COMENIUS UNIVERSITY OF BRATISLAVA

FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

APPLICATION DESIGNED TO HELP EMPLOYEES PREPARE FOR VARIOUS CERTIFICATIONS

Bachelor thesis

2022

Pavel Semenov

COMENIUS UNIVERSITY OF BRATISLAVA

FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

APPLICATION DESIGNED TO HELP EMPLOYEES PREPARE FOR VARIOUS CERTIFICATIONS

Bachelor thesis

Study programme: Applied Computer Science

Study field: 9.2.9 Applied Informatics

Department: Department of Applied Informatics

Supervisor: RNDr. Marek Nagy PhD.

Consultant: Ing. Pavol Jesenský

2022

Pavel Semenov

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Pavel Semenov
Study programme:	Applied Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak
Title:	Application designed to help employees prepare for various certifications
Annotation:	Resulting product of the project is ready-to-use, fully functional app. App allows employees to choose the certification and check their knowledge on the topic. These data are provided by back-end service and requested by the app through its API. This project will be of a huge help for employees as no such tools are yet available for them. Project aims to make preparation for the certifications easy and effective at the same time. The goal of the application is upskilling of the employees primarily in the cloud computing area e.g. AWS certifications, Openshift certifications, etc. The Backend will provide data at REST API, the consumer in scope is mobile app which consumes the API.
Comment:	Designed for IBM company needs. Contact person and consultant: Pavol Jesenský (pjesensk@sk.ibm.com)

Supervisor:	RNDr. Marek Nagy, PhD.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.

Assigned: 22.10.2021

Approved: 25.10.2021 doc. RNDr. Damas Gruska, PhD.
Guarantor of Study Programme

.....

Student Supervisor

DECLARATION

I declare that I have prepared the bachelor's thesis independently and I have listed every used literature.

……

student’s handwritten signature

Acknowledgement

I would like to thank my supervisor RNDr. Marek Nagy, PhD. and consultant Ing. Pavol Jesenský for help and advice.

Abstract

IT certifications are becoming more and more valuable nowadays. While they are considered a good benefit for juniors, they are now absolutely essential for everyone applying for a senior position. IBM Slovakia discovered a need for internal certifications preparation system to be able to offer the most well-trained specialists to the clients.

This thesis considers the development of multiplatform mobile application providing tests designed to help IBMers prepare for the certifications. Main features of the app include a list of the available learning resources, different kinds of preparatory tests, an access to the remote exercise environment from the mobile device. We examine closely the technologies, design and implementation of the app.

Keywords: mobile app, certifications, Spring Boot, React Native, Kubernetes.

Abstrakt

IT certifikácie sú v dnešnej dobe čoraz cennejšie. Zatiaľ čo sú považované za dobrý prínos pre juniorov, v súčasnosti sú absolútne nevyhnutné pre každého, kto sa uchádza o seniorskú pozíciu. IBM Slovensko objavilo potrebu interného systému prípravy ku certifikáciam, aby mohla klientom ponúknuť čo najkvalitnejších špecialistov.

Táto práca sa zaoberá vývojom multiplatformovej mobilnej aplikácie poskytujúcej testy navrhnuté tak, aby pomohli IBM zamestnancom pripraviť sa na certifikácie. Medzi hlavné funkcie aplikácie patrí zoznam dostupných učebných zdrojov, rôzne druhy prípravných testov, prístup k vzdialenému cvičebnému prostrediu z mobilného zariadenia. Táto práca uvažuje technológie, dizajn a implementáciu aplikácie.

Keywords: mobilná aplikácia, certifikácie, Spring Boot, React Native, Kubernetes.

Table of contents

Introduction

According to the survey conducted by me among the IBM Slovakia employees, 100% of respondents (24 employees) are either doing a certification or preparing for one. Third of them spends more than 8 hours a week preparing for the certification. A lot of the time is spent searching for the information and the practice tests instead of actual productive learning. Thus, it seems to be convenient if practice tests and most useful information resources for a large set of certifications were available at hand via the mobile app, and, according to the survey, 92% of IBM SK employees agree to this statement. After research of the market no apps satisfying all the requirements at once were found. This created an idea for such an app, that would save the time for IBM employees, allowing them to prepare for the certifications more efficiently.

Technically the app is based on a client-server architecture, where the Spring Boot framework is used for the backend part and the frontend is developed using React Native. Most of the data is stored in NoSQL MongoDB database on the server side and is accessed by the frontend through the API interface by communicating via the HTTP protocol. Backend, in its turn, executes the necessary commands against the Kubernetes cluster to setup or clean out the exercise environment if needed.

The following chapters will be focusing on the main aspects of the app. They will provide a deeper research of the problem, present the technologies and the technical design. First chapter shortly introduces the world of certifications, gives an overview of the existing solutions to the problem and presents the technologies used for the implementation. Features of the app, its architecture and design are described in the second chapter. Lastly, the third chapter focuses on the implementation of the app and its final looks. It points out the most challenging aspects of the implementation process and comments on the most interesting pieces of source code.

1 Problem statement

This chapter gives a complete image of the problem along with the existing solutions. First, a brief introduction into the certifications is given. Then, a closer look at the results of the survey with analysis is provided. Finally, there are a few chapters presenting the results of the market research of existing solutions and a list of theses covering similar problems.

1.1 Certifications

Certifications are a good way to strengthen one's CV. Awarded by professional organizations, they confirm a person's ability to do a specific job or his possession of specific knowledge. Certifications are acquired through passing an exam and usually valid for a limited amount of time.

Most popular IT certifications include ones from Amazon, Microsoft, RedHat and Oracle. For example, Amazon offers an "AWS Certified Cloud Practitioner" certification. As the company states "Earning AWS Certified Cloud Practitioner validates cloud fluency and foundational AWS knowledge." [1]

Usually a certification represents a series of multiple-choice tests, which are supplemented with multimedia elements in some cases. However, so-called "performance-based testing" has been steadily growing in popularity throughout past years. In performance-based testing, a simulated environment is presented to the exam taker, who is then required to complete a task, rather than answer questions. Performance-based testing is favored, because it replicates the work environment and emphasizes results. Employers see simulations as a far superior way to measure whether someone has the real-world skills required to troubleshoot a network, for example, or configure a workstation.

After the exam is completed, the overall score is calculated. If the score is greater or equal to the passing score, the certification is earned by the exam taker. It can be then verified online by the unique certificate id. Some exams also provide detailed information on the performance. After completing any of the certification offered by the Red Hat, for instance, exam taker receives an email with an overall score and a performance evaluation for each of the exam objectives.

1.2 Survey

A survey was conducted among the employees of IBM Slovakia. It's results confirm that certifications are a vital step towards career progress. All participants of the survey are doing the certifications or at least planning to do so soon.

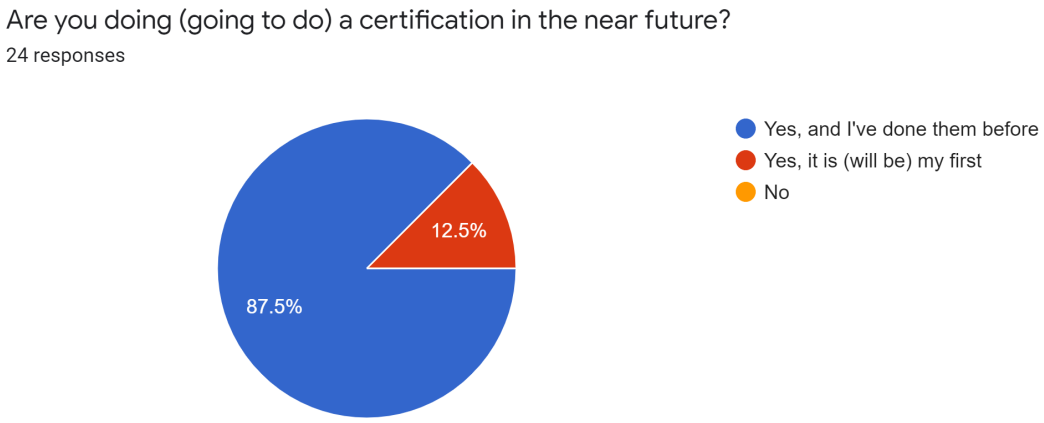

Figure 1. Pie chart clearly shows, that the majority of employees have already earned a certification, while other 12.5% are preparing for their first.

Most of the respondents also stated that it would be helpful to have an app which would provide the necessary preparation materials. This is probably caused by the fact that the currently available information related to the specific certification is scattered across the Internet, making it rather time-consuming to look for it.

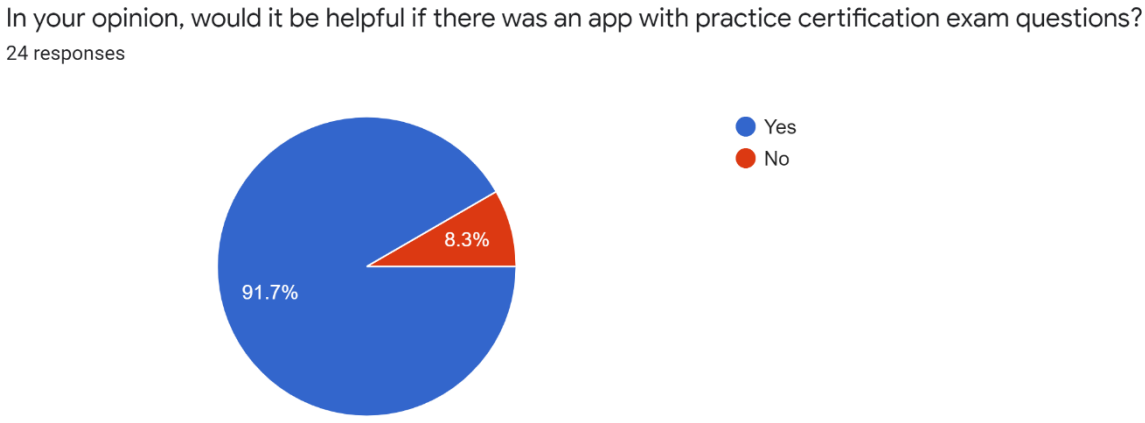

Figure 2. According to the pie chart presented above, most of the employees agree, that it would be helpful to have an app allowing to practice before the certification exam.

The survey also included a few questions regarding the information resources IBM employees use for the preparation. Respondents had to evaluate the usefulness and the ease of use (by marking it from 1 to 10) of the following resources: Slack[1] channels, notes from a colleague, official courses and web resources. They were asked to provide some concrete examples, as well. The most popular were web resources, which were used by approximately 87.5% of respondents. However, the easiest ones to use were official courses with an average mark of 7.7 and notes from a colleague with an average mark of 7.2, while web resources and official courses were the most useful with average marks of 8.1 and 7.8 respectively. Therefore, the resources tab of the app should certainly include links to the web resources like Udemy, which is the most popular of them, and official preparation courses.

1.3 Existing solutions

There are a few apps preparing for the certifications available on the market. However, no such theses were found, that would consider the same problem this thesis is considering. Thus, we will have a closer look at a few theses considering student testing apps or good examples of server-client architecture implementations.

1.3.1 Certification Questions

This app [2] provides training tests for a big variety of IT certifications. It also tries to reproduce the exam environment with a timer ticking in the top left corner, although it also provides an explanation for each question, which might be quite useful. However, the list of the available certifications doesn't include some of the most popular ones, like "EX-288". Moreover, this application does not provide "performance-based tests". It is also lacking the automated answer evaluation for each question, which deprives user of the actual exam experience. The

[1] Slack is a business communication platform widely used among IT companies. [13]

explanation is rather hard to read through and the “Previous” and “Next” buttons on the bottom are too small.

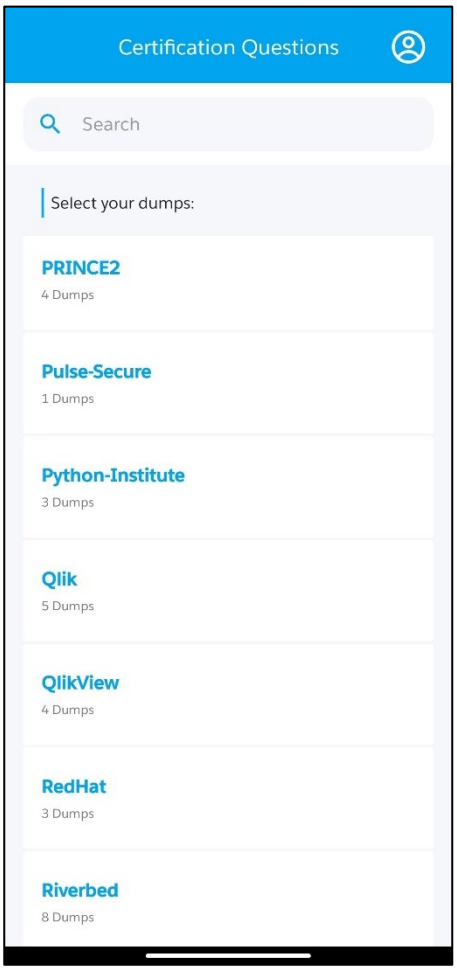

Figure 3. A list of the certification vendors.

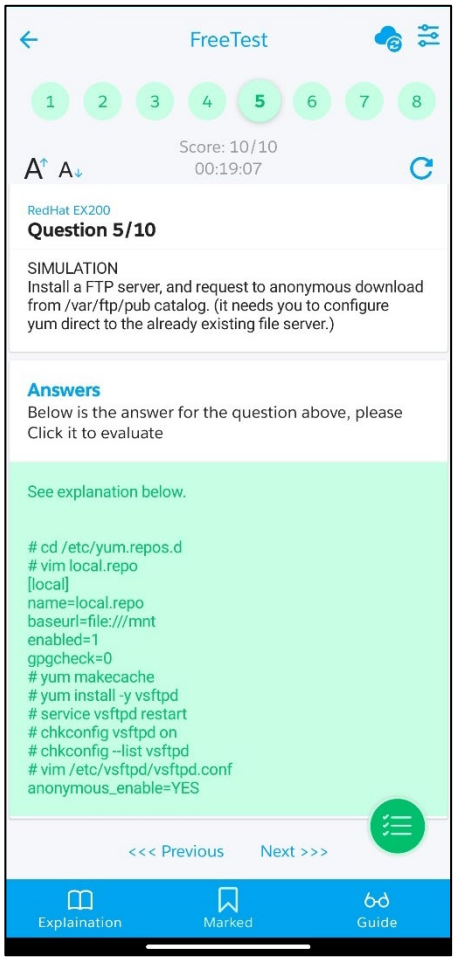

Figure 4. One of the tasks for the “EX200” certification.

To sum up, this app impacts my implementation in a few ways. Firstly, the size of the text and buttons should be considered as this has a huge impact on the overall app usability. Secondly, tab navigation looks native on mobile apps and, thus, should be the main navigation component in my implementation. Thirdly, a search bar would be a great addition to the list of the certifications and to the list of available learning sources. It allows to find the necessary resources fast, making it easier to navigate in the app and increasing overall usability of the app.

1.3.2 PMP Certification Mastery

This app [3] is available on iOS devices and provides a great preparation environment for the Project Management Professional (PMP)® certification. It offers exam-like practice questions, PMP concepts explanation, quiz questions and instructional videos. A whole tab is designated to the performance tracking, encouraging users to stay on track and maintain practice streaks. On the downside, there are only multiple-choice questions in every exercise and the paid subscription is required to finish the exercise and get the performance evaluation.

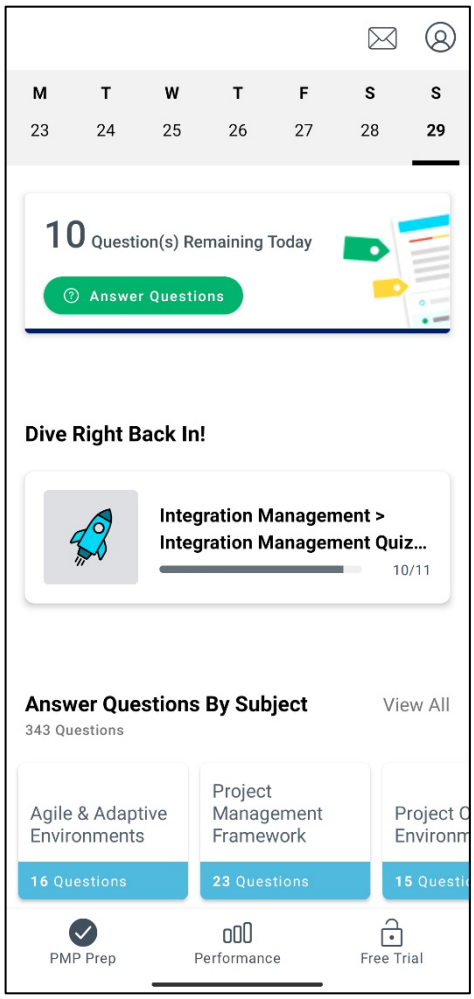

Figure 5. Main menu of the app.

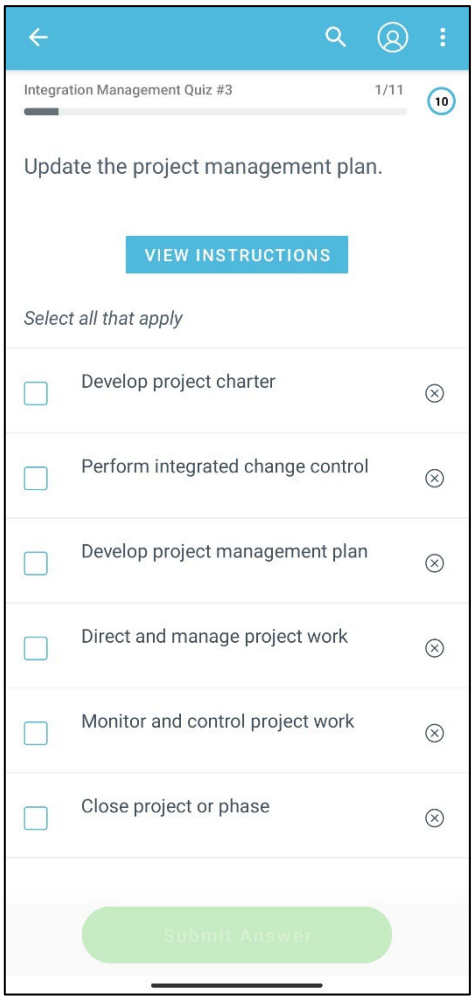

Figure 6. Test page with the multiple-choice question.

All things considered, material design is probably the only thing worth to borrow for my implementation of the app.

1.4 Technologies

This section describes the technological solutions used by the app. They are divided into subsections: backend and frontend. The former lists the tools required for the server side of the app; the latter lists the tools required by the client side.

1.4.1 Backend

Backend of the solution is developed using mainly Java programming language. A few static bash scripts are stored on the server and called by the application.

1.4.1.1 Java

Java is a high-level, class-based, object-oriented programming language. It is widely used in the software development, allowing to build robust and portable products. Compiled Java code can run on all platforms that support Java without the need to recompile. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture.

Another strong point towards choosing Java as the main backend language is the Spring framework. Spring provides everything required beyond the Java programming language for creating enterprise applications for a wide range of scenarios and architectures. More details about the framework can be found in further chapters dedicated to the subject.

Since nowadays official releases of Oracle Java require a paid subscription for commercial support, IBM is using free open-source reference implementation of Java called OpenJDK. For the development was chosen Java 11 as it is the most stable and widely supported release after the Java 8, yet it also has some very useful features like *var* keyword and additional String methods (*strip(), lines(), isBlank()*).

1.4.1.2 Spring

Spring [4] is the most popular application development framework for enterprise Java. Lots of developers use Spring Framework to create high performing, easily testable, and reusable code.

Spring framework is an open source Java platform. It is lightweight when it comes to size and transparency. The basic version of Spring framework is only around 2MB in size.

Spring framework is modular, which allows developers to pick only the modules applicable to their product. The following diagram lists some of the modules provided by the framework. The ones, that are used in my implementation are colored in green.

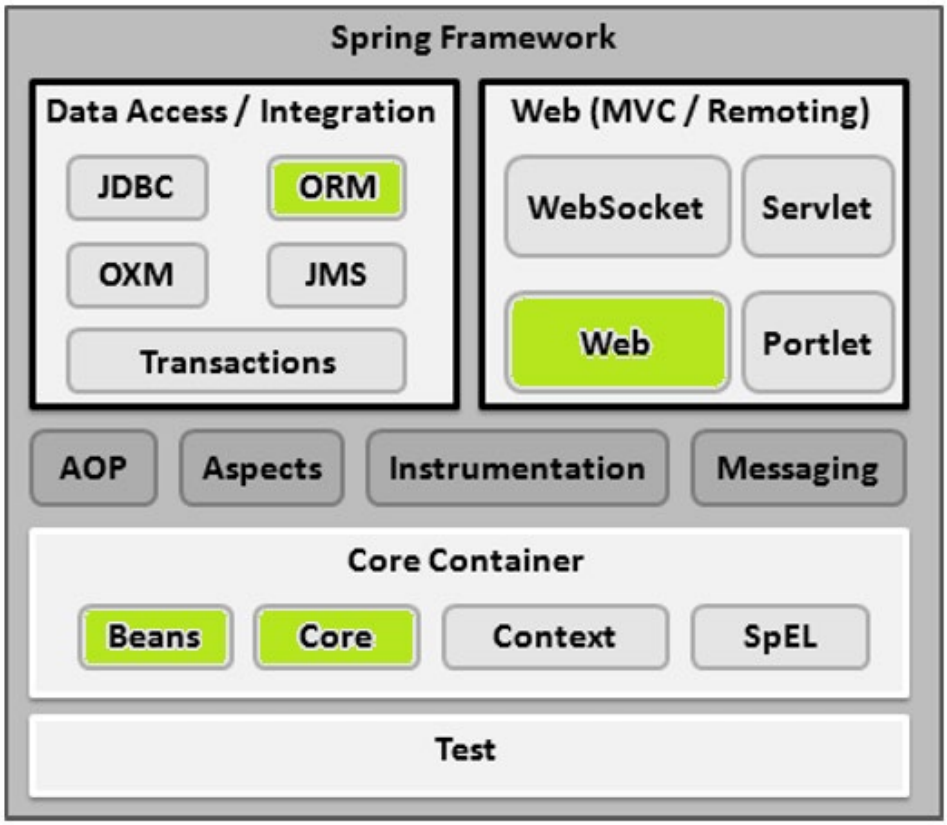

Figure 7. Modules of the Spring framework.

The Core Container consists of the Core, Beans, Context and Expression modules. The Core and Beans modules provide the most fundamental parts of the framework and provide the IoC and Dependency Injection features. IoC or Inversion of Control is a programming principle, where the flow of the program is controlled by the external sources. Dependency Injection, often referred to as DI, is a more specific version of IoC, where implementations are passed into an object through either constructors or setters or service lookups, which the object will "depend" on in order to behave correctly. The basic concept here is the *BeanFactory* which provides a sophisticated implementation of the factory pattern. It removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.

The ORM module provides integration layers for popular object-relational mapping APIs, including JPA, JDO and others. My implementation makes use of the JPA mapping API. I describe how in further chapter dedicated to the subject.

The Web module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context. Spring is also shipped with an embedded Tomcat server.

1.4.1.3 Spring Boot

Spring Boot [5] is a tool that helps developers to create stand-alone, production-grade Spring-based applications that you can run. It takes an opinionated view of the Spring platform and third-party libraries, so that very little Spring configuration is needed to get started.

Among the features of Spring Boot are:

- Radically faster and widely accessible getting-started experience for all Spring development.
- Possibility to be opinionated out of the box but get out of the way quickly as requirements start to diverge from the defaults.
- A range of non-functional features that are common to large classes of projects (such as embedded servers, security, metrics, health checks, and externalized configuration).
- Absolutely no code generation and no requirement for XML configuration.

Spring Boot applications are annotation-based. This means, that there are lots of annotations used in every class and interface definition. We'll consider most common ones in implementation chapter.

1.4.1.4 Lombok

From the official description [6] "Project Lombok is a java library that automatically plugs into your editor and build tools, spicing up your java." Lombok basically provides a lot of useful annotations to make Java code clean and compact. Among most popular features of Lombok are annotations, that can handle generation of getters, setters and constructors automatically.

1.4.1.5 Swagger-UI

Swagger UI allows development team to visualize and interact with the API's resources without having any of the implementation logic in place. In other words Swagger allows to generate an API documentation without actually implementing it.

There is an *io.springfox.documentation* module available for the Spring, which allows to generate a documentation webpage easily by declaring an additional Bean. This webpage provides a list of controllers with the endpoints specifications. Each endpoint has an example response model, available request parameters and an option to execute the request on the spot. Besides the controllers, there is also a list of the models.

1.4.1.6 Spring Data for MongoDB

The Spring Data MongoDB project provides integration with the MongoDB document database. Key functional areas of Spring Data MongoDB are a POJO centric model for interacting with a MongoDB *DBCollection* and easily writing a Repository style data access layer. Features of the Spring Data MongoDB include:

- *MongoTemplate* helper class that increases productivity performing common Mongo operations. Includes integrated object mapping between documents and POJOs.
- Annotation based mapping metadata but extensible to support other metadata formats
- Java based *Query*, *Criteria*, and *Update* DSLs
- Automatic implementation of Repository interfaces including support for custom query methods.
- *GridFS* support

1.4.1.7 MongoDB

MongoDB [7] is a document database designed for ease of development and scaling. It is a NoSQL database, which MongoDB stores data records as BSON documents. BSON is a binary representation of JSON documents, though it contains more data types than JSON.

Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. Documents doesn’t need to have a schema defined beforehand. Instead, the fields can be created on the fly.

The document structure is more in line with how developers construct their classes and objects in their respective programming languages. Usually classes and objects can easily be represented as key-value pairs.

The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.

Another upside of MongoDB is its scalability – the MongoDB environments are very scalable, allowing to define clusters with hundreds of nodes.

1.4.1.8 Maven

Maven [8] is a build automation tool used primarily for Java projects. Maven builds a project using its project object model (POM) and a set of plugins. Maven provides useful project information that is in part taken from your POM and in part generated from your project's sources. For example, Maven can provide:

- Change log created directly from source control
- Mailing lists managed by the project
- Dependencies used by the project
- Unit test reports including coverage

1.4.1.9 yq

yq is a lightweight and portable command-line YAML, JSON and XML processor. It can be used to modify and reassign YAML properties, merge or create YAML files.

1.4.1.10 kubectl

kubectl is a command line tool for communicating with a Kubernetes cluster's control plane, using the Kubernetes API. It looks for configuration file in *$HOME/.kube* directory or uses the location specified in the *KUBECONFIG* environment variable. A default namespace, authentication provider and cluster IP address are among the properties that can be defined using the configuration file.

1.4.1.11 Kubernetes

Kubernetes is an open-source container orchestration tool. Basically, Kubernetes allows to define the way applications are deployed using YAML files. Kubernetes then distributes the load between the available hardware resources (nodes) inside the cluster (a set of nodes). This feature is called load-balancing. Besides, load-balancing Kubernetes can manage the scaling and exposing of your deployments.

To partition a single Kubernetes cluster into multiple virtual clusters we can use namespaces. Namespaces provide a mechanism for isolating groups of resources within a single cluster. There are a lot of types of resources, that can be deployed into the Kubernetes cluster. Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. Pod can run several containers at once. It can also have a volume, which can be mounted to a specific directory inside the container. One way to populate that volume is by ConfigMap. A ConfigMap is an API object used to store data in key-value pairs. When mounted, each key becomes the filename and its value makes the contents of the file inside the mounted directory.

1.4.1.12 Docker

Docker provides a set of tools to build application images. Docker images are basically a template with the set of instructions to build a container, which is in some ways similar to VM (Virtual Machine). The *Dockerfile* makes the specifications for creating an image. Since Docker uses a layering system each *Dockerfile* must specify the image to build upon using the "FROM" keyword. Docker Engine is designed to build images from *Dockerfiles* and run them.

1.4.2 Frontend

Frontend is developed using React Native framework for the JavaScript programming language. Expo is also used as it provides a great set of tools for building and testing the application.

1.4.2.1 JavaScript

JavaScript is a high-level, often just-in-time compiled language. It has dynamic typing, prototype-based object-orientation, and first-class functions. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

1.4.2.2 React Native

React Native [9] is an open-source UI software framework created by Meta Platforms, Inc. It is used to develop applications for a variety of platforms by enabling developers to use the React framework along with native platform capabilities.

React Native provides a core set of platform agnostic native components like View, Text, and Image that map directly to the platform's native UI building blocks. Besides that, React Native provides animations, navigation and routing libraries. Below are listed some libraries used in my React Native project.

1.4.2.3 @react-navigation

This library allows to construct navigation containers for the app. Alongside bottom tabs and stack navigation it provides a useful *useRoute()* hook to access route in every functional component. We examine the use of this hook in the implementation chapter.

1.4.2.4 react-native-paper

Paper is a collection of customizable and production-ready components for React Native, following Google's Material Design guidelines. It allows to create convenient and user-friendly design.

1.4.2.5 @react-native-async-storage

This module [10] provides an asynchronous, unencrypted, persistent, key-value storage system for React Native. It allows to store and retrieve data on different platforms using the same API. For each platform, though, it implements its own type of storage based on the features a specific device possesses.

1.4.2.6 Expo

Expo is a framework based on the React Native. It extends available functionality with some great add-ons like *DocumentPicker*, *GoogleSignIn* or *SMS*. It is shipped with the Metro to bundle JavaScript for Android and iOS platforms.

2 Design

This chapter aims to give a full understanding of the technical aspects regarding the app implementation. First, the design of the solution will be considered in detail. Then, we will cover the DevOps side of the app.

2.1 Features

Test environments providing different types of exercises and different kinds of tasks are the main feature of the app. However, besides the “Test” tab where the tests are located, “Drillz” app also has a “Learn” tab with a list of links to the various courses and an “About” tab, where information about the app itself is located.

There are three exercise types available in the app: “test”, “lab” and “field”. The first one looks like the most common exams. User can encounter either single-choice, multiple-choice or text questions. Exercises of the second type represent the so-called “performance-based testing”, where the test environment is designed to test practical skills of the test taker. Tasks are arranged in a certain order and target one particular real-world problem. User is expected to construct a command from the offered command parts. The third type is similar to the second one except this time the commands are actually being executed against the Kubernetes cluster. Each exercise has a recommended duration predefined, and the amount of time spent for the user to complete the exercise impacts the final score for this exercise.

The maximum (although hardly ever reachable) score for the exercise is 10000. Half the score is awarded based on the correctness of exercise, another half is awarded based on the time spent. The amount of points awarded for each task depends on its weight. No points are awarded for the task if at least one mistake was made. Time points are awarded according to the following formula:

$$5000*\max(time\_expected - time\_actual, 0)/time\_expected$$

which means that zero points are received if time spent on the exercise equals or exceeds the time allocated on the exercise and a maximum of 5000 is received when no time is spent on the exercise.

Progress for each exercise is saved to the local storage. That is, if the exercise is left uncompleted, user has an option to continue practicing from the first uncompleted task. The timer and number of failures are also loaded from the local storage.

After the exercise is finished, a short summary is displayed. For each exercise there is the total completion time, the total mistake count and the overall score is displayed. An option to restart the exercise is also available.

2.2 App Architecture

App is constructed of the frontend, which makes HTTP requests to the backend, which, in its turn, communicates with the database and Kubernetes cluster. The following diagram gives a basic overview of the app structure.

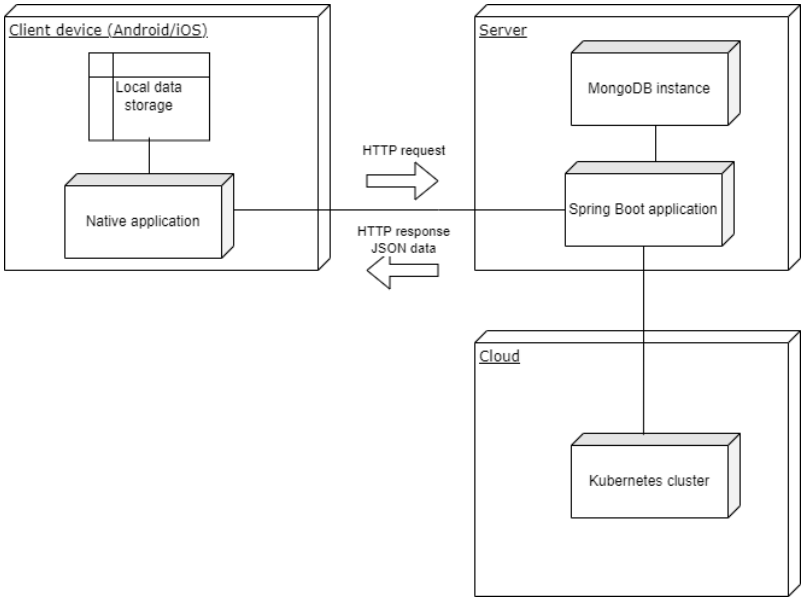

Figure 8. App architecture diagram.

2.3 Server

Backend of the solution is built upon Spring Boot. Its main purpose is to provide API, which responds to the HTTP requests from the frontend, sending the requested data in JSON format. Backend is also responsible for the creation and deletion of exercise environment when the task of type “field” is started or ended.

2.3.1 Data model

The following diagram illustrates the data model for the app.

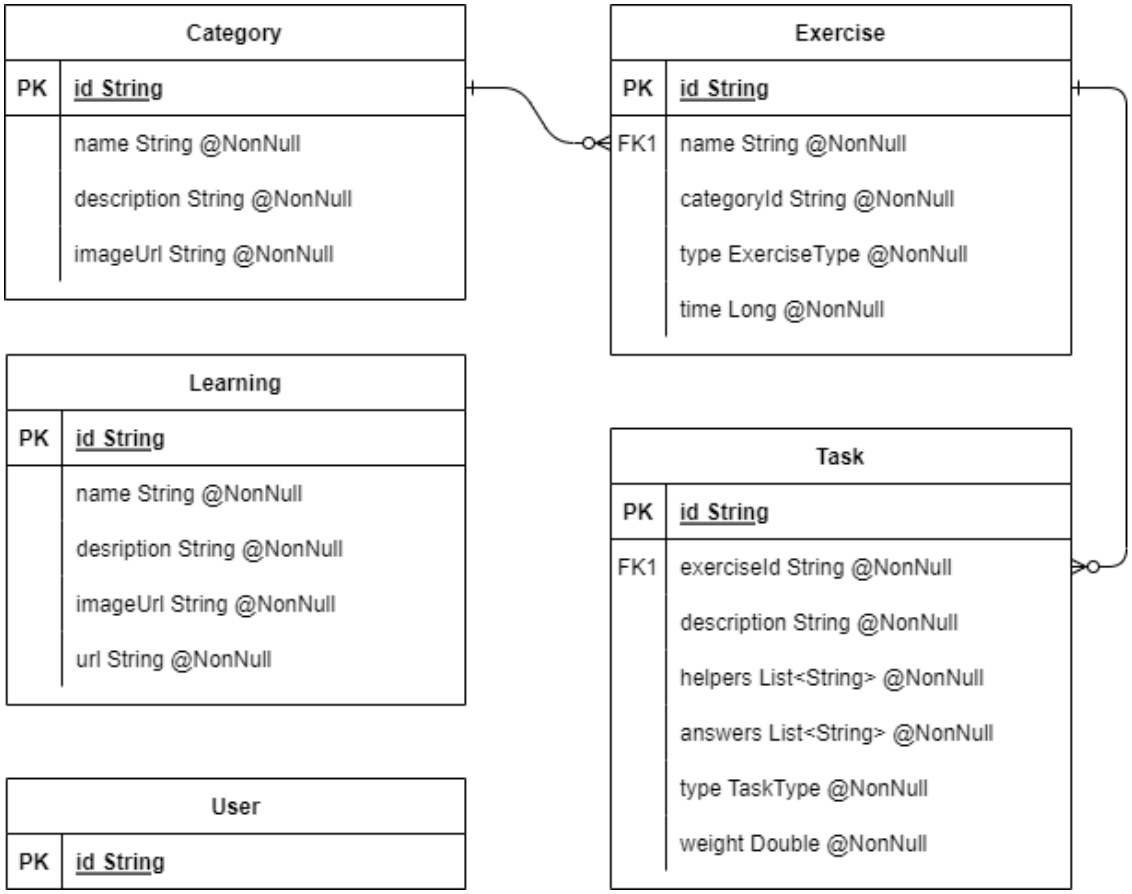

Figure 9. Data model diagram.

The app does not require a complicated data model. For *User* entity, for instance, only the id is stored as a way to uniquely identify users. *ExerciseType* and *TaskType* are two static Java enums.

2.3.2 Spring Boot application

Spring Boot application listens for the incoming requests and responds to them using the data fetched from the database. It provides endpoints for each model. Spring Boot app also ensures, that the endpoints are protected with Basic Authentication. It is responsible for all database transactions.

The application is designed to have main packages: models, repositories and controllers. The diagram below displays the package structure and dependencies between them. We consider each package in detail below.

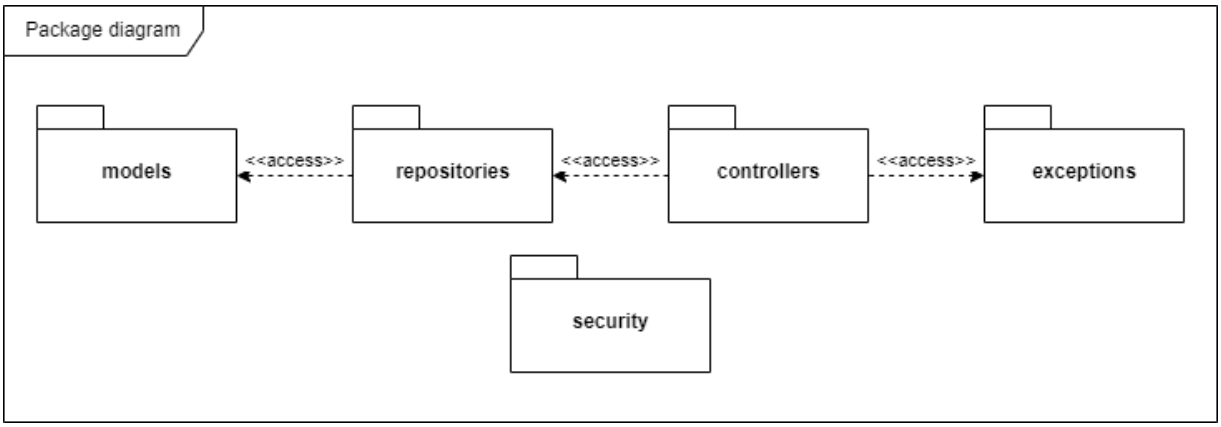

Figure 10. UML package diagram. Security package does not import or access any functionality from other internal packages.

Models package includes all the declarations of all database models used in application. Each model has an Id field, which is automatically generated when a new object is inserted into the database. Required fields are marked as *NonNull* to ensure their presence in every entry of the respective model. Besides models, the package includes the definitions of Java enums to distinguish between the types of exercises and tasks.

Repositories package contains repositories, which are the services responsible for the communication between the Mongo database and controllers. There is a separate repository interface for each one of the models.

Controllers package is a collection of REST controllers. Each controller has the definition of endpoints for a particular database model. Controllers are responsible for serializing the requests and building the responses.

Another important part of the app is security package. Security package handles the authentication entry point, security and CORS (Cross-Origin Resource Sharing) configurations. All requests should pass through the authentication entry point, which decides if the request satisfies the security requirements. If not, it sends the 401 UNAUTHORIZED response.

2.3.3 MongoDB

Mongo database does not require any further configuration except creating a separate user with a password for the backend to use. Once the MongoDB Server is up, Spring Boot manages the connection itself. It is only required to provide the *spring.data.mongodb.url* property in the application properties. Connection string has the following format:

mongodb://[username:password@]host[:port][/[defaultauthdb][?options]]

where *defaultauthdb* is the default database name to connect to and *option*s is query string that specifies connection specific options as key-value pairs.

2.3.4 Field exercises

A dedicated chapter is required for the exercises of the "field" type as they have the most complicated design and involve executing bash scripts and managing the Kubernetes cluster.

2.3.4.1 Kubernetes cluster and exercise environment

Each exercise environment can be uniquely identified by user id and exercise id. This allows multiple users to work on the same exercise at the same time and yet have a full environment for themselves. This also allows the same user to start the next exercise as soon as he finishes the previous one as some time needed to terminate the exercise environment. Three resources are created for each user-exercise pair. A single convention is used to name resources inside the cluster. That is, for user with id *1234* and exercise with id *4321* a namespace is named *ns-1234-4321*, a pod is named *pod-1234-4321* and a config map containing the Kubernetes configuration is named *kubeconfig-1234-4321*.

Kubernetes cluster has a namespace, let's say *user-vms*, where a pod and a config map are created when new "field" exercise is started by user. All commands issued by the user are then executed inside this pod. This means that user has no access to the server environment itself. The config map has a Kubernetes configuration file, which restricts the default namespace used when *kubectl* commands are executed inside the pod to the one, which was created specifically for this exercise. *user-vms* namespace has also a static config map called *credentials*, which is the same for every user and every exercise. It contains a JSON file with credentials to the service account, which is used to authenticate user's commands against the cluster. The diagram below

represents the state of the Kubernetes cluster, when a user from example above starts the exercise. Unrelated namespaces and resources are omitted in sake of clarity.

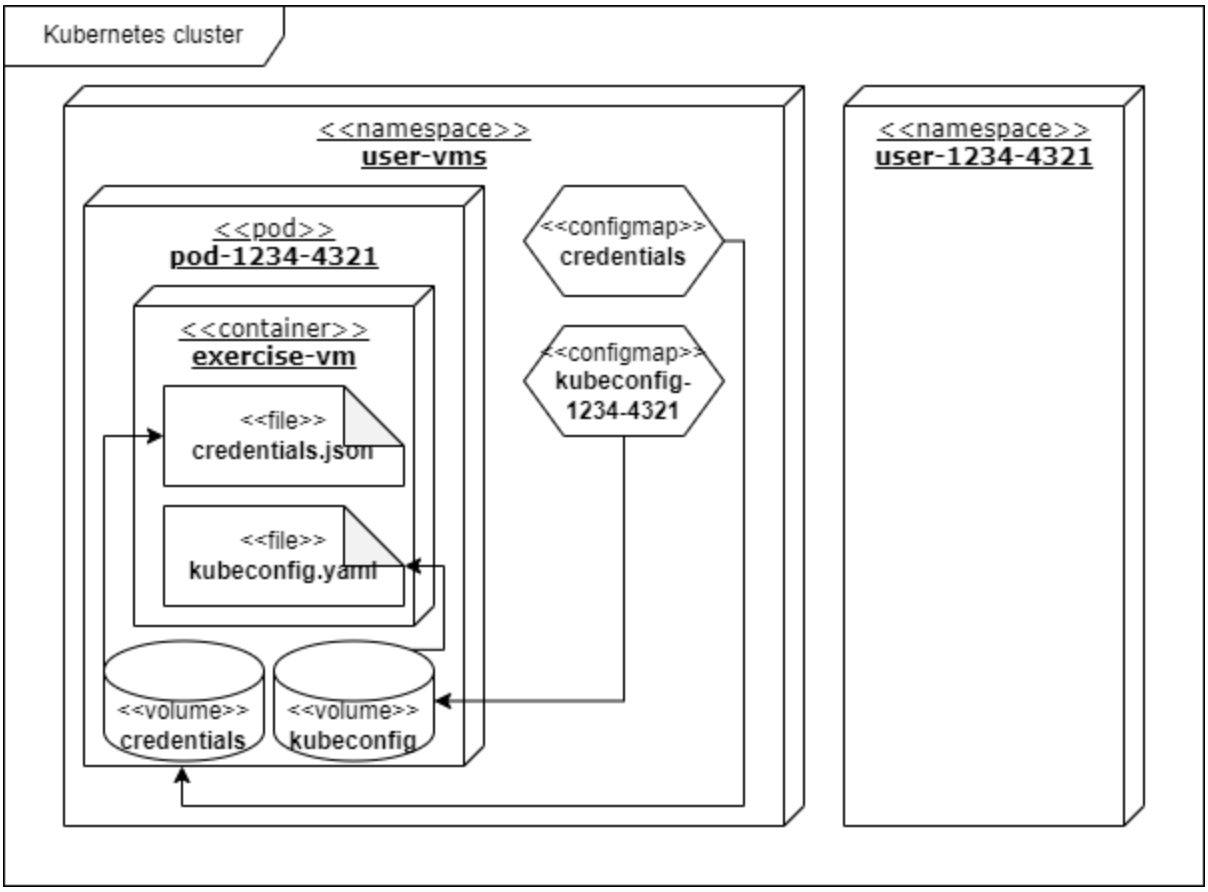

Figure 11. Kubernetes cluster diagram. User is free to use the exercise-vm container to execute commands or create new resources inside user-1234-4321 namespace.

When server receives a request to execute a command, it starts the bash script, which uses *kubectl* to execute the command inside the pod, which is identified by the user id and exercise id supplied with the request. Output of the command is then returned in the response body. If input stream is empty, then the output written to the error stream is returned.

2.3.4.2 Pod

Pods, created for each exercise, sleep for the time allocated on exercise and then terminate all resources created for this exercise including itself. Pods are created from the YAML template, which is modified for every new exercise using the *yq* utility. The following code listing contains the pod specification:

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: exercise-vm
  name: <change-me>
  namespace: user-vms
spec:
  containers:
    - command: <change me>
      env:
        - name: KUBECONFIG
          value: <change me>
        - name: GOOGLE_APPLICATION_CREDENTIALS
          value: /var/opt/credentials/credentials.json
      image: gcr.io/eastern-team-350219/exercise-vm:latest
      imagePullPolicy: IfNotPresent
      name: pod-template
      volumeMounts:
        - mountPath: /var/opt/credentials/
          name: credentials
        - mountPath: /var/opt/kubeconfig/
          name: kubeconfig
  restartPolicy: Never
  volumes:
    - configMap:
        name: credentials
      name: credentials
    - configMap:
        name: <change me>
      name: kubeconfig
```

Listing 1. Pod YAML specification.

Pod's name is changed according to the convention mentioned previously. Command is changed to

["/bin/sh","-c","sleep ${SLEEP_TIME} && kubectl delete ns ${NAMESPACE} && kubectl -n ${USER_VMS_NAMESPACE} delete cm ${CM_NAME} && kubectl -n ${USER_VMS_NAMESPACE} delete po ${POD_NAME}"]

where the variables are replaced by their respective values. Both config map name and *KUBECONFIG* environment variable value are changed according to the newly created config map name.

Image used for the pod is of big importance. It should be lightweight, so that new environments are created quickly, nevertheless, it must include the necessary for binaries for

the exercise. At the moment, only *kubectl* and *git* binaries are required, and the image is the same for each exercise environment.

The image is built from the lightweight Linux distro called alpine. *Dockerfile* specification of the image is in the next code listing.

```
FROM alpine:latest

RUN apk add git && \
        apk add curl && \
        curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl" && \
        install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Listing 2. Dockerfile contents.

2.3.4.3 Use-case scenario

As soon as user taps “Start exercise”, his id is retrieved from the local storage. Then, it sends a request to check if the environment for this exercise already exists. This can happen if the exercise was started but not completed or when it has been completed recently and the exercise environment is still terminating.

If user id was found locally, user id and exercise id are supplied with the request. Server executes the bash script and, depending on its exit code, returns either 0 if no environment was not found or 1 if the environment exists. In latter case user is asked to wait while the old environment is terminated and a termination request is sent. In former case a request to setup a new exercise environment is sent. In both cases, termination and creation of environments are made by separate bash scripts.

If no user id is found locally, then no exercise environment exists for this user. A request to setup a new exercise environment is sent. Since no user id is supplied with the request, a new user is inserted into the database and its id is returned and then stored locally.

When user finishes the exercise, a termination request is sent automatically. If user does not manage to finish the exercise in time, he is notified about this and is redirected to the final screen, where his score is calculated (and 0 points are awarded for the time component of the score).

2.4 Client

Client is a multiplatform application built with React Native. Its main purpose is to provide a user-friendly experience while also delivering the necessary information in easily consumable format.

2.4.1 React Native application

The application has three main screens accessible via the bottom tab navigation. There are Learn, Test and About tabs. Let's consider those in detail.

Learn tab is designed to gather learning resources in one place. That is, it has to possess two main features: a scrollable list of learning resources and a search window to search through this list. This tab as well as all of the described further fetches the data on the first render as there is no point in future requests due to the low rates of update in the database. For the same reason search filters through the data fetched before and does not make additional requests to the server. Each learning resource is listed as a card with the title, cover image and a short description. Upon click user is redirected to his default browser, where the learning webpage is loaded.

Figure 12. Learn tab.

Test tab is the most complicated one. It has two additional nested navigation stacks. By default test tab opens the Certifications screen, where available certifications are fetched from the server and listed as cards. The picture below displays one of the first mock-ups of the Certifications screen. Search feature is available at the top of the screen. By clicking the card user can navigate to the Exercises screen. Exercises screen has a list of exercises for the certification selected on the previous screen. Each exercise has the progress displayed as a progress bar behind the exercise item. Search feature is available here as well. Selecting the exercise activates next navigation stack. First, there is the starting screen, where the title, the type and current progress of exercise are displayed. Pressing the "Start"/"Continue"/"Retry" button (label depends on the user's progress in the exercise) navigates to the next screen. Next

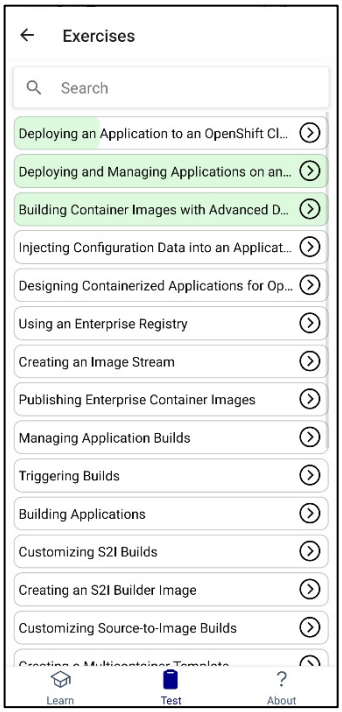

Figure 13. Exercises for the selected certification. Green background displays saved progress for each exercise.

screen is the Task screen. It has the task description, progress bar, timer and the answer view on it. Answer view looks different for every task type there is. For the tasks of type "auto" and "exec" there is a list of "bricks", from which the answer should be constructed and a terminal simulation window, for the "exec" tasks there is also an output window right under the terminal (see chapter *3.3 Final app* for the screenshots). For the single-choice, multiple-choice and text answers there are *RadioButton*, *Checkbox* or *Input* views respectively. "Continue" button rests on the bottom of the screen. To continue to the next task user must provide correct answer. Although, algorithm of answer evaluation is slightly different for each task type, basically, it all comes to string comparison. The answer, submitted by user, is compared with the expected answer fetched from the database. Incorrect parts of the answer are colored in red when answer is submitted. After all tasks have been correctly solved a congratulations screen appears. It displays the mistake count, duration and overall exercise score, which is calculated based on the number of mistakes, time spent and task weight. Two buttons rest at the bottom: "retry" and "finish". The former allows to start the exercise again from the first task. The latter navigates back to the exercises list.

The last one is About tab. About tab provides some basic information about the app and its purpose. There are also listed ways to get in touch with the developers.

2.4.2 Local storage

Although most of the data is stored in the database on the server side, there is also a bit of data stored locally on user's device. This allows to reduce the number of API requests and solves the problem of identifying client on server side.

Every user has its progress in each exercise saved locally, so that it can be displayed later. For each exercise there is the time spent and mistake count stored. Besides progress, user id is saved locally, too, and a separate static key used to access it.

This feature uses *@react-native-async-storage* module described earlier in the Technologies chapter.

2.5 Communication protocols

The only protocol used for the communication between the frontend and the backend is HTTP. Backend controllers accept GET, POST, PUT and DELETE requests. Each request receives a response from the server with a corresponding response code. That is, successful requests receive a 200 OK code in case the requested data were successfully found or 201 CREATED when the requested entity was successfully created. Failed requests are responded with the 404 NOT FOUND status or 401 UNAUTHORIZED if the request did not provide the necessary authorization header.

Each endpoint of the API is protected by the Basic Authentication. This means, that all requests are required to possess an Authentication header with the credentials in the following format:

Basic <auth_string>

where *auth_string* is a Base64 encoded username and password joined by the colon. For example, for user “user” and password “password” the Authentication header will look as follows:

Basic dXNlcjpwYXNzd29yZA==

Some endpoints also expect a request variable(s) passed either through the request path or by using the HTTP GET parameters. URL in the form of

http://<api_url>/<parameter1>/<parameter2>

where *<api_url>* is the base URL of the API, *<parameter1>* and *<parameter2>* are the path variables represents the former style. Whereas the latter style looks as foolows:

http://<api_url>?<key1>=<value1>&<key2>=<value2>

In the first case parameters are distinguished by their order only and in the second order of parameters does not matter as each value is assigned to a particular key.

2.6 Data formats

Data exchanged between the client and the server is in most cases has the JSON format. This allows to easily serialize objects to be used in request and response bodies. On the server side this is realized by returning POJO (Plain Old Java Object), which is automatically converted into JSON by Jackson serializer. Client side converts the response body to JSON using the *Response.json()* JavaScript method.

When only one property of the object is being requested and that property is represented by a single string, the response body has the text format. Then the client accesses it using the *Response.text()* method.

Server database is populated through the specific endpoint, which expects a JSON body. This body contains the list of the certifications, a list of exercises for each of them and a list of tasks for each exercise. Here is an example of such body:

```
[
  {
    "title": "EX-288",
    "description": "Red Hat OpenShift Development II: Containerizing Applications",
    "imageUrl": "https://www.redhat.com/profiles/rh/themes/redhatdotcom/img/red-hat-social-share.jpg",
    "exercises": [
      {
        "title": "Deploying an Application to an OpenShift Cluster",
        "type": "LAB",
        "time": 600,
        "tasks": [
          {
            "description": "1. Enter your local clone of the DO288-apps Git repository and checkout the master
branch",
            "helpers": [
              "DO288-apps",
              "docker-build",
              "master",
              "${RHT_OCP4_DEV_USER}",
              "${RHT_OCP4_DEV_PASSWORD}",
              "${RHT_OCP4_MASTER_API}",
              "https://github.com/${RHT_OCP4_GITHUB_USER}/",
              "echo-1-555",
              "cd",
              "git",
              "checkout"
            ],
            "answers": [
              "cd DO288-apps",
              "git checkout master"
            ],
            "type": "AUTO",
            "weight": 1
          }
        ]
      }
    ]
  }
]
```

Listing 3. An example of JSON body used to populate the certifications database.

This way, a one or a hundred of certifications can be imported into the database using the same entrypoint. JSON is parsed by the backend using the *org.json* package. Then objects are created and inserted into the database. From the example above the script will first insert a new category, then a new exercise, which will reference the newly created category, then a task referencing inserted exercise will be inserted into the database. The following method is responsible for parsing JSON entries:

```
    @PostMapping("/entry")
    ResponseEntity<?> newEntry(@RequestBody String input) {
        JSONArray categories = new JSONArray(input);
        for (int i = 0; i < categories.length(); i++) {
            JSONObject category = categories.getJSONObject(i);
            String categoryId = categoryRepository.insert(new Category(
                    category.getString("title"),
                    category.getString("description"),
                    category.getString("imageUrl")
            )).getId();
            JSONArray exercises = category.getJSONArray("exercises");
            for (int j = 0; j < exercises.length(); j++) {
                JSONObject exercise = exercises.getJSONObject(j);
                String exerciseId = exerciseRepository.insert(new Exercise(
                        exercise.getString("title"),
                        categoryId,
                        exercise.getEnum(ExerciseType.class, "type"),
                        exercise.getLong("time")
                )).getId();
                JSONArray tasks = exercise.getJSONArray("tasks");
                for (int k = 0; k < tasks.length(); k++) {
                    JSONObject task = tasks.getJSONObject(k);
                    taskRepository.insert(new Task(
                            exerciseId,
                            task.getString("description"),

task.getJSONArray("helpers").toList().stream().map(x ->
(String)x).collect(Collectors.toList()),

task.getJSONArray("answers").toList().stream().map(x ->
(String)x).collect(Collectors.toList()),
                            task.getEnum(TaskType.class, "type"),
                            task.getDouble("weight")
                    ));
                }
            }
        }
       return ResponseEntity.status(HttpStatus.CREATED).build();
    }
```

Listing 4. Endpoint for the data import. Objects are parsed using the JSONObject and JSONArray from org.json package.

Since the application does not directly write or read any files, no further data format specification is needed. Nevertheless, some data is being saved locally on the client device. Refer to the chapter *2.4.2 Local storage* for further explanation.

3 Implementation

This chapter is dedicated to my implementation of the app. Chapter is split up into the two parts describing backend and frontend parts respectively.

3.1 Backend

Implementation the backend mostly comes to the development and deployment of the Spring Boot application since Mongo database requires almost no additional configuration. We'll first look at the few steps required to get it running as intended.

3.1.1 MongoDB configuration

For security reasons Mongo database shouldn't be accessed without authentication. A separate user for the Spring to use should be created. User can be created with one line of code. After connecting to the MongoDB and typing the command

db.createUser({user: "spring", pwd: "password", roles: [{role: "readWrite", db: "drillz"}]})

a new user will be created. Then the credentials can be passed to Spring via the application properties.

3.1.2 Sping Boot application

The following class diagram gives an overview of the Spring Boot application structure.

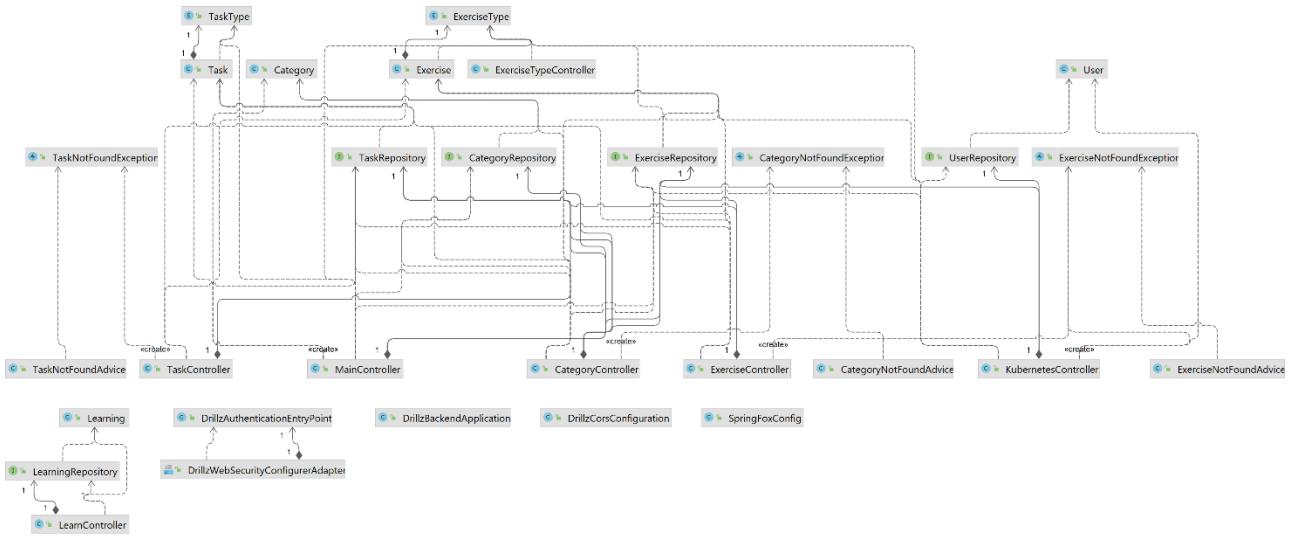

Figure 14. UML class diagram.

Traditionally, Spring Boot applications rely on annotations to implement the dependency injection paradigm. That's why it is crucial to go through the annotations I have used to implement the app. Next chapters focus on Lombok and Spring annotations.

3.1.3 Lombok annotations

Lombok annotations are there to make developer's life easier. Instead of rewriting the same chunks of code over and over again, developers can use annotations generating those pieces of code for them.

3.1.3.1 @Data

This annotation is designed specifically for data models and applied to the model class. It combines *@ToString*, *@EqualsAndHashCode*, *@Getter*, *@Setter* and *@RequiredArgsConstructor* annotations. Therefore, it generates *toString(), equals(), hashCode()* methods, a constructor with accepting all required fields as arguments, getters and setters for every applicable field.

3.1.3.2 @NonNull

@NonNull annotation marks a class field as required. In addition, Lombok throws an exception if the field value was set to null.

3.1.4 Spring annotations

Spring applications are driven by the annotations [11]. That's why I am explaining some them below.

3.1.4.1 @SpringBootApplication

This annotation tells Spring Boot to use auto-configuration, component scan and allowes to define extra configuration on their "application class". A single @SpringBootApplication annotation can be used to enable those three features, that is:

- *@EnableAutoConfiguration*: enable Spring Boot's auto-configuration mechanism
- *@ComponentScan*: enable *@Component* scan on the package where the application is located

- *@Configuration*: allow to register extra beans in the context or import additional configuration classes

3.1.4.2 @RestController

This annotation combines the Spring *@Controller* and *@ResponseBody* annotations, making it easier to implement controllers. *@ResponseBody* annotation enables automatic serialization of the return object into the *HttpResponse* object.

3.1.4.3 @GetMapping, @PostMapping, @PutMapping, @DeleteMapping

These four annotations act as a shortcut for *@RequestMapping* annotation, each passing its own value to the method argument. They expect a path passed as a default argument. This path is then mapped to the method annotated by any of those annotations. This means that the request made to this path will be processed by the mapped method. Multiple mappings of the same method with the same path are not allowed and will produce an error during the Spring initialization.

3.1.4.4 @RequestParam, @PathVariable, @RequestBody

These three annotations handle the data sent with request. *@RequestParam* annotation indicates that a method parameter should be bound to a web request parameter. This allows us to handle requests with URLs like

http://<api>/<endpoint>?param=value

with the following method:

```
@GetMapping("/<endpoint>")
ResponseEntity<?> get(@RequestParam String param) {
    // Some application logic which makes use of the param
    return ResponseEntity.ok().build();
}
```

Listing 5. @RequestParam usage example

@PathVariable annotation indicates that a method parameter should be bound to a URI template variable. This allows us to handle request variables embedded in the URL like

http://<api>/<endpoint>/value

with the following method:

```
@GetMapping("/<endpoint>/{param}")
ResponseEntity<?> get(@PathVariable String param) {
    // Some application logic which makes use of the param
    return ResponseEntity.ok().build();
}
```

Listing 6. @PathVariable usage example

@RequestBody annotation indicates, that a method parameter should be bound to the body of the web request. Jackson serializer is able to parse the body into the object instance, so that JSON objects can be passed to the backend using the request body.

3.1.4.5 @Bean

Indicates that a method produces a bean to be managed by the Spring container. It must return an object, which can be then autowired by Spring.

3.1.4.6 @Autowired

Marks a constructor, field, setter method, or config method as to be autowired by Spring's dependency injection facilities. Basically, the object is instantiated and controlled by Spring automatically.

3.1.5 Exception handling

Application handles some exceptions separately to return more specific error message. For each model there exists its own exception and its own handler. Let's consider, for instance, the case when the Category with a specific id was requested but wasn't present in the database. Category controller throws a custom exception, which is defined in the following code listing:

```
public class CategoryNotFoundException extends RuntimeException {
    CategoryNotFoundException(String id) {
        super("Could not find category with id " + id);
    }
}
```

Listing 7. Custom error class

This exception is recognized by the advice, so it sends a response, wrapping the exception message in the response body. The code listing below contains the code for this exception's advice.

```
@ControllerAdvice
public class CategoryNotFoundAdvice {
    @ResponseBody
    @ExceptionHandler(CategoryNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    String categoryNotFoundHandler(CategoryNotFoundException e) {
        return e.getMessage();
    }
}
```

Listing 8. Controller advice implementation

3.1.6 Bash scripts

A lot of work is put into the bash scripts running on server side. They are the main wheel of "field" task functionality. Let's consider each of them in detail.

Firstly, there is the *check_environment_existence.sh* script. It is called by Kubernetes controller upon every valid request to the *environmentExists* endpoint. The script expects two arguments, user id and exercise id, to be passed. It exits with 0 if the respective environment was not found or 1 in other case.

```
#!/bin/bash

USER_ID=$1
EXERCISE_ID=$2

if kubectl get ns | grep -q "ns-$USER_ID-$EXERCISE_ID" || kubectl -n user-
vms get po | grep -q "pod-$USER_ID-$EXERCISE_ID"; then
        exit 1
else
        exit 0
fi
```

Listing 9. check_environment_existense.sh

Secondly, there is *prepare_exercise_environment.sh* script, which is responsible for the creation of the namespace, pod and config map. First two arguments are user id and exercise id respectively, and the third one is a container life time in seconds. This script is called whenever a valid request to the *startExercise* endpoint is received and the exercise with the supplied exercise id is of "field" type. This script calls a small helper bash script designed to create a YAML definition of a config map, which contains the Kubernetes configuration file.

Thirdly, the *execute_exercise_command.sh* script is the one used to execute commands inside the pod. Again, first two arguments are ids, and all extra arguments are treated as a command.

The script is called upon every valid request to the *exec* endpoint. Its output is returned with the response body.

```
#!/bin/bash

POD_NAME="pod-$1-$2"
COMMAND=${@:3}
NAMESPACE="user-vms"

kubectl -n $NAMESPACE exec -t $POD_NAME -- /bin/sh -c "$COMMAND"
```

Listing 10. execute_exercise_command.sh

Lastly, to clean up after the exercise is finished a *clear_exercise_environment.sh* bash script is used. Traditionally, it expects ids as the first two arguments. However, if *null* is received as exercise id, the script deletes all the exercise environments for the user with the provided user id. This is useful to be invoked on the app close event. The script is responsible for the termination of the pod, namespace and the config map.

```
#!/bin/bash

USER_ID=$1
EXERCISE_ID=$2

clear() {
        kubectl delete ns ns-$USER_ID-$EXERCISE_ID
        kubectl -n user-vms delete po pod-$USER_ID-$EXERCISE_ID
        kubectl -n user-vms delete cm kubeconfig-$USER_ID-$EXERCISE_ID
}

if [ $EXERCISE_ID = "null" ]; then
        for ns in $(kubectl get ns -o=custom-columns=':.metadata.name'|
grep "ns-"); do
           EXERCISE_ID=$(echo $ns | sed -E "s/ns-[^-]{24}-([^-]{24})/\1/g")
           clear
        done
else
        clear
fi
```

Listing 11. clear_exercise_environment.sh

3.2 Frontend

Frontend implementation is way more tricky. A lot of time is spent on tweaking little things like animations, positions and styles. Another thing to keep in mind is the multiplatform aspect. Every component or third-party library used should be compatible with both iOS and Android platforms. Below are screenshots of the user interface.

3.2.1 React Native application

The biggest question when developing with React Native is whether to use classes or function as components. Class components allow developers to use React lifecycle methods and benefits of classes themselves, whereas functional components mostly just responsible for rendering the UI based on the passed properties. Personally, I prefer to use functional components and keep my logic separated from the rendering mechanisms. Furthermore, hooks can be conveniently used in functional components to make them stateful.

The most interesting feature of the frontend is the "auto" and "exec" tasks. They allow user to construct a command from the offered parts. The feature is implemented using two stateful arrays, and the "bricks" are rendered using the *Badge* component from *'react-native-paper'* library. Initially, terminal window has a number of empty "bricks" equal to the number of words received by splitting the answer by spaces. Upon user tap on one of the "bricks", it is being removed from the first array and appended to the second one, which is displayed in the terminal window. Same process is applied upon tapping on one of the non-empty "bricks" in the terminal window.

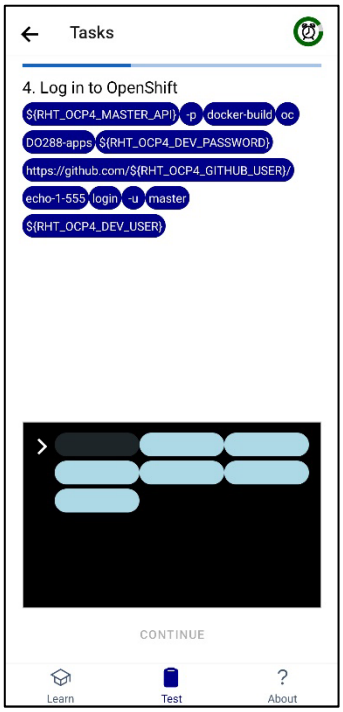

Figure 15. "Auto" task screen. Timer is running in the header. Tapping on the "brick" will append it to the answer sequence.

3.2.2 Hooks

Hooks are one of the recent additions to the React. They were released just in the winter of 2019 and solved many of React's problems [12]. Hooks allow you to use state and other React features without writing a class. Hooks are the functions which "hook into" React state and lifecycle features from function components.

3.2.2.1 useState()

Returns a stateful value, and a function to update it. Basically, this hook allows us to preserve data while using the same function to re-render the view. It is the most commonly used hook in

my implementation since I am only using functional components and this hook is a very straightforward way to make them stateful.

3.2.2.2 useRoute()

Returns the route prop of the screen it's inside. I am mostly using it to access route parameters when navigating between different screens.

3.2.2.3 useEffect()

This hook allows to execute code after the render process is finished. It also allows to specify a list of dependencies, which trigger the effect when at least one of their values change. An empty list of dependencies means that the effect will be executed only once for the whole app run.

3.2.3 Animations

Animations are very important to create a great user experience. Stationary objects must overcome inertia as they start moving. Objects in motion have momentum and rarely come to a stop immediately. Animations allow developers to convey physically believable motion in their interface.

React Native provides two complementary animation systems: *Animated* for granular and interactive control of specific values, and *LayoutAnimation* for animated global layout transactions. In my implementation I use only the former one.

Animations in React Native are composed in ordered sequences of actions by *Animated.sequence()* method or executed in parallel by *Animated.parallel()* method. Both of them accept an array of actions, which modify the object state. Actions can be of different types depending on the nature of the action. Among offered types is, for instance, *Animated.timing()* method, which supports animating a value over time using one of various predefined easing functions, or a custom one.

3.2.4 Async Storage

To conveniently store and retrieve the progress of exercises locally I have defined two functions. The listing below contains the declaration of those two functions.

```
export const storeData = async (value) => {
  try {
    const jsonValue = JSON.stringify(value)
    await AsyncStorage.setItem(ITEM_KEY, jsonValue)
  } catch (e) {
    console.error(e);
  }
}

export const retrieveData = async () => {
  try {
    const jsonValue = await AsyncStorage.getItem(ITEM_KEY)
    return jsonValue != null ? JSON.parse(jsonValue) : null;
  } catch(e) {
    console.error(e);
  }
}
```

Listing 12. Async storage data manipulation

Besides the two functions described above, there are also utility functions designed for manipulation with user id, which is also being stored locally.

3.3 Final app

Here I have gathered some screenshots of the app made on Android device. They provide a basic understanding of how the app looks and feels.

Figure 17 shows the certifications screen, which appears when user visits the Test tab first time. For each certification there is a title, a short description and a cover image. Figure 18 shows an example of multiple-choice task. On the screenshot user has tapped the “continue” button, but his answer was not correct. Application highlights the wrongly selected options. Exercise finishing screen is shown on the Figure 19. It is accompanied with particles to make the message more congratulatory. Figure 20 shows the previously completed exercise. The description of the task type is loaded from the backend. Task of type “exec” is shown on Figure 21. Since the user has provided a correct answer, the “next” button is active. The output of the executed command is written to the textarea. However, on Figure 22 the answer submitted by the user wasn’t correct. The command was executed and outputted the error message. “Next” button is not active since the answer is wrong.

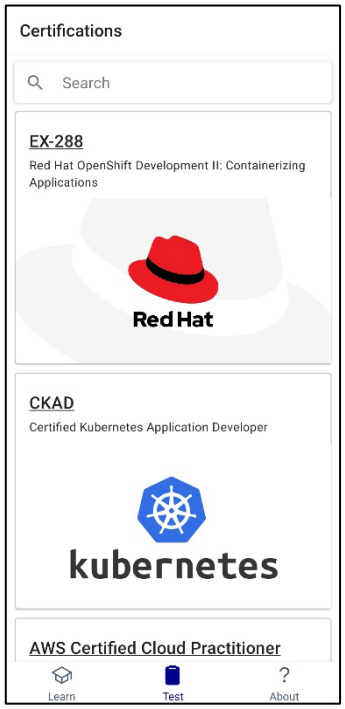

Figure 16. Test tab.

Figure 17. "Multiple-choice" task. Wrong answers are colored in red.

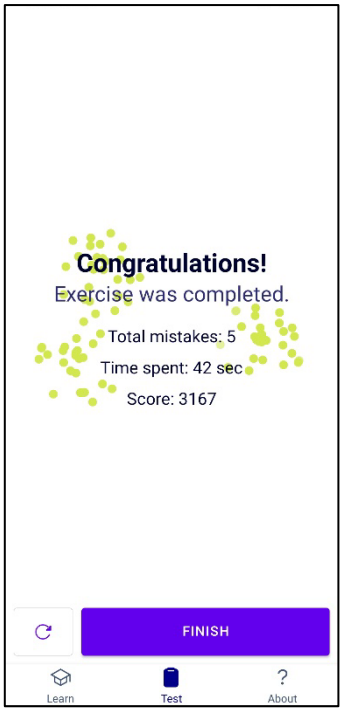

Figure 18. Congratulations screen. Total number of mistakes, time spent and score are displayed.

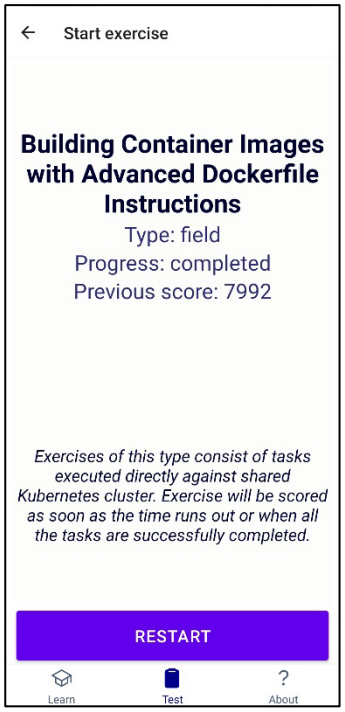

Figure 19. Previously completed exercise. Previous score is retrieved from local storage and displayed.

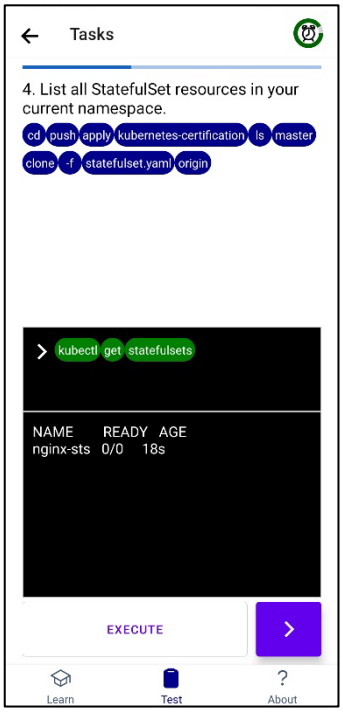

Figure 20. "Exec" task screen. Command was successfully executed and the output is displayed to the user.

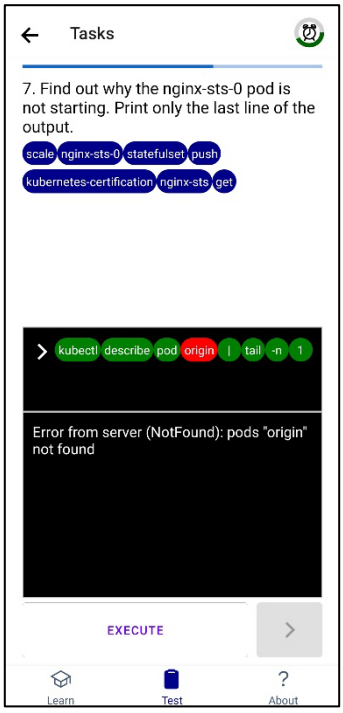

Figure 21. "Exec" task screen. Since the command is wrong, continue button is disabled.

3.4 App test

The app was tested using different iOS and Android devices. The common test process consists of the following steps designed to test the system behavior in different conditions:

1. User opens the app, the “Learning” tab appears
2. User searches through the Learnings, only Learnings matching the search criteria are left
3. User taps on one of the search results, a browser window opens
4. User returns to the app
5. User navigates to the “Test” tab
6. User taps on the certification, the “Exercises” screen opens
7. User starts one of the exercises
8. User tries to submit both wrong and right answers
9. User exists the exercise, returns to the “Exercises” screen
10. User continues the exercise from where he left
11. User completes the exercise
12. Steps 7-11 are repeated for “test”, “lab” and “field” exercise types
13. User reviews the “About” tab

There were three full tests completed on different devices. The test reports can be found below.

3.4.1 Test report 1

The participant possessed an iPhone 13 Pro Max. Since I didn’t have access to the Mac or iOS system during the development, this was the first time the app was tested on iOS device.

The participant was satisfied with “Learning” tab. It looked exactly as was intended and the search was working properly. However, he had encountered several problems while completing the tests. Firstly, text on the “bricks” was looking too small on the large screen of his device. Secondly, the “arrow-back” icon in the header had transformed into simple line, but still was pressable. Thirdly, the iOS keyboard has overlapped the input field during the “text” task. While the first two issues are not critical, the last one will be our first priority when updating the app.

Overall, though, participant was content with the app. He found the app enjoyable and has asked to send him a download link when the app will be officially released into the production.

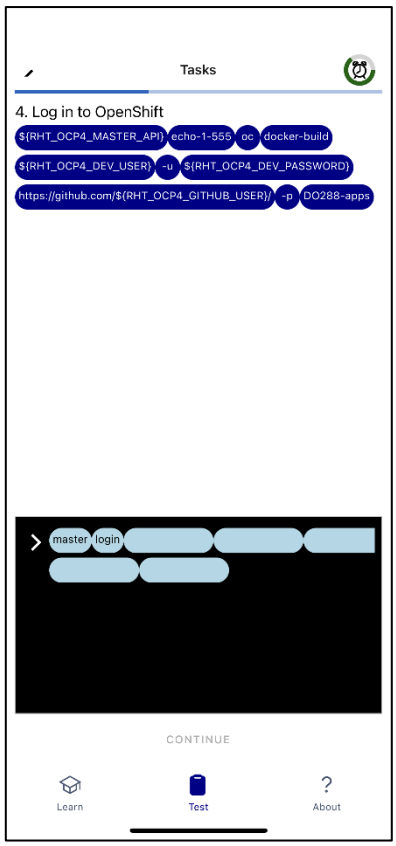

Figure 22. "Auto" task on iOS device. The font is too small and the line in the header is supposed to be an arrow.

3.4.2 Test report 2

The second participant had the Android device. The device name was Xiaomi POCO X3 NFC. App was tested multiple times on Android emulators during the development and was running smoothly, nevertheless, during this test a few issues were encountered.

The participant found out that the app does not restricts the orientation, but when switching to the landscape mode, "Test" and "About" tabs become practically unusable. This issue must be addressed in the first patches, and the orientation should be locked to the portrait mode. Besides that, the participant pointed out, that the timer does not bring attention and it is easy to forget about the time. He suggested the timer to start flashing when the time is about to run out.

In the end the participant stated that the app left a good impression and the tests were interesting but rather hard for him.

3.4.3 Test report 3

The third participant tested the app on his Xiaomi POCO M3. Overall run was satisfactory, yet a few things could be improved according to the participant.

He noticed the simplicity of the design and called it user-friendly. The participant especially liked the interactive "bricks" exercises. He didn't like that exercise names are truncated on the "Exercises" screen. The tester has also suggested that the app would benefit from the progress tracking system.

No critical issues were encountered during this test, so it can be concluded as successful.

Conclusion

The goal of this bachelor thesis was to develop a mobile application for Android and iOS devices, which would be a helpful assistant to every IBM employee, who is wishing to prepare for the certification. Since certifications have different formats of questions, my mission and the most challenging part was to implement various kinds of tasks. The app has been finished and satisfied all of the requirements. Not only it supports a variety of task types, it is also able to provide a Linux environment and a separate Kubernetes namespace for practicing to each user.

While the official backend is still in development by other team, I asked colleagues to test and give feedback on my application. Feedback turned out to be mostly positive, though, there are also a lot of opportunities for improvement.

Everyone found the app useful when it comes to preparing for the certification. Designed for mobile devices, it allows to keep practicing on-the-go and yet still have the opportunity to execute commands on the remote server. Some also noticed that it is convenient to be able to find good courses for the exam preparation since they can search for learning resources right in the app. Nevertheless, according to the test reports, there are still some issues on iOS devices that have to be fixed as soon as possible. Besides that, there are a lot of ways in which the app could be improved in future.

To begin with, learning section can be integrated into the app, so that there is no need to open browser to access learning resources. Each learning page can also refer to the corresponding exercises tab allowing users to exercise right after they complete a particular learning topic. Then, the “field” tasks can be extended to support Openshift clusters. This way, users can practice not only *kubectl* commands, but also the *oc* ones. App’s answer evaluation system can benefit from the improvement as well. For the “field” tasks we can compare the actual state of the environment with the desired one. This can be done by capturing snapshot of the environment into YAML file and then checking the correctness of specific properties. Finally, a good way to motivate users for completing the exercises would be streaks and goals. By completing at least one exercise a day user maintains a streak. There could also be an achievement system, where user is awarded with achievements for completing the goals. Maintain a streak for seven days – might be one of the goals. This might be extended further:

users can have an option to share their achievements among friends, thus, comes the need for authorization and friendship system.

All things considered, the system design and implementation received commendation from the project manager. Here his comment on the accomplished work: “We had a need of some kind of tool to prepare the candidates for various enterprise certifications. Although some of the platforms exists most of them are paid and are lacking our specific needs. Thus, we decided on creating a mobile app which will be able to provide the training environment for certifications. The application provided by Pavel fits the gap and covers all the necessary requirements. On top of that the application is able to validate the answers against existing environment what is nice add-on feature to have. For the near future we plan to integrate the application with our certification portal which provides all the information for the candidate to prepare for certification.”

Bibliography

[1] Amazon, "Amazon certified cloud practitioner certification," 15 January 2022. [Online]. Available: https://aws.amazon.com/certification/certified-cloud-practitioner/.

[2] Google, "Certifications Questions," 15 January 2022. [Online]. Available: https://play.google.com/store/apps/details?id=com.certquestions.mobile.app&gl=SK.

[3] Apple, " PMP certifications mastery," 15 January 2022. [Online]. Available: https://apps.apple.com/us/app/pmp-certification-mastery/id1150587715.

[4] "Spring Framework Documentation," April 2022. [Online]. Available: https://docs.spring.io/spring-framework/docs/current/reference/html/.

[5] "Spring Boot Reference Documentation," April 2022. [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/html/.

[6] "Project Lombok," April 2022. [Online]. Available: https://projectlombok.org/.

[7] "MongoDB documentation," April 2022. [Online]. Available: https://www.mongodb.com/docs/manual/reference.

[8] "What is Maven?," April 2022. [Online]. Available: https://maven.apache.org/what-is-maven.html.

[9] "React Native," April 2022. [Online]. Available: https://reactnative.dev/.

[10] "Async Storage," April 2022. [Online]. Available: https://react-native-async-storage.github.io/async-storage/.

[11] "Spring Framework 5.3.20 API," May 2022. [Online]. Available: https://docs.spring.io/spring-framework/docs/current/javadoc-api/.

[12] "Introducing Hooks," May 2022. [Online]. Available: https://reactjs.org/docs/hooks-intro.html.

[13] Slack, 15 January 2022. [Online]. Available: https://slack.com/.

Attachments

There are no attachments to this thesis. Source code of the system cannot be made publicly available as it is considered an IBM intellectual property.