SAVVAS REALIZE – AN ONLINE LEARNING EXCHANGE PLATFORM

A Project Report

Submitted by

Umang Kamdar(AU1841069)

In partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY
IN
INFORMATION AND COMMUNICATION TECHNOLOGY(ICT)
at

Ahmedabad University

School of Engineering and Applied Sciences (SEAS)
Ahmedabad, Gujarat
May, 2022

DECLARATION

I hereby declare that the project entitled “SAVVAS REALIZE – AN ONLINE LEARNING EXCHANGE PLATFORM” submitted for the B. Tech. (ICT) degree is my original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.

Signature of the Student

Place: Ahmedabad

Date: 6th May, 2022

CERTIFICATE

This is to certify that the project titled "SAVVAS REALIZE – AN ONLINE LEARNING EXCHANGE PLATFORM" is the bona fide work carried out by Umang Kamdar, a student of B Tech (ICT) of School of Engineering and Applied Sciences at Ahmedabad University during the academic year 2021-2022, in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (Information and Communication Technology) and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Signature of the Guide

Place: Ahmedabad
Date: 6th May, 2022

ABSTRACT

An Online Learning Management System (LMS) is a web software that allows to create, deliver, track, and report on educational courses and outcomes. It may be used to support both traditional face-to-face and blended/hybrid and distant learning situations. LMS software is used in schools to plan, implement, facilitate, assess, and track student learning. All of these transactions take place behind a virtual barrier that ensures authentication, privacy and security.

“Savvas Realize” is an online learning platform for K-12 students. Currently, it’s a monolith web application where all the frontend and backend services run on the same server. The task is to convert this monolith web application into different microservices with upgraded tech stack. This will minify the load to the server and also will be able to scale high as per the need.

ACKNOWLEDGEMENT

I, the Software Development Intern at PlayPower Labs, would like to offer the project "SAVVAS REALIZE – AN ONLINE LEARNING EXCHANGE PLATFORM". This project's approach has provided me with several opportunities to think about, apply, and engage with various facets of management as well as new developing technology.

Every successful project relies on the ongoing encouragement, good will, and support of those around you. I'd want to take this opportunity to thank a lot of folks who have given their time, complete support, and cooperation. Mr. Kenil Domadia, the Technical Leader, deserves particular appreciation for giving me with this chance, as well as Mr. Manan Gajjar and Yash Tank, who took the time to listen, guide, and keep me on track.

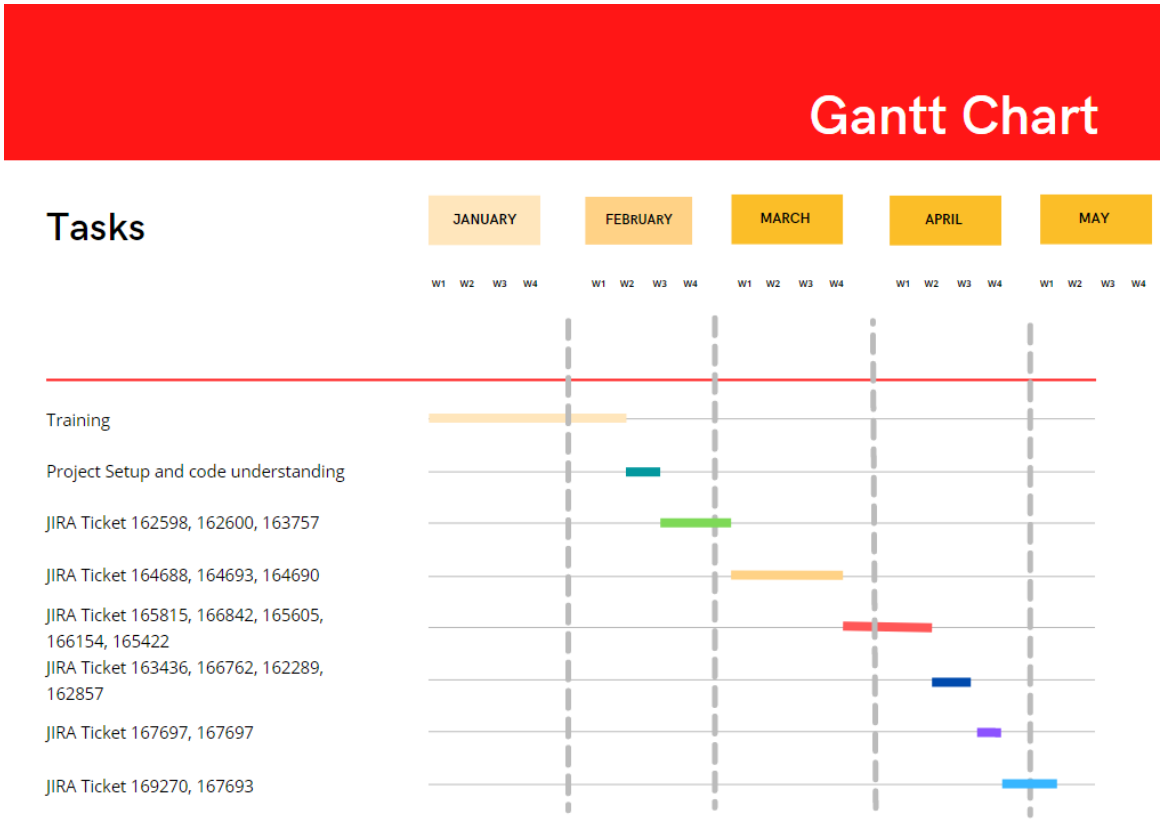
Prof. Shashi Prabh, my internal advisor, deserve special thanks for their help during the Internship. I have been motivated to put in long hours and learn new technologies and concepts. This opportunity is a significant step forward in my professional growth. I will make every effort to put my newly acquired skills and knowledge to the greatest possible use.

Umang Kamdar (AU1841069)

Table of Contents

LIST OF FIGURES

LIST OF TABLES


Gantt Chart
Tasks
JANUARY
FEBRUARY
MARCH
APRIL
MAY
W1 W2 W3 W4 W1 W2 W3 W4 W1 W2 W3 W4 W1 W2 W3 W4 W1 W2 W3 W4
Training
Project Setup and code understanding
JIRA Ticket 162598, 162600, 163757
JIRA Ticket 164688, 164693, 164690
JIRA Ticket 165815, 166842, 165605, 166154, 165422
JIRA Ticket 163436, 166762, 162289, 162857
JIRA Ticket 167697, 167697
JIRA Ticket 169270, 167693

Gantt Chart

CHAPTER 1: INTRODUCTION

1.1 COMPANY OVERVIEW

Play Power Labs is an ed tech company whose main focus is on building software and games that would help you to enhance learning with the help of AI. The major clients of the company are Savvas (formerly Pearson K12 Learning), Redbird, BrainPOP and even working with CBSE board of our country. The company ensures and provides high quality EdTech services which have been designed and researched in depth. I am currently working on Savvas (formerly Pearson K12 Learning) project with the backend team.

1.2 PROBLEM DEFINITION

The current Savvas Realize is a monolithic web application where frontend & backend runs on the same server. It is using assessment-service (legacy service) to interact with the databases. Now, our task is to convert this monolithic application into different microservices. For this we have to remove client proxies from our current application and directly implement the methods from assessment-service (legacy service). As of now, legacy assessment service is in older Java version. There is need to migrate assessment and mastery data to a separate microservice with RBS Authentication support.

1.3 PROJECT OVERVIEW

Currently, Assessment Service is hosted as part of Realize environment and is accessible only by Realize monolith backend. We intend to carve the Assessment Service out of the Realize. We plan to take care of dependencies and migration while doing so.

- Move the assessment service to RBS authentication
- Use the latest SpringBoot version for the rewrite
- The migration would take place in a phased manner
- We also plan to identify risks in each phase and propose ways to mitigate them

We use Agile Scrum methodology in the company. We have a 2-week sprint. This methodology is a style of project management that emphasises incremental progress. Each iteration is divided into two to four-week sprints, with the objective of completing the most essential features first and delivering a possibly deliverable product at the end of each sprint. In successive sprints, more features are added to the product, which is then tweaked depending on stakeholder and consumer input in between sprints. A sample JIRA board is shown which shows the different stages:

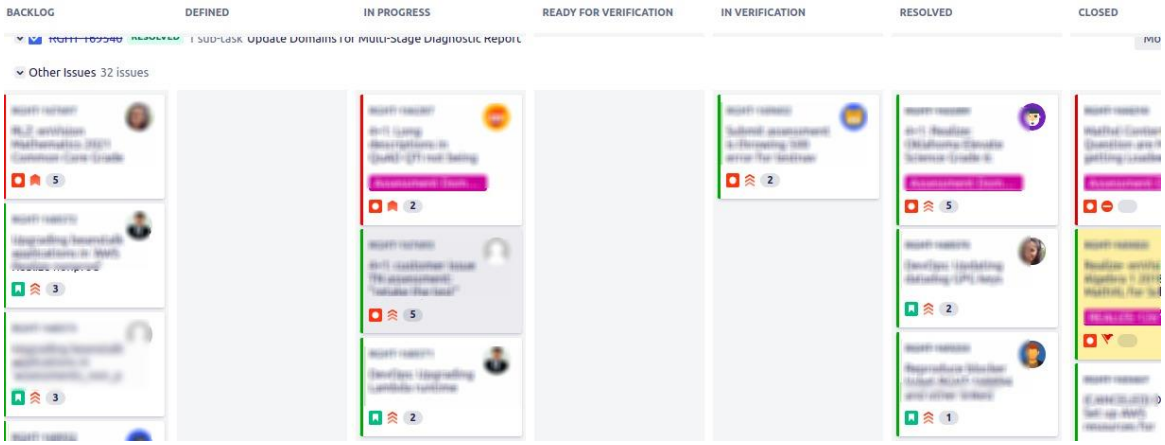


Fig. 1.3 Sample JIRA Board

1.4 HARDWARE REQUIREMENTS

No external hardware is required for implementation. Normal configured Personal Computer can handle the specific system.

Minimum Requirements:

- RAM: 8.0 GB
- Processor: Intel core i5 CPU
- Input Device: Mouse, Keyboard.

1.5 SOFTWARE REQUIREMENTS

- Operating System: Linux/MacOS
- Java version: 8 and 11
- Tools and Technology:
    - IntelliJ, VS Code, Git, Postman, PgAdmin, Jira, various AWS services, Jenkins, DataDog
- Framework:
    - Spring Boot, Angular, NodeJS
- Databases:
    - Postgres, DynamoDB

CHAPTER 2: LITERATURE SURVEY

2.1 EXISTING SYSTEM

The current system is a monolith web application. This means the backend and frontend runs on the same server. Monolithic architecture is regarded as a conventional method of developing apps. A monolithic application is made up of a single, indivisible piece of software. A client-side user interface, a server-side program, and a database are often included in such a system. It is unified, with all functions handled and served from a single location.

Monolithic applications are characterised by a single big code base and a lack of modularity. Developers use the same code base when they wish to update or replace something. As a result, they make modifications to the entire stack at the same time.

2.1.1 Pros and Cons of Monolith Architecture

Pros	Cons
Easier troubleshooting: Because the app's code is all in one location, it's much easier to go through it and find problems. Monolithic design also allows you to do end-to-end testing faster because it is indivisible.	Updates to the code: Because monolithic apps have all of the code in one location, each change to the code affects the entire system, hence any changes must be carefully prepared. In this respect, code modifications are more difficult to implement and take substantially longer.
Faster development: Monolithic apps are fundamentally easier to create due to their simplicity. Monolithic apps are simple to design since they often employ old systems and have everything in one place; you build, test, deploy, debug, and eventually grow as needed.	Reduced scalability: Monolithic programs are slow to respond to code changes. It swells and becomes a monster of information that is difficult to grasp and impossible to scale as more code and features are added.
Less latency: Because everything is confined in one location, monolithic apps	Low Reliability: Because of how tightly connected its components are in monolithic

are more streamlined. As a result, communication and processing times between services are reduced, and latency concerns are reduced.	app development design, one minor error might ruin the entire app. In this respect, if just one minor item goes wrong, the entire program might be rendered unusable.
Faster deployment: Don't have to deal with many implementations since monolithic app development architecture just requires one directory.	Modern technology has limitations: When designing monolithic apps, it's practically hard to adopt new technology. Because a little change to the system impacts the entire project, you must employ the same tech stack from start to finish.

Table 2.1.1 Monolith Architecture Pros and Cons

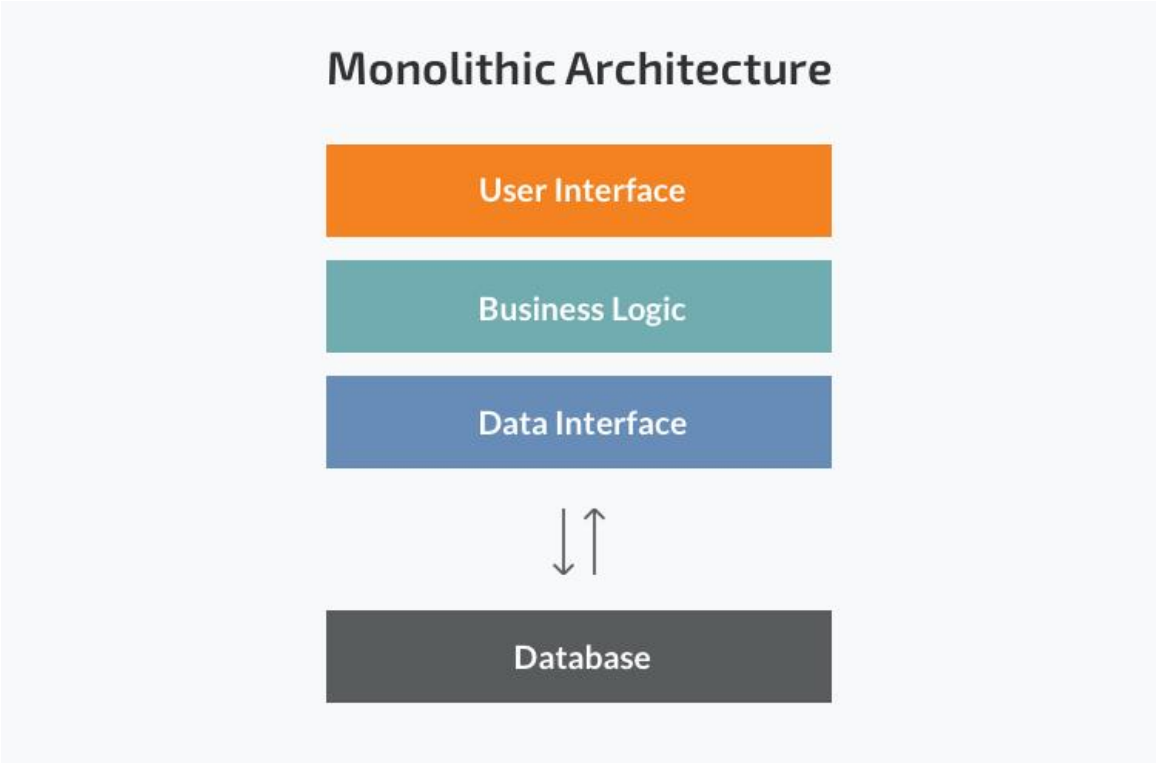


Fig. 2.1.1 Monolith Architecture

2.2 PROPOSED SYSTEM

The proposed system is to use different microservices instead of the monolith architecture. Thus, there will be different microservices handling different tasks. It is a more current, efficient, and mature approach to application development than monolithic design. Individual services that are isolated and act independently are used to develop the apps and their infrastructure in a microservices paradigm. These discrete services, or modules, each have their own databases and logic, yet they nonetheless function together as a distributed system in which the components maintain regular API contact. In other words, a microservices app works in the same way as a monolithic program and performs the same activities, but each of its "pieces" is a separate service.

2.2.1 Pros and Cons of Microservice Architecture

Pros	Cons
Enhanced productivity: Microservices design allows you to manage your app's components as distinct services, making it easier to create and maintain large apps no matter how big your team is. Without losing efficiency, you may divide responsibilities and have each team accomplish various things at the same time.	More complexity: Microservices apps are substantially more complicated than monolithic ones. The complexity of a microservices architecture grows as the number of microservices grows since it is a distributed system.
Increased scalability: Microservices design allows for independent deployments, which means that each microservice may be operated independently. As a consequence, instead of updating the entire program, you may update each microservice separately. You may also develop each service in a separate language, select the appropriate stack for each module, build once, and grow as you add additional functions and features	Microservices are less secure than monolithic programs since all of the services must interact frequently.

without jeopardising the system's compatibility.	
Horizontal scaling: If it turns out that your users are abusing a particular feature and that your microservices are overburdened, you may just expand that microservice rather than the entire system.	More expensive: The design of microservices app development is intrinsically associated with higher workloads, resulting in additional operational cost. Because all of the microservices in the system must connect with one another, there are a lot of remote calls, which increases network latency, runtime, and resource costs.
Microservices design allows you to make easy modifications whenever a newer technology comes out on the market. To adapt any new technology, your team may simply make a tiny adjustment in each service module without any additional difficulty or labour.	More difficult testing: Microservices architecture and its independently deployable elements are a pain to test due to their dispersed nature.
Bugs or malfunctions that impact one microservice don't affect the others, which means the remainder of the program is unaffected, making debugging faster and easier.	

Table 2.2.1 Microservice Architecture Pros and Cons

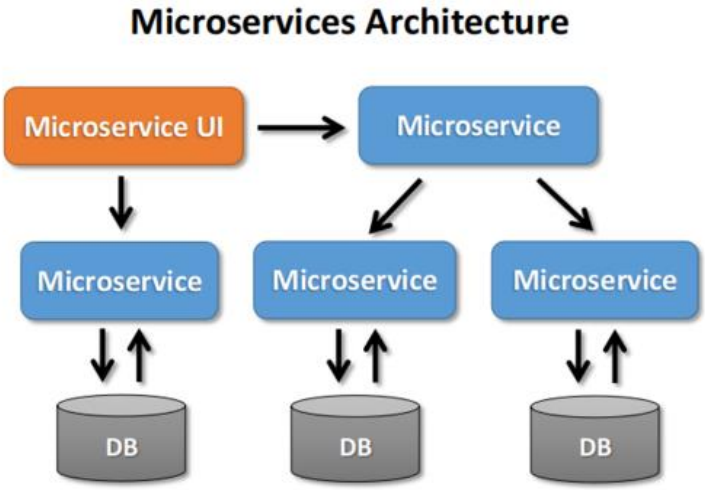


Fig. 2.2.1 Microservice Architecture

2.3 FEASIBILITY STUDY

2.3.1 Technical Feasibility

The main technologies used in the project are:

- IntelliJ
- VS Code
- Git
- Postman
- PgAdmin
- Jira
- Various AWS services
- Jenkins
- DataDog
- Java - Spring Boot
- JavaScript - Angular
- JavaScript - NodeJS
- Postgres
- DynamoDB

2.3.2 Financial Feasibility

The web application has extensive use of many different AWS services. Few of the AWS services used are:

- EC2
- Amazon RDS
- Amazon S3
- Amazon Lambda
- SNS
- SQS
- CloudWatch
- DynamoDb
- IAM
- Elastic BeanStalk

This all services mainly adds the total cost of web application.

CHAPTER 3: SYSTEM ANALYSIS

3.1 REQUIREMENT SPECIFICATION

3.1.1 Purpose

The purpose of the current system is to enable online learning available to students. Students will get access to a resource pool. The system will break through customary boundaries and will make learning more efficient by incorporating technology. The system will grant the ability to access material at any moment and will make the process of evaluation and feedback as simple as possible.

3.1.2 Intended Users

- Student
    - Students enrolled in a school can use this system as per their school policies.
    - They can access the learning contents shared with them by their respective teachers.
    - They can get feedback and grades on the assessments they had taken with the help of system.
- Teacher
    - Teachers of schools can access the system as per the school policies.
    - They can create the learning contents and share that to particular class or students.
    - They can give feedback and grade the assessments given to a class or students.
- School
    - Schools can create different programs and classes.
    - They can add teachers/students within their learning management system and can handle all the details.
- Admin
    - Manage different schools and all the data.

3.1.3 Functional Requirements

Few of the functional requirements are mentioned below:

- User Login/Registration
    - Students/Teachers can login/register to the system as per their school policies
- View Dashboard
    - Students/Teachers should be able to view the dashboard after login
- Programs
    - Students/Teachers should be able to browse the programs they are registered with
    - Programs should be added as per the need
    - Users should be able to search different programs based on various filters available
- View Grades
    - Students should be able to view their grades after assessment completion
    - Teacher should be able to score the assessment after student submission
    - Teacher should be able to submit the score to student.
- View Report
    - Students should be able to see their mark reports of different assessments they had given
    - Teachers should be able to view the reports of their classes and their students.
- Assessment
    - Students should be able to take the assessment given by their teachers
    - Teacher should be able to create the assessment and also view the assessment submitted by the students
- Library
    - Students/Teachers should be able to explore the different library items available.
    - New content should be able to be added by the teacher
- Schools/Classes management
    - The admin should be able to manage the school/organization

- School admins should be able to manage the data regarding teachers/students and their classes.

3.1.4 Non-Functional Requirements

- Availability
  - The website should be available to all the students and teachers 24*7
- Reliability
  - The system must track requests and manage logs
- Security
  - The system must provide configurable role-based authentication.
- Capacity
  - The website should handle large number of users at a particular time and should not be crashed
- Data integrity
  - Data should be accurate across all database tables
- Usability
  - The website should be responsive and should be able to load successfully in Chrome, Firefox and Safari.
  - The website should be responsive with tablets.

3.2 STUDY OF THE PROPOSED SYSTEM

3.2.1 Phase 1 - Exposing all required endpoints via rbs-assessment-services

- Adding new endpoints to rbs-assessment-service (just by proxying)
- Create new rbs service called Mastery Service. This would expose mastery endpoints by proxying requests to Realize monolithic
- After end of phase 1
  - All required endpoints by consumers are proxied to legacy assessment service and Realize core
  - RBS Assessment Service and Mastery Service is ready to get consumed by all BFF consumers with all required endpoints.

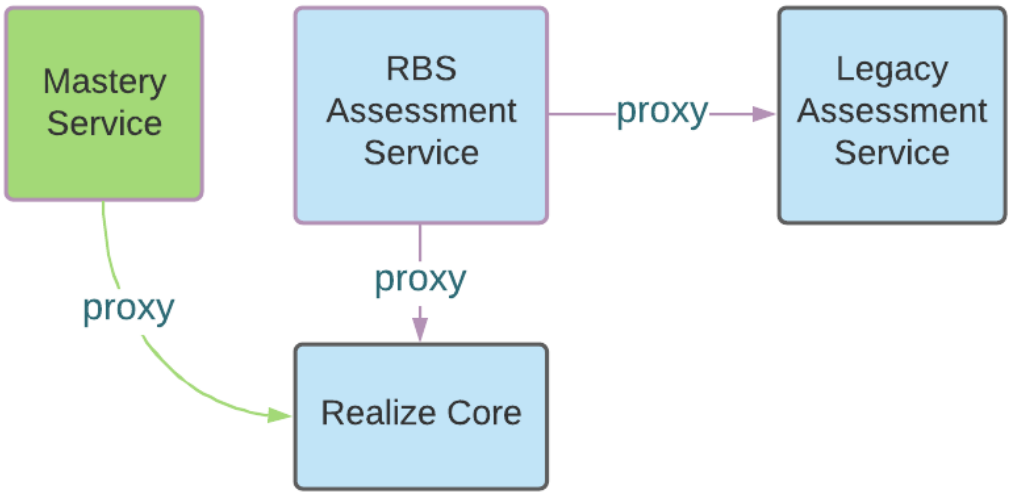


Fig. 3.2.1 Phase I

3.2.2 Phase 2 - Cutoff Realize dependencies

- Migrate business logic from Realize to rbs-assessment-service and mastery service
- Migrate tables from Realize to Assessment DB and Mastery DB
- After end of phase 2
    - Connection between Realize core to Legacy assessment service is completely cut-off
    - All requests to Legacy Assessment Service are routed via RBS Assessment Service
    - Mastery data is fully managed by Mastery Service

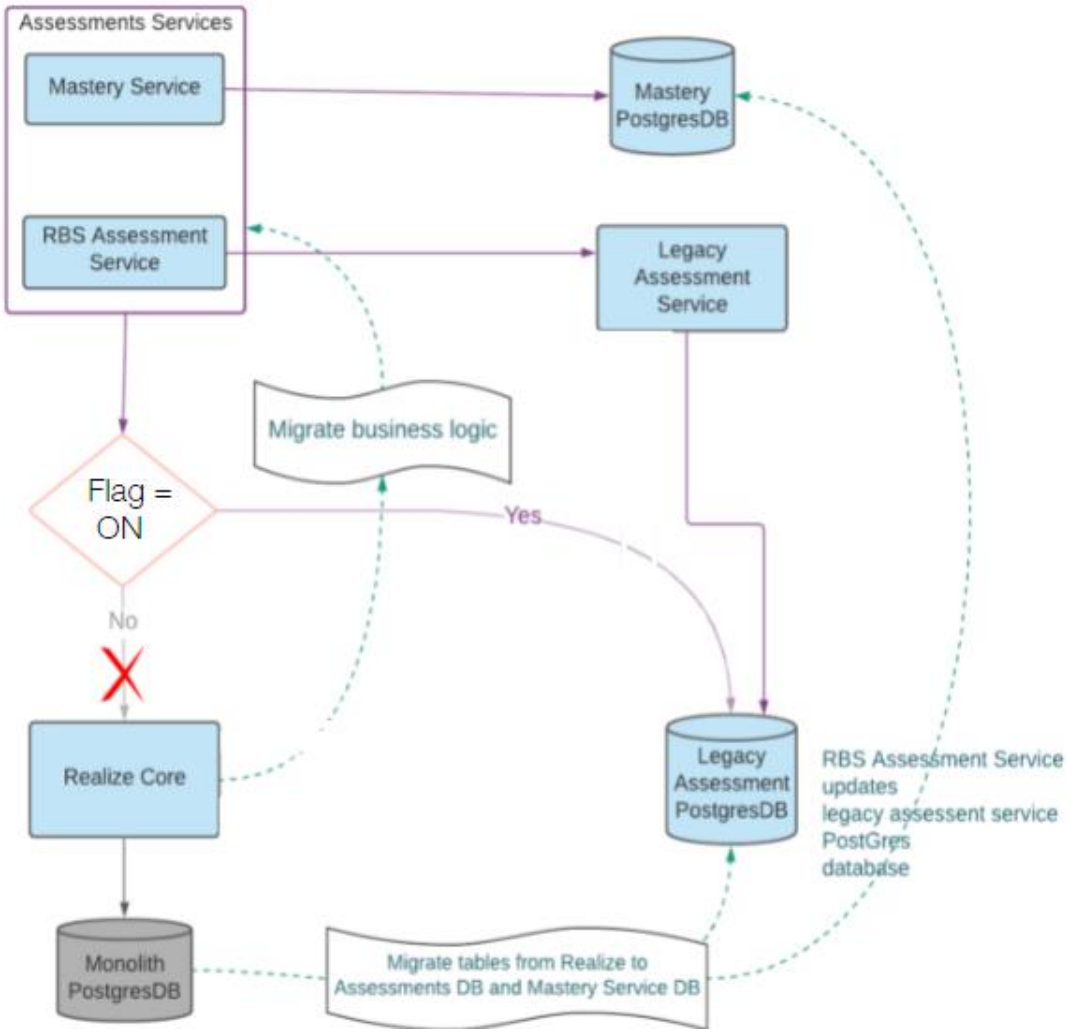


Fig. 3.2.2 Phase II

3.2.3 Phase 3 - Cutoff Legacy Assessment Service dependencies

- Migrate business logic from legacy assessment service to rbs-assessment-service
- RBS Assessment Service directly updates the Postgres database
- No changes required in Mastery Service
- After end of phase 3
    - All business logic is inside RBS Assessment Service
    - Legacy Assessment Service is completely shut down

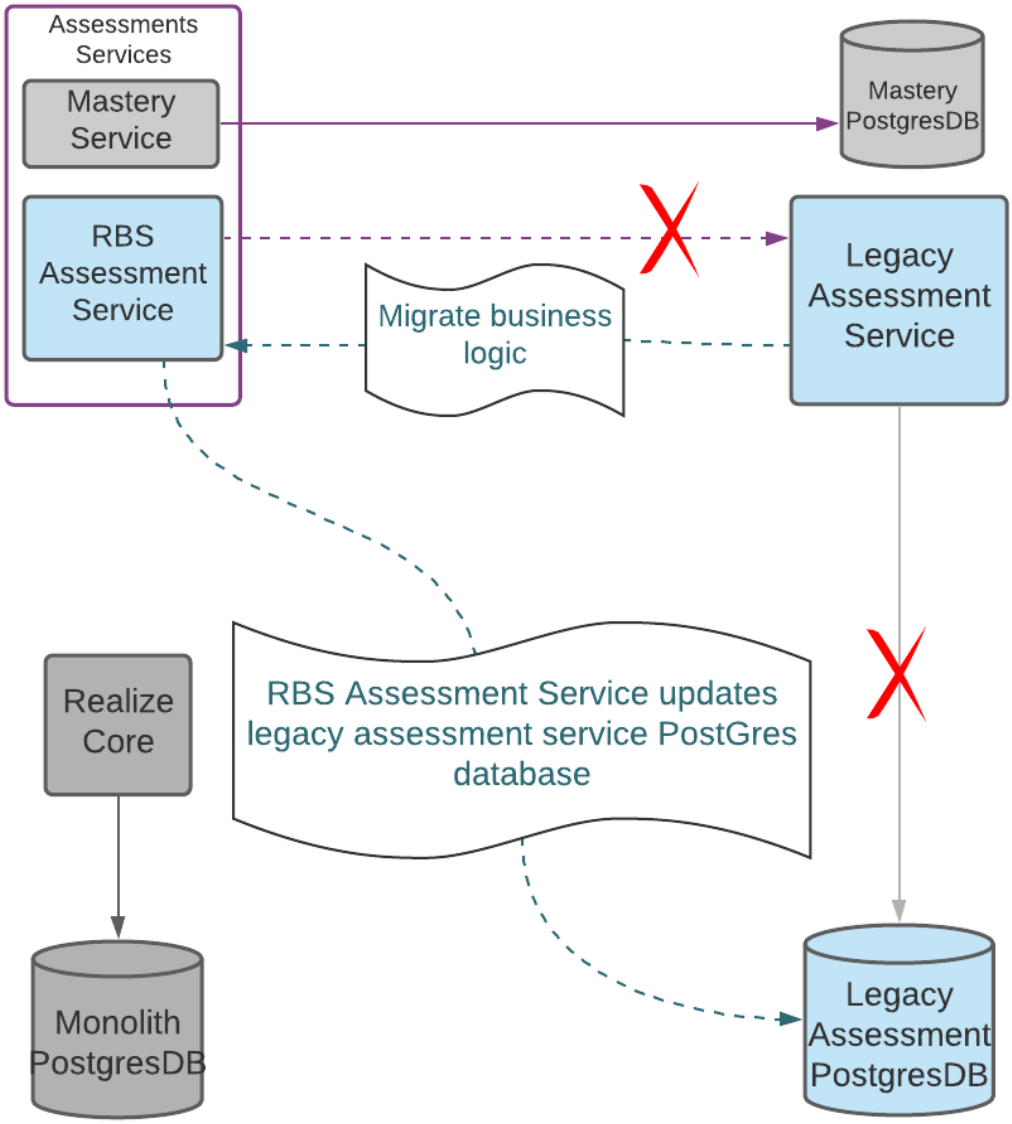


Fig. 3.2.3 Phase III

3.3 DESIGN AND TEST STEPS / CRITERIA

To create the new APIs, following are the steps that have been followed:

- Remove proxy from existing endpoints or expose new endpoints and implement logic in rbs-assessment-service.
- Add unit test cases for these endpoints.

- Manually create/edit local Postgres DB for Postman testing
- Update Swagger.yaml file for these endpoints
- Update RBS-Contract-Document
- Deploy the branch with Jenkins and verify on Nightly Realize; Mark as RFV(Ready for Verification) for SQE

The newly developed APIs will follow the Spring Reactive Web Architecture (Webflux) instead of the traditional Spring Web based architecture. The advantage of using Spring Reactive Web Framework is that it is Non-blocking and Asynchronous while the traditional Web Framework is Blocking and Synchronous.

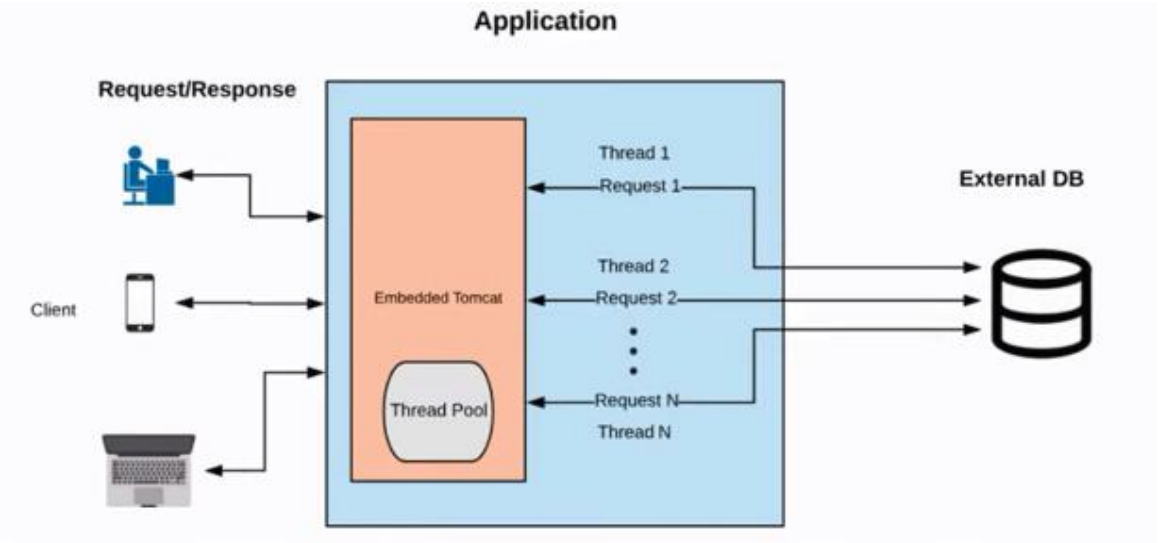


Fig. 3.3.1 Traditional Spring Web Architecture

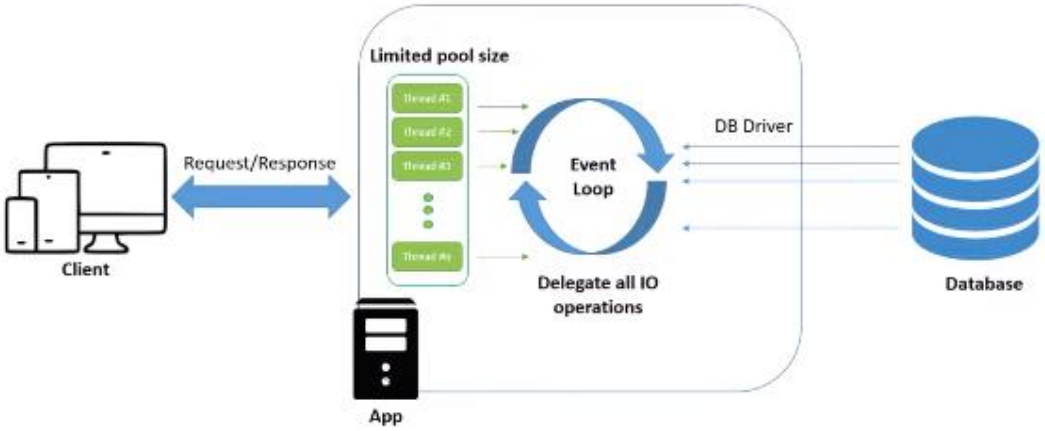


Fig. 3.3.2 Spring Reactive Webflux Architecture

3.4 TESTING PROCESS

3.4.1 Local Testing of API

Any new API which is created by developer, the first checking of the output is done locally. The database entries are created manually in our local database and check whether the output we get is expected or not. We also check whether the valid inputs are given or not and expect error on wrong inputs. Also verify that the mandatory items should be passed as input and check for errors in case where mandatory item is missing in the input request. The API inputs/outputs are mentioned in a Contract Document.

3.4.2 Testing of API on Nightly (Development) Server

Next, when the API is made after local testing, it is deployed to nightly server. We then run the APIs on this nightly server and checks whether the satisfactory result is coming or not; where the data is fetched from Nightly Database and not local database. The same conditions are checked here by the developer.

3.4.3 Testing of API by SQEs (Software Quality Engineers)

After the testing is done with Nightly server, SQEs create their test plans and check for the inputs/outputs as mentioned in the Contract Document. They check every condition and verify the output is correct as expected or else assign a rework in case of errors occurred. SQEs after thorough verifying the newly created APIs, they make Automation scripts which test these endpoints automatically. These Automation scripts are created to validate

the future changes in the endpoints.

Here is the general response status based on different scenarios list which is covered while manual/automation testing:

Response Status	**Scenario**
200	When the expected output is achieved
400	When there is a bad request in input
401	Unauthorized request where token or auth missing in header
403	Forbidden in case where some API is only allowed by teacher
404	Not found error
500	Any Internal Server Error

Table 3.4 General Response Status

CHAPTER 4: RESULTS / TASKS COMPLETED

4.1 API IMPLEMENTATION

4.1.1 Api Design Pattern

Developed the new RESTful APIs with Spring Reactive Web (Webflux) following functional programming methodology.

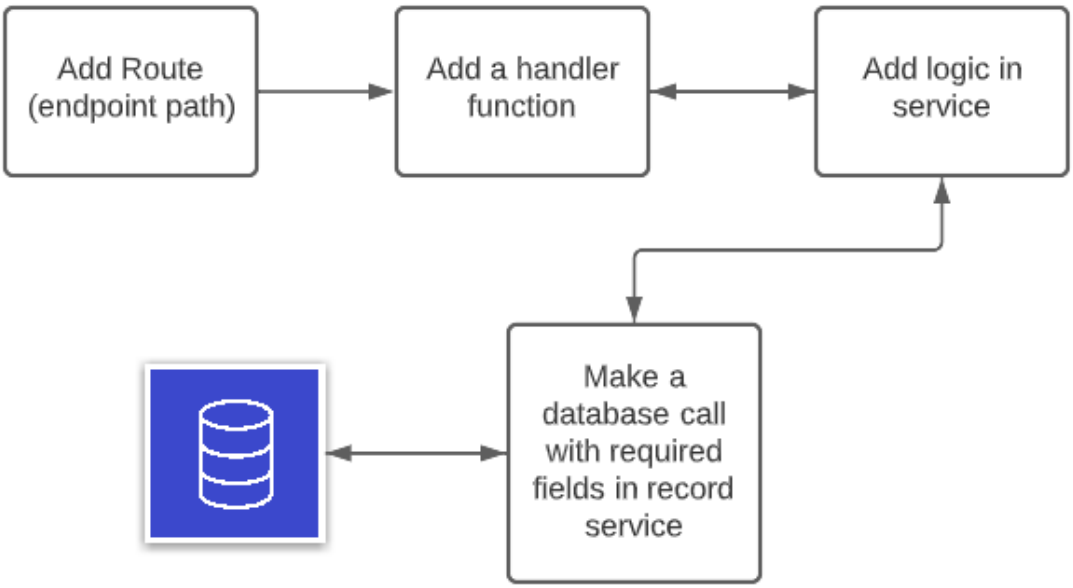


Fig 4.1.1 API Design

4.1.2 Apis Implemented/Modified

Method names and Routes:

- setImmutable- PUT /assessments/{assessmentId}/mutable
- getExternalQuestionMapping -

GET /v1/rbs/assessments/{assessmentId}/externalQuestions

- createAssessmentSession - POST v1/rbs/assessments/session
- needsManualScoring-

GET /assessments/{assessmentSessionId}/needsManualScoring

- getFlagByAssessmentSessionId-

GET /assessments/{assessmentSessionId}/getFlagByAssessmentSessionId

- getUserAssignmentIdByAssessmentSessionId-

GET:

/assessments/{assessmentSessionId}/getUserAssignmentIdByAssessmentSessionId

- getAssessmentSessionIds-

POST: /assessments/{assessmentId}/assessmentSessions

- getAssessmentScoreDetailsForSessionIds-

POST: /assessments/score/details

- getAssessmentFromAssessmentId-

GET: /assessments/{assessmentId}

- getAssessmentSessionIDDateByClass-

POST: /assessments/class/{classId}/{assignmentId}

- getLastAssessmentSessionIdByAssignmentUserId

GET: v2/rbs/assessment/userAssignments/{userAssignmentId}/last

- getUserAssignmentIdsByAssessmentSessionIds-

POST: /v2/rbs/assessments/assessmentSessionIds/userAssignmentIds

- getUserAssignmentIdByAssessmentSessionId-

GET:
v2/rbs/assessments/assessmentSessions/{assessmentSessionId}/userAssignmentId

4.2 REALIZE LOCAL SETUP

Set up the main scripts in the local machine. For this, have to Dual-Boot the laptop and install Ubuntu for the same. Next, have to set up the main project. It was quite challenging to set up this project because the build time took too long (~3 hours), faced many errors, resolved them, and took proper care for the dependencies to be added.
Also, there was a need to manage different Java versions as per the different services made (Java 8 & Java 11). For this, set up Jenv to manage different java versions. Additionally, added/modified the existing dependencies as per the need.

4.3 EMIT THE SIMPLE NOTIFICATION SERVICE(SNS) EVENT

Learned how the SNS/SQS works in AWS. Implemented the SNS event at the time of MSDA assignment completion. Verified SNS event payload is matching with the requirements by adding an email subscription. Modified the confluence page documenting the event payload

Fig. 4.3 Sample SNS mail notification

4.4 SCORE TABLE SERVICE - UPGRADE SPRING BOOT VERSION

There was a need to upgrade the Spring boot Version to 2.5.12 due to a new vulnerability that occurred recently. Info: https://spring.io/blog/2022/03/31/spring-framework-rce-early-announcement. Successfully updated the Spring boot version along with adding/updating other dependencies and changes in some files.

4.5 WORKED ON THE MAINTENANCE/BUG TICKETS

Maintenance tickets are the tasks basically related to queries obtained from school/parents/teachers/students.

Steps Followed:

- First verified whether the issue still exists via impersonating the user.
- If it exists, then reproduce the data in the prod/nightly environment.
- Took a deep look at what is the actual root problem if the error exists in every environment and found a proper solution for that.
- Updated my findings on the tickets.
- Watched the logs with AWS CloudWatch and DataDog to get the actual reason for errors.

Completed a data fix that happened due to Database Inconsistency.

Debugged a case where Submit button was not visible in the case of TestNav assessments.

4.6 TEST COVERAGE

Added test cases in Rbs-Assessment-service, Rbs-mastery-service & Score-table-service and moved the code coverage to >90% in Rbs-mastery and Score-table services.

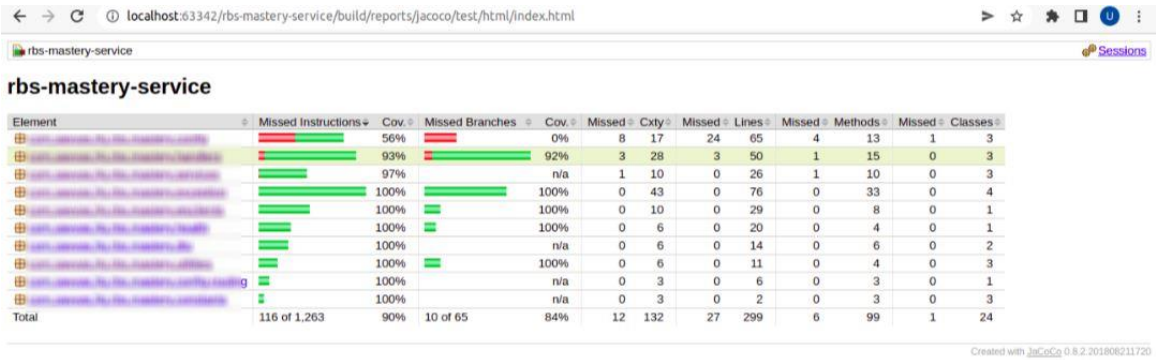


Fig. 4.6 Jacoco Report for Rbs-Mastery-Service

4.7 REMOVAL OF PAF DEPENDENCIES

Researched a bit on removing the client calls of PAF services for a few of the endpoints from RBS-Assessment-Service. Debugged and fixed/added a few testcases in different files which were failing after removing the paf calls.

CHAPTER 5: CONCLUSION

- Worked on creating new microservices from existing monolith architecture by carving out the necessary business logic
- Worked on the logic for data migration or implementing REST calls while developing the new microservices
- Developed the new RESTful APIs with Spring Reactive Web (Webflux) following functional programming methodology
- Implemented SNS emission on specific events
- Improved overall test coverage to >90% in different microservices
- Helped team on finding the root cause for few bugs in the prod environment
- Worked with Postgres and DynamoDb databases
- Monitored logs on different microservices with Datadog and Cloudwatch
- Gained knowledge on working with Jenkins and various AWS services

REFERENCES

[1] “Spring Boot,” Spring. [Online]. Available: https://spring.io/projects/spring-boot.

[2] “Guide to spring 5 webflux,” *Baeldung*, 02-May-2022. [Online]. Available: https://www.baeldung.com/spring-webflux.

[3] *5. messaging*. [Online]. Available: https://cloud.spring.io/spring-cloud-static/spring-cloud-aws/2.0.0.RELEASE/multi/multi__messaging.html.

[4] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, “From monolithic systems to Microservices: An assessment framework,” Information and Software Technology, vol. 137. Elsevier BV, p. 106600, Sep. 2021. doi: 10.1016/j.infsof.2021.106600.