it makes them flexible and fast. However, it means that they are naturally designed to be short-lived and fragile. To solve this issue, instead of making each software component bullet-proof (such as VMs), the system can become more stable by assuming that each component is going to fail and designing the overall process to handle it [7]. Currently, Docker is considered to be the leading container platform.

Efficient management and scheduling of containerized applications is ensured by container orchestration platforms. These can natively fulfill performance constraints and requirements imposed by the management and orchestration (MANO) standard [8]. Kubernetes, as atypical representative of these platforms, is an open-source software for automating deployment, scaling and management of applications. It manages cluster of machines (nodes) and schedules containerized applications at them. Managed containers can be monitored and if any container seems to be faulty, it is killed and started again. Kubernetes ensures that the application is in the desired state not only at a single point in time, but continuously. The same container can also be started multiple times and incoming traffic is balanced between those containers. This ensures high availability and easier scalability of the system. Declarative configuration makes application resource management clearer and easier.

For example, it is documented that docker-based containerization techniques along with a Kubernetes container orchestration platform emphasize an efficient way to manage and monitor the status and events in IoT applications in the scale of smart cities [9].

(Thurgood and Lennon [10]) emphasize that cloud computing and artificial intelligence (AI) technologies are becoming increasingly prevalent in the industry, necessitating the requirement for advanced platforms to support their workloads through parallel and distributed architectures. Then Kubernetes provides an ideal platform for hosting various workloads, including dynamic workloads based on AI applications that support ubiquitous computing de-vices leveraging parallel and distributed architectures. The rationale is that Kubernetes can be used to support backend services running on parallel and distributed architectures, hosting ubiquitous cloud computing workloads.

3. BODYINNUMBERS ARCHITECTURE

The modular architecture of the BodyInNumbers system is based on the traditional MVC architectural pattern that enables developers to work on its modules separately. The software system consists of three main subsystems: kernel, database, and user applications. In the original architecture the kernel and user web application were more tightly interconnected. The current

architecture of these subsystems, their components and connectors is depicted in Figure 1.

The MVC architectural pattern can be considered as a common and suitable solution for such kinds of applications and this pattern was preserved and improved (splitting of the kernel and application user applications through REST API) in the current architectural solution. However, during operation of the BodyInNumbers system we encountered several in conveniences related mostly to non-functional (security, scalability and maintainability) parameters of the system. These inconveniences included a limited set of user roles that was insufficient to guarantee access only to the parts of the system the user should xhave seen, the technologically obsolete authentication process, difficulties to incorporate database model changes, troubles with scalability of the system during big events when large amounts of data were collected and overall difficulties with the system deployment. A shutdown of the system during one of the data collection events and its difficult recovery finally enforced the changes in the system architecture. These changes include the improvements of the system security (updated and more general system of user roles and improvement of the authentication process), scalability and manageability (database migration, user interface concept and technology, containerization, orchestration of the containers and overall optimization of the system deployment). Both kinds of improvements (architectural/conceptual and technical/technological) supplemented with the necessary background information are described in the following sections.

3.1 Kernel

The main task of the kernel is to provide computations over the data stored in the database. Each component of the kernel architecture defines its own controller with its key functionalities, so called API endpoints. The kernel also provides authentication, authorization and API services for web and mobile applications. To communicate with these applications, the REST API has been created. It provides data in the JSON format that can be effectively used by the web application, as well as any other client software that may be implemented in the future, like a mobile application. For the communication with the database, the framework SQLAlchemy is used; it provides an abstract database layer for object-relational mapping (ORM).

The kernel is implemented in Python (version 3.7) using the Flask micro framework (this framework is, for instance, used for handling routing). However, Flask does not provide functionalities common for other frameworks, such as a database layer or form validations; for these purposes, other approaches described later were used.

3.1.1 System of user roles

User access rights are managed by a system based on user roles. Roles possess lists of string permission keys (simple character strings identifying a user action), which signify which actions a user with such a role can take. By default, all permissions are denied to a role, and each permission is granted by giving it permission keys of individual actions.

Each role may have a parent role, from which it inherits all permissions that the parent role already has. By design, there is no way to deny an inherited permission – child roles may only extend their parent roles.

There is also a ”virtual” permission (i.e. it does not belong to any specific user action) with the *#all#* key, which grants access to all restricted endpoints. Itis typically meant for a super user role.

The ERA model in Figure 2 shows that the design and implementation of such a system of user roles is relatively simple. Currently, the system is powerful enough to allow complex

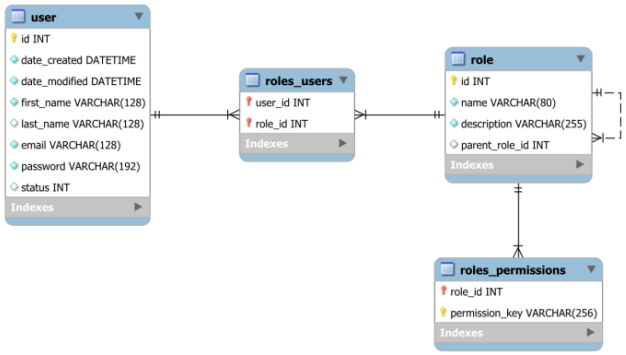

Figure 2. The part of the ERA model related to user roles and permissions.

management of access rights.

3.1.2 Authentication

Users authenticate themselves using a combination of their e-mail and password, which is a de-facto standard for basic web application authentication. When the user provides their valid credentials, the application server establishes a session, backed by a JSON Web Token (JWT)

All JWT session data is stored completely client-side. The application server holds no additional in-formation about a particular session. The authenticity of a JWT is ensured by a digital signature (using a secret key stored in the server’s configuration) of the token – that way the server knows that a session was indeed issued by it and has not been altered by the client.

3.2 Database

The system data is stored in two relational databases. The first database, *exercise-wellness*, stores local sys-tem data, whereas the second one, *exercise-wellness-test*, serves as a data storage for automated tests.

Because the schema of the database containing the system data may need to change during the development, and it is undesirable to wipe the database or adapt it manually for each deployment of the version in which such a change occurred, database migration shave been put in place. Migrations are handled by the Alembic library, which uses a hierarchy of upgrade scripts to serve its purpose. The database contains in-formation about the last upgrade that has been applied to it. That way, every time a new change to the schema is required, the server administrator only needs to run a single command to apply the required changes.

3.3 Web Application

The client web application (as one of the designed and implemented user applications) displays values of measured health parameters as well as results of derived statistics. To make these data transparent for end users the open-source Recharts library containing chart components is used.

The web application is written in JavaScript using React, version 16.9.0. For requesting the kernel API endpoints, the *axios* HTTP client is used.

Since version 16.8.0, React includes a stable implementation of React Hooks giving local functional components the ability to use a state. The web application represents data instances provided by

the Kernel API as state variables and uses Hooks to schedule actions after rendering the page. Styling of the web application is handled with the *styled-components* library and CSS.

4. BODYINNUMBERS DEPLOYMENT

The BodyInNumbers system runs in two (production and development) instances. It is deployed using Docker containers in a Kubernetes cluster.

4.1 Cluster

The BodyInNumbers system is currently split into two containers: the Flask application and the PostgreSQL database. The system runs in the Kubernetes cluster that consists of three nodes, each with the following specification: 15GB HDD, 2GB RAM, 1×CPU. To ensure that the containers are safely managed by Kubernetes, several resource types have to be defined.

ConfigMap and Secret These resources describe the database connection configuration (e.g.name and password). These values are passed to the PostgreSQL and Flask containers as environment variables.

Deployment The deployment resource defines the Flask webserver container. The container image is defined here, alongside with the number of replicas of the container. Kubernetes ensures that the correct number of replicas are running (if sufficient resources are available). Environment variables needed for the database connection are passed from the **ConfigMap** and **Secret** resources.

Stateful set The stateful set resource is used to define the PostgreSQL database container. As in the case of the deployment resource the container image, number of replicas and environment variables are declared here. Besides deployment, a volume is defined in this resource to persist data when the container is destroyed. The relational database is a potential bottleneck to system scalability. It could be replaced by a NoSQL database in the future.

Service Containers can be created and destroyed dynamically and the **Service** resource is the mechanism to assign a static unique IP address to them in the cluster network. The definition of a service is a set of selector rules, which target Pods (containers). Any incoming traffic to service IP is than forwarded to the random container defined by the service. The application has two defined services – Flask and Postgres. The Postgres service is reachable only from within the cluster. The Flask service is exposed to the Internet by the external server outside the cluster. This server also loads balances traffic between Kubernetes nodes.

4.2 Deployment Process

To deploy a new version of the system on the cluster, a new Flask application image has to be built and pushed to the container registry. Once the image is pushed, Kubernetes deployment can be updated to use this new image. Kubernetes deployments have several updating strategies to keep application availability high even during updates

The deployment is made using the GitLab. CI/CD configuration is made using three steps: Build, Test and Deploy. During the first step, the Docker image is built and pushed to the GitLab Container Registry. In the second step, tests are run from the image that has been built. In the future, different kinds of tests can be started concurrently. If the Build and Test stages are finished successfully, the manual step Deploy can be started. This updates Kubernetes deployment to use the new image from the Container Registry.

5. TESTING

The kernel logic has been tested with a set of unit tests using the standard *unittest* framework included in Python 3.

The whole BodyInNumbers health strategy framework has been tested at two events during which larger heterogeneous data collections were collected. No issues related to the system security, scalability or maintainability have been reported.

6. CONCLUSION

The further progress has been made in the BodyInNumbers system architecture and its implementation. The architecture has been adapted to be more secure, scalable and maintainable. The user application and kernel layers have been split, so any changes in the user parts of the system do not require full access to the system code. Thus, any change on the user inter-face does not require additional security reviews.

A new system of user roles has been implemented. It allows more users (user roles) to have access to the system and ensures that they will be allowed to access only the parts of the system they should see.

The authentication process was reviewed and technologically upgraded as well as easy to perform database migrations were technically/technologically enabled.

A new deployment has been prototyped using Docker containers, Kubernetes and GitLab, which offer advanced automation tools. Thanks to Kubernetes, the application configuration can be organized and simplified. It allows better availability, scalability and maintainability. The deployment process itself has been automated and simplified.

There are still some opportunities to increase scalability in the future, such as migrating persistent storage to a distributed database.

7. ACKNOWLEDGMENTS

This publication was supported by the UWB grant SGS-2018-019 Data and Software Engineering for Advanced Applications.

8. REFERENCES

[1] Brůha, P., Mouček, R.,Šnejdar, P., Bohmann, D., Kraft, V., and Řehoř, P. 2017. Exercise and wellness health strategy framework. BIOSTEC 2017, pages 477–483.

[2] Brůha, P., Mouček, R.,Šnejdar, P., Vařeka, L., Kraft, V., and Řehoř, P. 2018. Advances in building bodyinnumbers exercise and wellness health strategy framework. BIOSTEC 2018, pages 548–554.

[3] Bustamante, C., Alcayaga, C., Lange, I., and Inigo, M. 2012. SIGSAC Software: A tool for the Management of Chronic Disease and Telecare. Nursing informatics 2012: proceedings of the 11th International Congress on Nursing Informatics, 2012:56.

[4] Brůha, P., Mouček, R., Vacek, V.,Šnejdar, P.,Černá, K., and Řehoř, P. 2018. Collection of human reaction times and supporting health related data for analysis of cognitive and physical performance. Data in Brief, 17.

[5] Silva, A. 2015. Model-driven engineering: A survey supported by A unified conceptual model. Computer Languages, Systems & Structures, 20.

[6] Fleurey, F., Breton, E., Baudry, B., Nicolas, A., and Jézéquel, J.-M. 2007. Model-Driven Engineering for Software

Migration in a Large Industrial Context. InMoDELS'07, volume 4735.

[7] Rensin, D. K. 2015. Kubernetes. page 3.

[8] Gawel, M. and Zielinski, K. 2019. Analysis and evaluation of kubernetes based nfv management and orchestration. In 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), pages 511–513.

[9] Muralidharan, S., Song, G., and Ko, H. 2019. Monitoring and managing iot applications in smart cities using kubernetes. CLOUD COMPUTING 2019, page 11.

[10] Thurgood, B. and Lennon, R. 2019. Cloud Computing With Kubernetes Cluster Elastic Scaling. pages 1–7.