



LearnZilla

B.Tech Project

Information and Communication Technology, Semester 8

Mentored by: Prof. Sanjay Chaudhary sanjay.chaudhary@ahduni.edu.in

25.02.2021

GROUP MEMBERS:

Shyam Patel	AU1741030	shyam.p@ahduni.edu.in
Jay Shah	AU1741084	jay.s@ahduni.edu.in
Shantanu Sheth	AU1741088	shantanu.s@ahduni.edu.in

Abstract

The demand and utilization of learning platforms are growing very fast everyday. Online Learning meets the thirst for information and delivers the online content to the learner at any place, at any time and at any age through a wide variety of online solutions compared to conventional learning systems. The designing and development of an online learning platform is the key part as it reflects on the usage of the system. Various existing online learning platforms are described and compared. The proposed system is designed using efficient System architecture and various software engineering models, programming methods and database models. The system is focused around several features like Learning Management, Evaluation Management, Content management.

Table of Contents

Abstract	1
Table of Contents	2
Introduction	4
Description	4
Objectives/Deliverables	4
System Architecture (Project Design)	5
Architecture Selection	5
Microservices	5
Great Architecture	5
Why microservices are better:	5
Challenges for Microservices:	5
Microservices-Database Architecture	6
Our Microservice Architecture:	6
Backend Architecture	6
Architecture Diagram	7
Authentication as a Microservice	7
JWT	7
Refresh Tokens	8
Diagram	8
Clean Architecture for forums/comments Microservice	9
Class Diagram	9
API Gateway	12
Express Gateway Core	12
How our Gateway works:	13
Frontend Architecture	14
Client:	14
Use Case Diagram:	15
Development	16
List of Modules and Programs	16
Frontend	16
Components:	16
API Gateway	16
Config:	16
Server:	16

Middleware:	16
Backend Microservices	16
Feedback Service	16
User-profile Service	16
Forums Service	16
IELTS Service	17
GRE Service	17
Software Platform	17
System Specification:	17
Frontend:	17
Backend:	17
Tools and Technologies used for Development:	18
Testing & Evaluation Plans	18
Testing Implemented	18
Automated API testing integrated with GitHub Project	18
Testing and Evaluation Planned	18
Current Status	18
Backend Development:	18
Frontend Development:	19
Timeline to complete remaining tasks	19
Future Direction	20
References	20

Introduction

Online Learning is a mere term used to refer to a type of learning in which, through the use of online technology, the teacher and student are separated by space or time where the distance between the two are bridged. Many technologies are currently being used in online learning like:

- Adaptive Assessment exams
- Virtual Classrooms
- Storage for Teaching Materials
- Virtual meeting platforms
- Discussion boards

Every year lots of students are preparing for their masters in foreign universities and for that they have to prepare for competitive exams like GRE and IELTS. Many students don't know where to start and prepare for the exams. Keeping this thought in mind, we came up with the idea of building a platform where anyone can just login in one click and start to prepare for their career. Other than learning through videos, users will also be able to give adaptive assessment exams for a better learning curve.

Description

The proposed product here is a web application which lets you learn for the competitive exams with interactive videos and prepare for the exams through different exams. The software will be having a Login/Signup functionality followed by browsing through two different courses GRE and IELTS. The GRE section comprises of verbal and quantitative modules. Some practice tests and vocabulary section for practice. The IELTS section comprises of Listening, Reading, Writing and Speaking modules. Some practice tests and vocabulary section for practice.

The proposed product is beneficial for students and also for those who want to pursue higher education in foreign countries.

Objectives/Deliverables

The introducing product LearnZilla is a platform where only trusted sources are available. The major task of this system is to streamline the learning process in a hassle free manner. The system is going to be used by varied types of customers i.e. Customer base: contains all the whole front facing of the business idea.

The end users are the customers who can use a complete functional web-app. Customers can Login/Signup in the system and browse through courses. Customers can go to any particular course section and learn through interactive videos, give the tests in which they are weak. There is one additional feature of adaptive test. The objective of this adaptive test feature is to give insights to any particular user in which section the user is weak and in which section the user needs more practice. The system also has a feature of progress analysis, in which the user can analyse its progress after each test. To run all of these smoothly, the system will be developed with specific UI.

System Architecture (Project Design)

Architecture Selection

There are typically 2 types of architecture that are mainly used for designing a Web App: Monolithic Architecture and Microservice Architecture. A **monolithic architecture** is the traditional unified model for the design of a software program. **Monolithic**, in this context, means composed all in one piece. According to the Cambridge dictionary, the adjective **monolithic** also means both too large and unable to be changed. A monolithic App is all or nothing -- a big, giant thing that can't fit in a container. **Microservices** - also known as the **microservice** architecture - is an architectural style that structures an application as a collection of services that are. It is highly maintainable and testable, Loosely coupled, Independently deployable. It can be organised around business capabilities. Microservice app is continuously updating in parts.

Microservices

Great Architecture

- Scales Development Teams
- Delivers Quality
- Enable High performance/ low cost
- Supports future features naturally

Why microservices are better:

- Decades of best practice in modern web era
- Web-scale experiences (Amazon, Google, Netflix, etc)
- Cloud Native Technologies
 - Microservices as an individual processes
 - Processes encapsulated in containers
 - Containers orchestrated

The natural evolution of microservices are APIs to the "N" client - clients traditionally were built by the company, but now can be built by anyone.

Challenges for Microservices:

- Duplication of code among services (overcome by api gateway)
- Propagation of server implementation details to the client

And many more.

Microservices-Database Architecture

- Each Microservices application has its own dB
- No service is allowed to connect to other dB
- Other services use only the service Interface i.e. API + Events

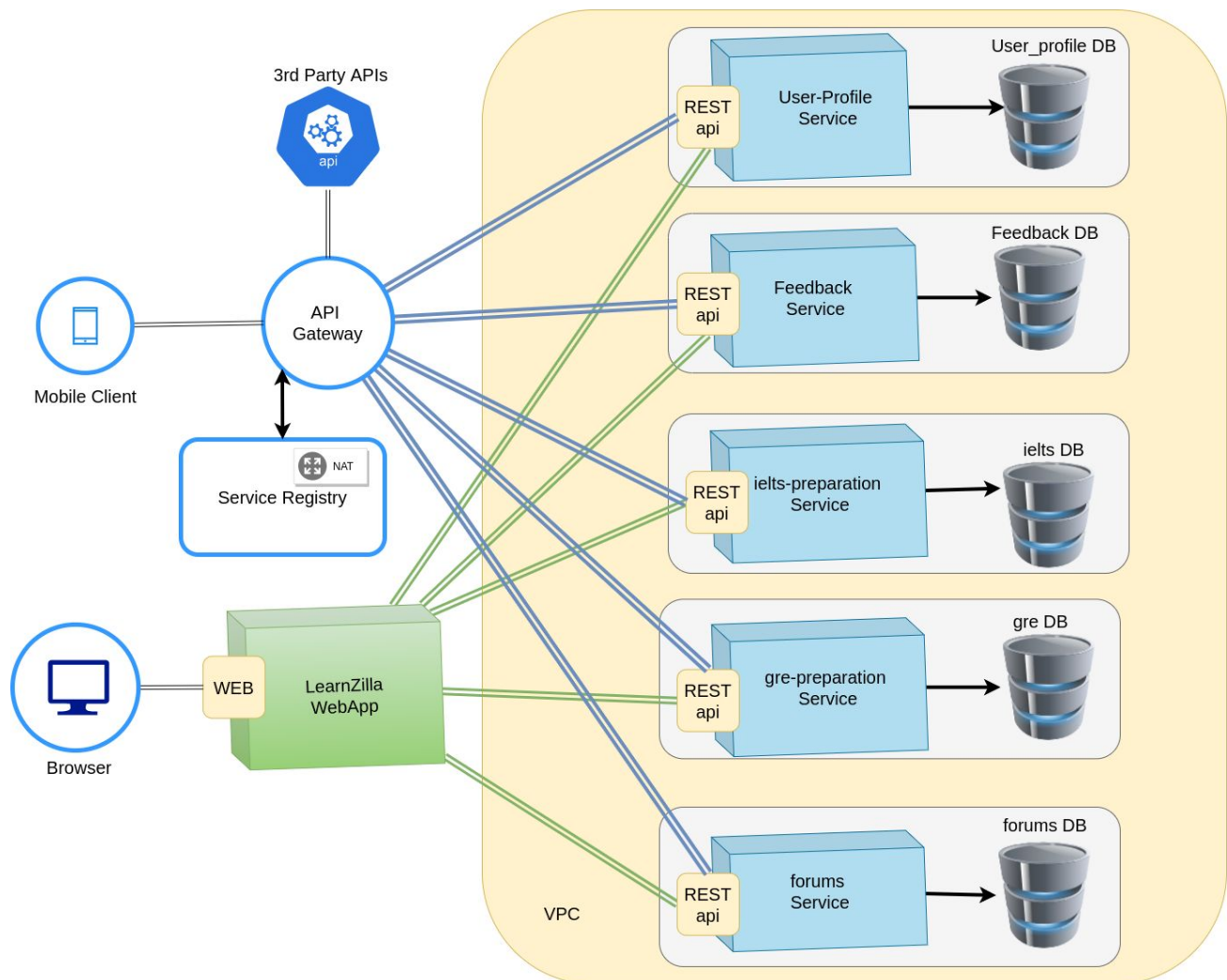
Our Microservice Architecture:

Our System is briefly divided into 3 main architectures namely Client-Side architecture i.e. Frontend architecture and Server-Side architecture i.e 1) API Gateway and Service Registry 2) Backend architecture for Microservices.

Backend Architecture

Basic Architecture of our backend can be seen in the below figure.

Architecture Diagram



As the Diagram describes we have configured 5 micro-services for now. All the services are isolated from each other such that no Microservice can operate on another service's database or interact with the other Microservice in any way. We use **an Express API Gateway** for exposure of microservice to external web and for the communication purpose between the services. There is a **Service Registry** which keeps track of which microservices are active on which ip addresses and ports. It also keeps track of things like if

any service is scaled horizontally or not, if yes then the range of ports that are used and if a service is down what to send through API gateway etc. All the client requests go through API Gateway in order to perform any business logic or operations from any service.

Authentication as a Microservice

JWT

- JSON web tokens
- Signed with public/private key pair
- Can be validated by other API without calling User-Profile API
- Contains roles

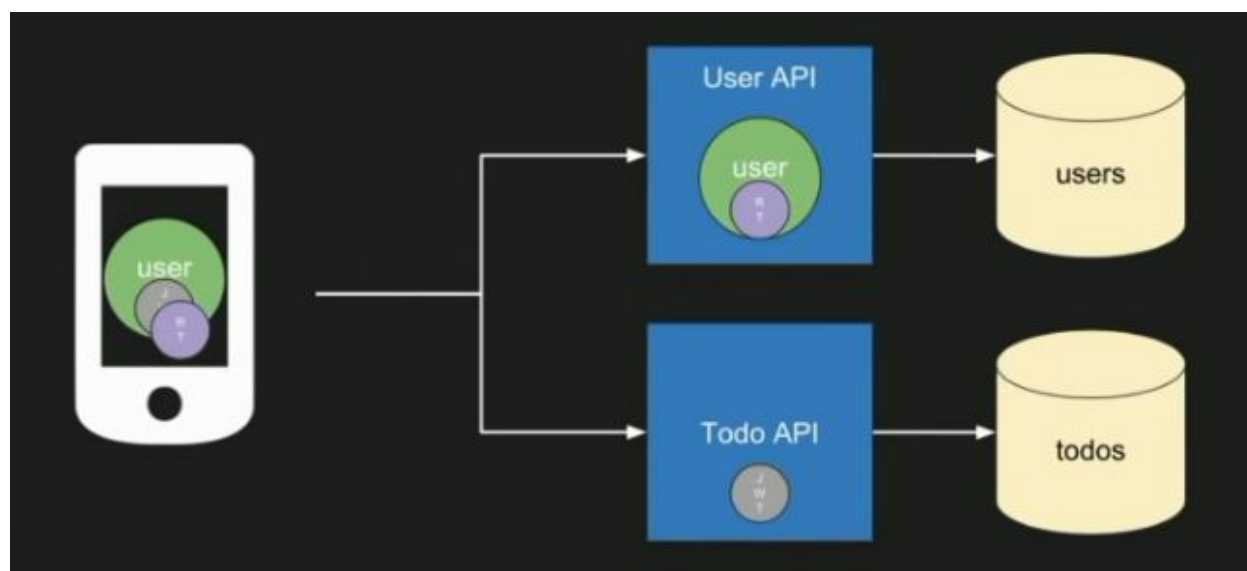
Refresh Tokens

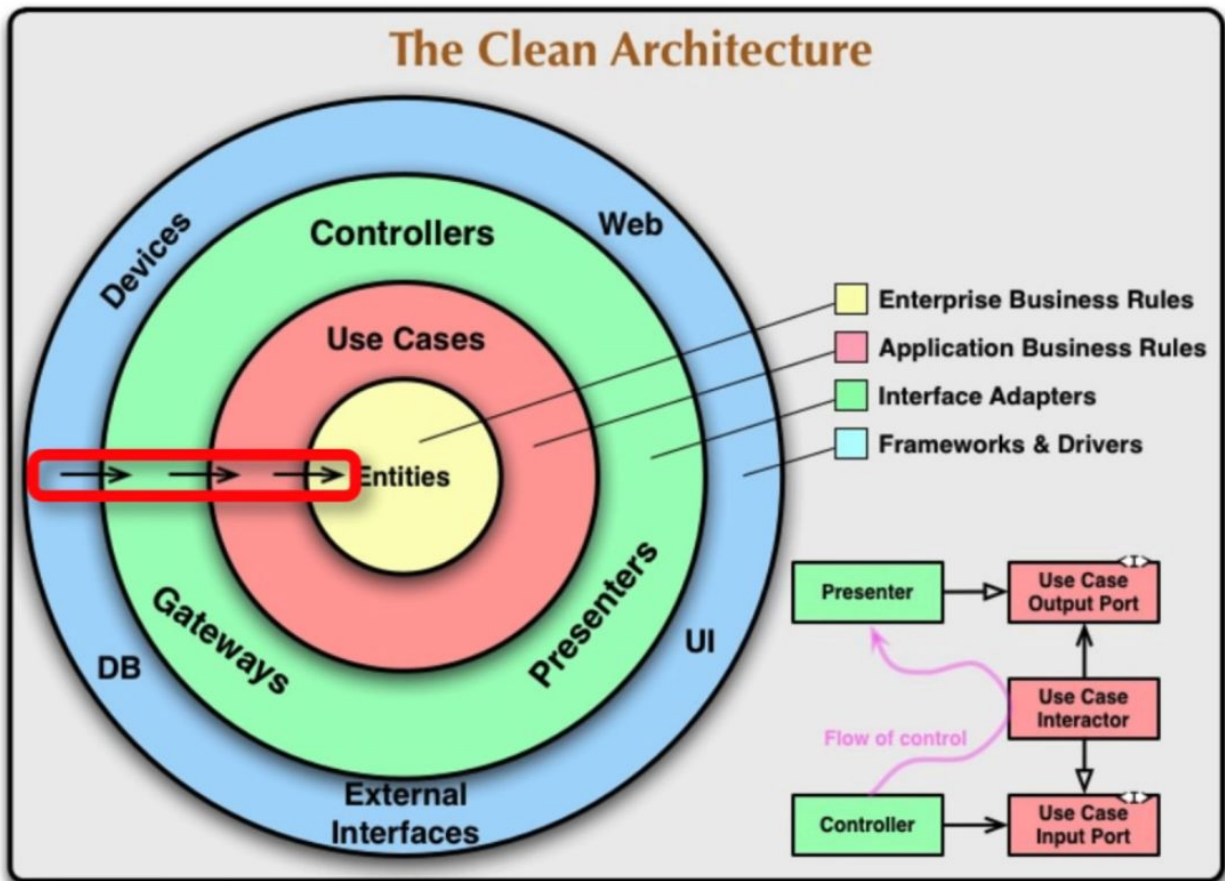
- JWTs are meant to be short lived (minutes)
- Refresh tokens are long lived (days or sometimes months)
- Refresh tokens can be used to create new JWTs

Here is how these are used:

As seen in the diagram below a User is assigned Refresh Token and JWT when they signup. Since JWTs are short lived we use Refresh tokens to assign new JWTs until a refresh token expires. Refresh tokens are generated every time a user logs in and deleted every time a user logs out.

Diagram





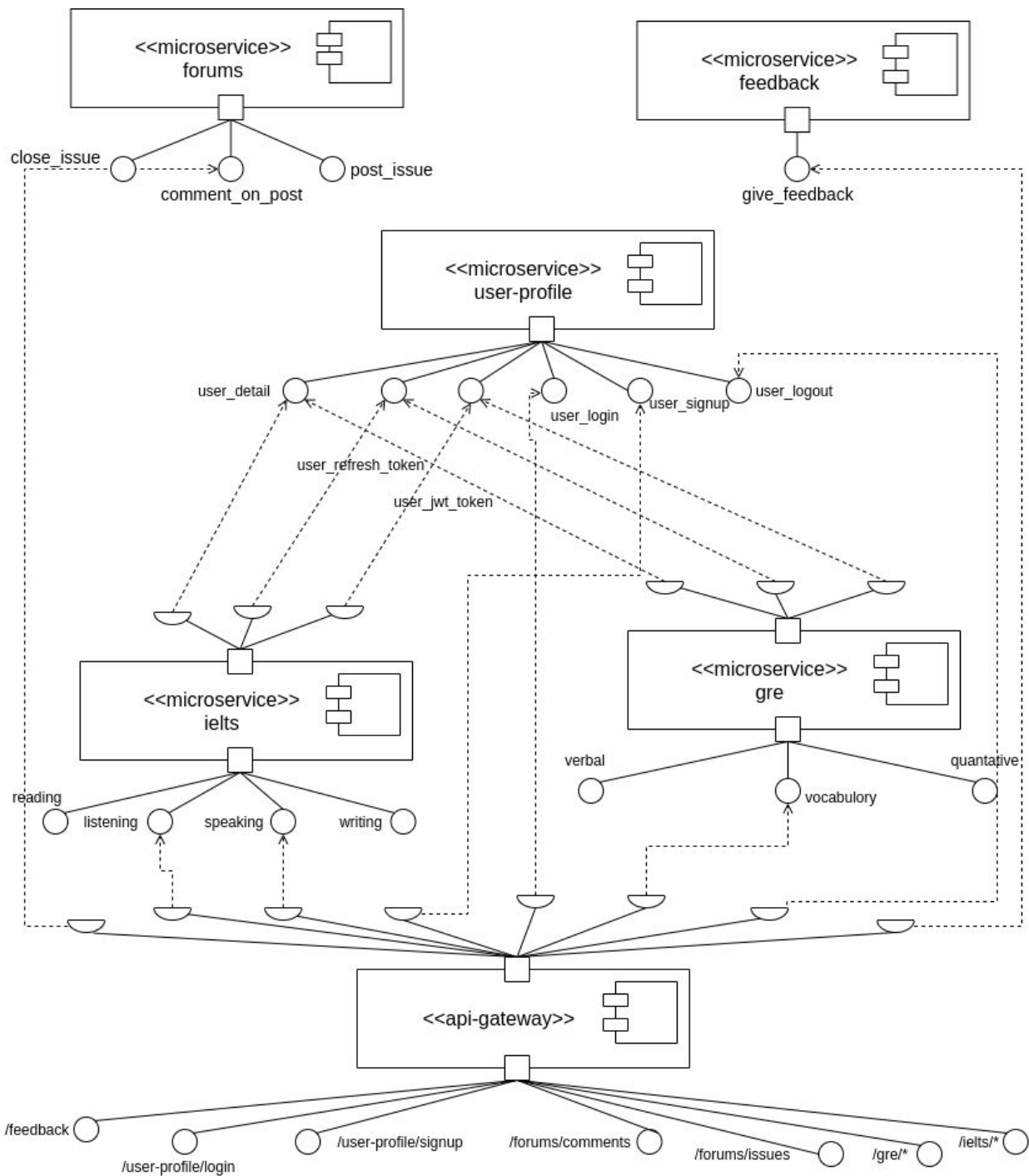
Class Diagram

Defining a class diagram was a tricky part in our project, Since we are using Microservices architecture and a NoSQL Database. We defined the following levels in relation to a typical SQL class/ER diagram.

Levels of Logical Views		Data Models		
		Entity Relationship	Relational	NoSQL
Level 1	Information Concerning Entities and Relationships	Entities, Relationships, Attributes ↓		
Level 2	Information Structure	Entity-Relationship Diagram ↓	3NF Relations <i>Decomposition Approach</i> ↑	
Level 3	Access Path Independent Data Structure	Table	Relations(Tables)	Key-Value , Document , Colum Family , Graph
Level 4	Access Path Dependent Data Structure			

Figure 1. Logical view of NoSQL Data Model

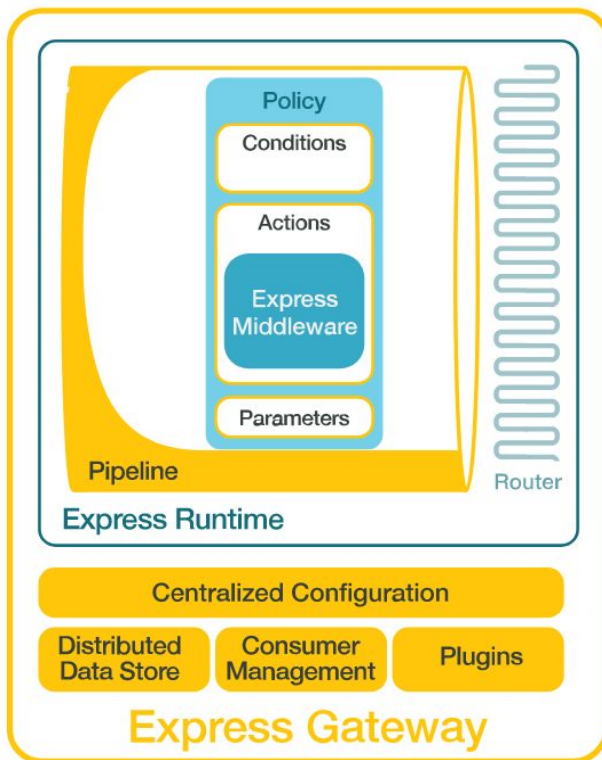
Our class diagram can be seen below.



API Gateway

We have used Express Gateway which is a flexible and efficient gateway built on top of express JS.

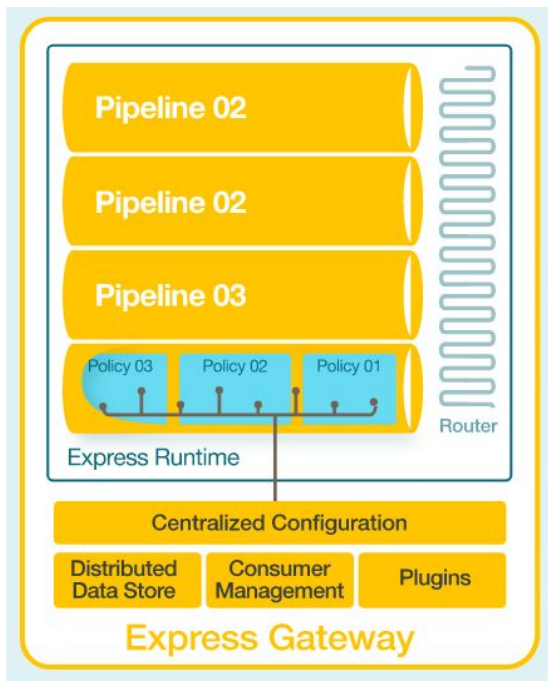
Express Gateway Core



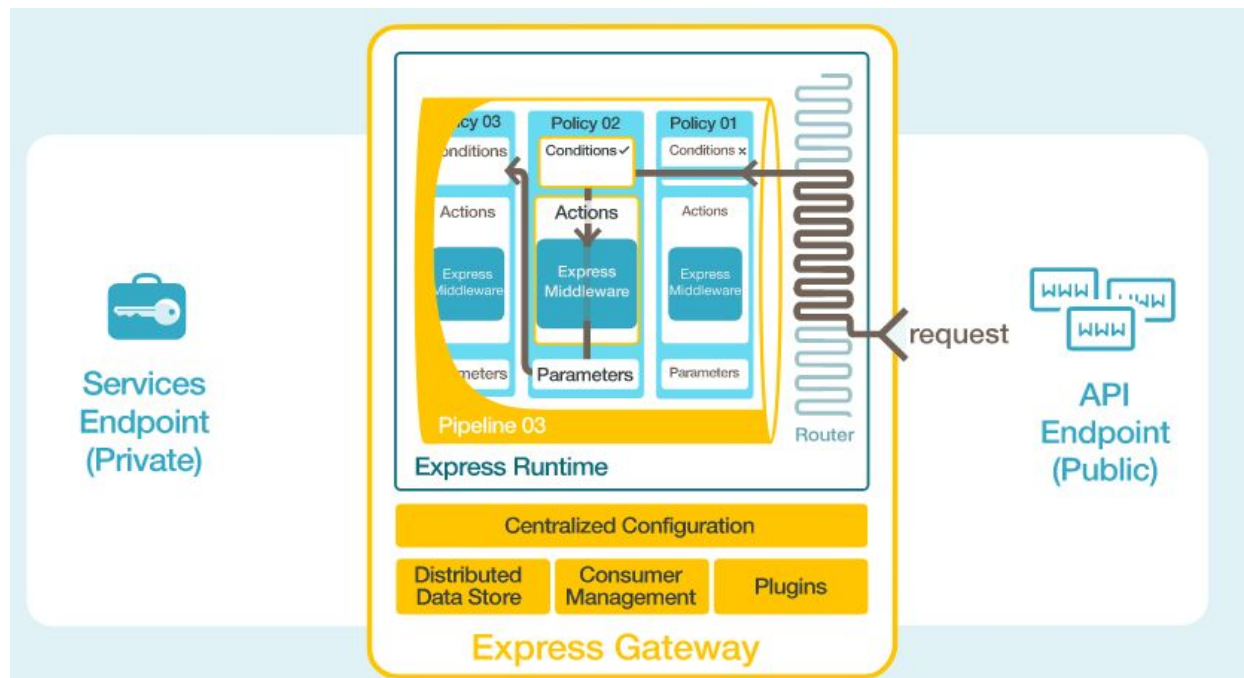
Express Gateway transforms Express into a dynamic runtime engine. For example - we can easily hardcode routes statically through Express' API. In Express Gateway, however, those values can continually change and are dynamically injected as parameters without having to alter the underlying code.

Essentially, all of the core components within Express Gateway make Express more dynamic and declarative.

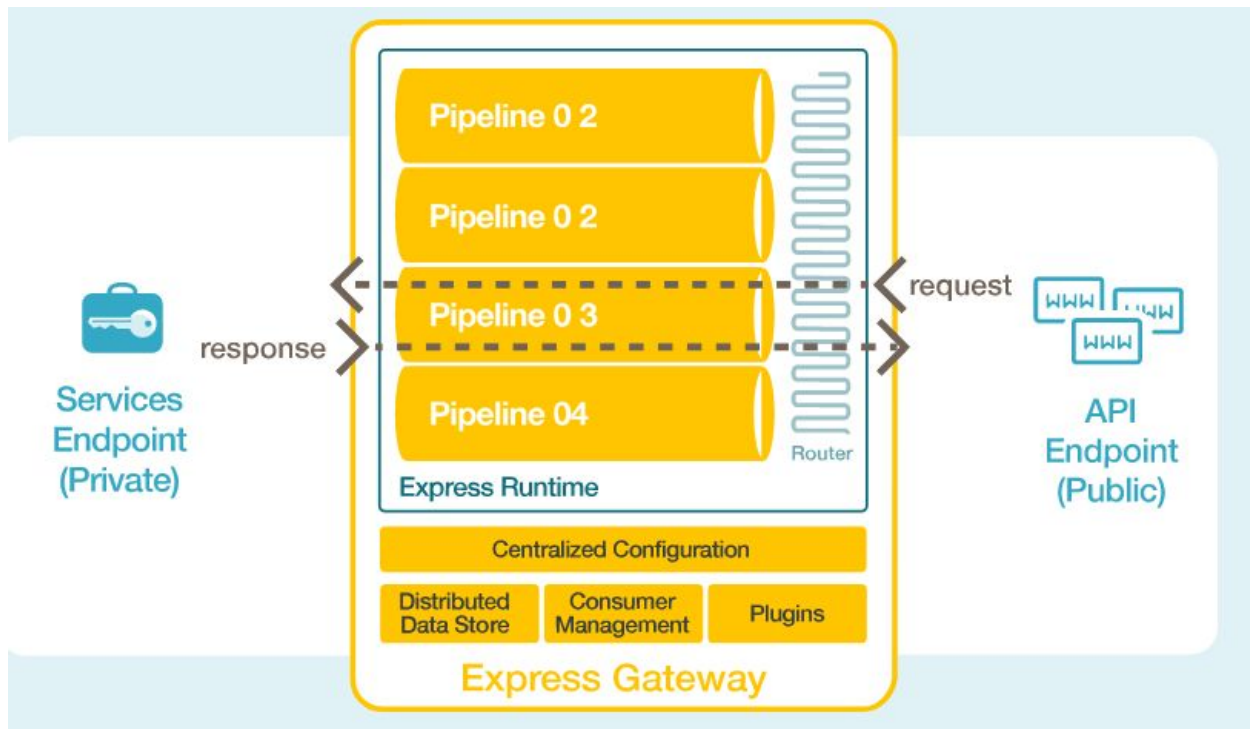
How our Gateway works:



- Express Gateway YAML is read.
- Config parameters initialize Express and its middleware.
- A request comes in through an API endpoint and is received by the Router.
- The Router connects the API endpoint to a pipeline.
- Within each pipeline is a set of policies.



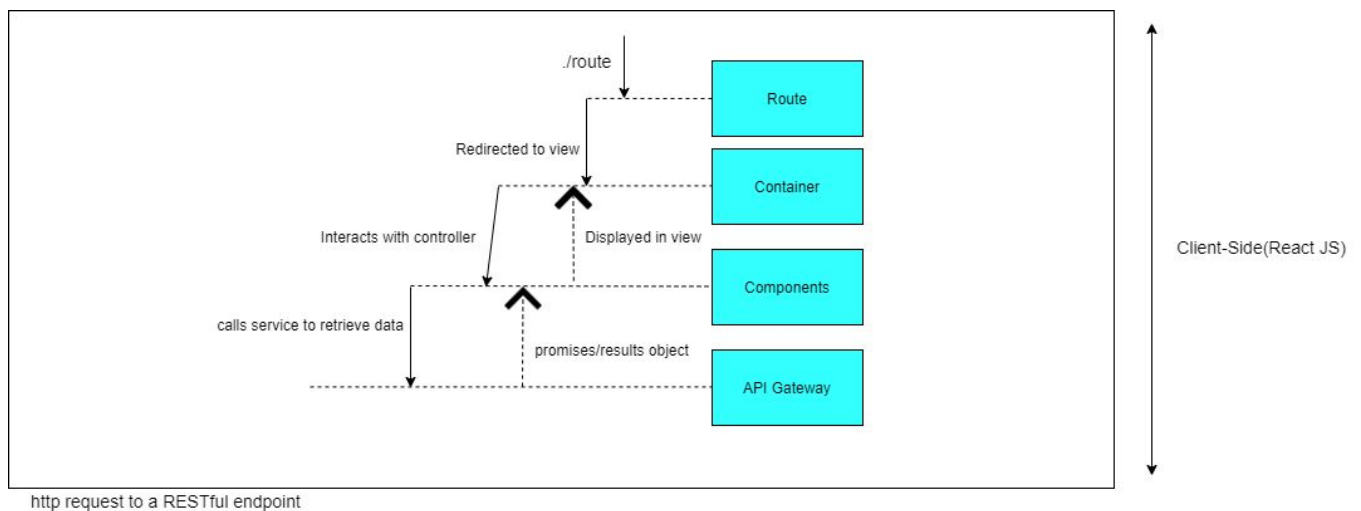
Each policy has a set of conditions dynamically injected from the Centralized Config. If a condition is met its corresponding action is taken by the Express middleware at the heart of the Express Gateway policy. The request is passed onto the next policy within the pipeline and the process repeats for every policy within the pipeline.



After all of the policies in the pipeline are passed, the request is usually proxied to an underlying microservice which applies the business logic and sends out the response to the API consumer

Policies can act on that response before it's forwarded to the API consumer.

Frontend Architecture

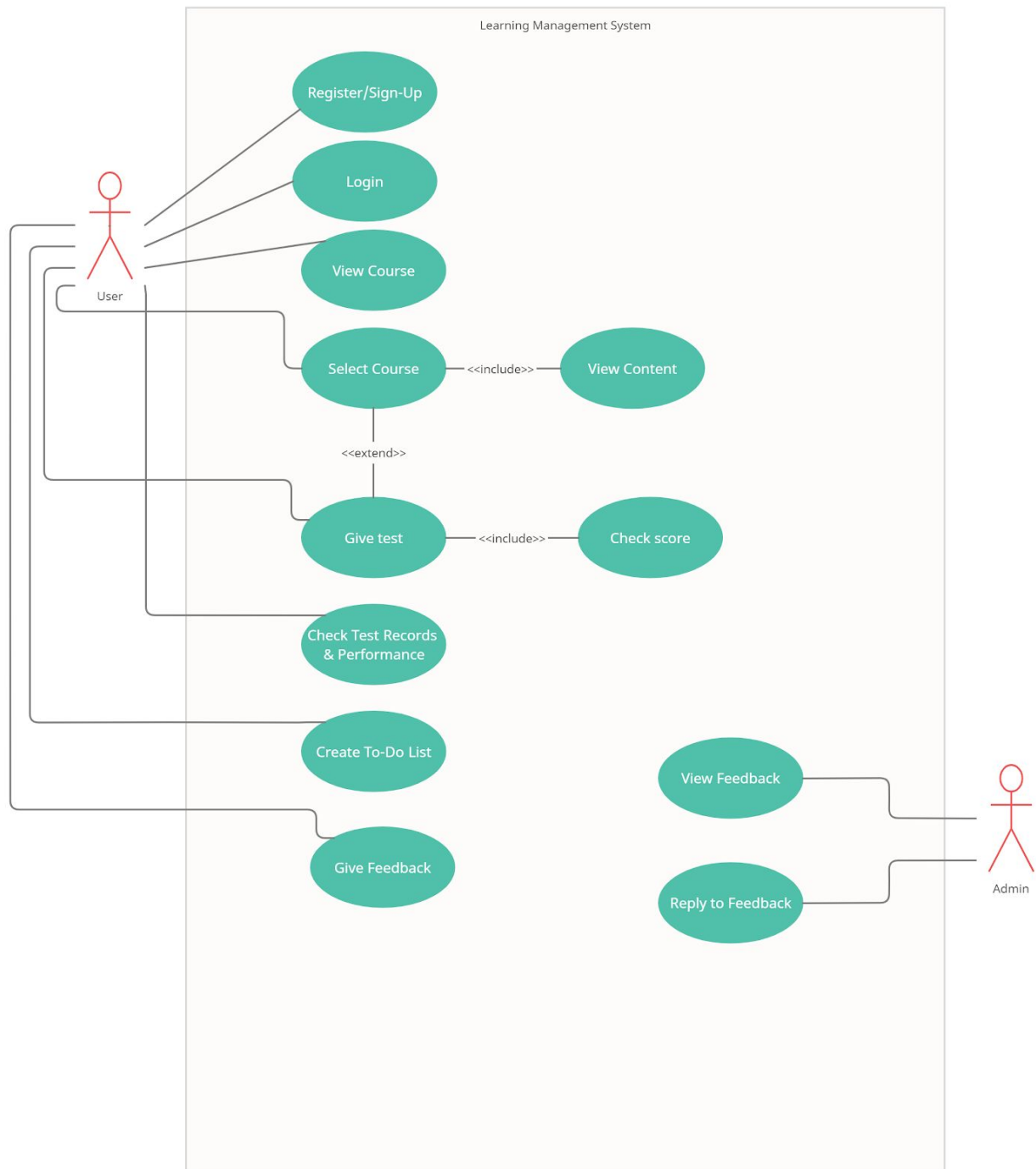


Client:

The flow starts from frontend. Users visit the website on a particular **route**. On hitting that route, a main **container** will be called which contains the logic of individual

components. Based on individual components, API calls will be made through the **API Gateway**.

Use Case Diagram:



Development

List of Modules and Programs

Frontend

Components:

- Login
- Signup
- Sidebar
- Navbar
- Main App
- Cards

API Gateway

Config:

- gateway.config.yml
- system.config.yml

Server:

- Server.js

Middleware:

- Validation.js
- Authentication.js

Backend Microservices

Feedback Service

- Feedback.js (Business Logic)
- Feedback (DataModel)

User-profile Service

- Users.js
- Validation.js
- Profile.js
- User (Data Model)

Forums Service

- Post.js
- Comments.js

- Post (Data Model)
- Comment (Data Model)

IELTS Service

- Reading.js
- Writing.js
- Speaking.js
- Listening.js
- Reading (Data Model)
- Writing (Data Model)
- Speaking (Data Model)
- Listening (Data Model)
- Helper.js

GRE Service

- Verbal.js
- Quantitative.js
- Vocabulary.js
- Verbal (Data Model)
- Quantitative (Data Model)
- Vocabulary (Data Model)
- Helper.js

Software Platform

System Specification:

Frontend:

- OS: Windows 10
- Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz
- CPU RAM: 16 GiB
- System Type: 64-bit, x64 based processor

Backend:

- OS: Arch Linux
- Kernel: x86_64 Linux 5.11.1-arch1-1
- Shell: bash 5.1.4
- DE: KDE 5.79.0 / Plasma 5.21.1
- WM: KWin
- CPU: Intel Core i7-7700HQ @ 8x 3.8GHz
- RAM: 16 GiB

Tools and Technologies used for Development:

- Visual Studio Code (latest version) for coding
- Github for Code Collaboration and version control
- Github Projects for task and milestone completion
- MongoDB Atlas for Database Collaboration
- Node & Express for Backend
- React for Frontend
- Express for API Gateway
- Figma for Prototyping

Testing & Evaluation Plans

Testing Implemented

Automated API testing integrated with GitHub Project

Testing and Evaluation Planned

Unit Testing:

- A level of software testing where individual units/components of the software are tested. The purpose is to validate that each individual unit performs as designed.

Integration Testing:

- Individual software modules are combined and tested as a group.

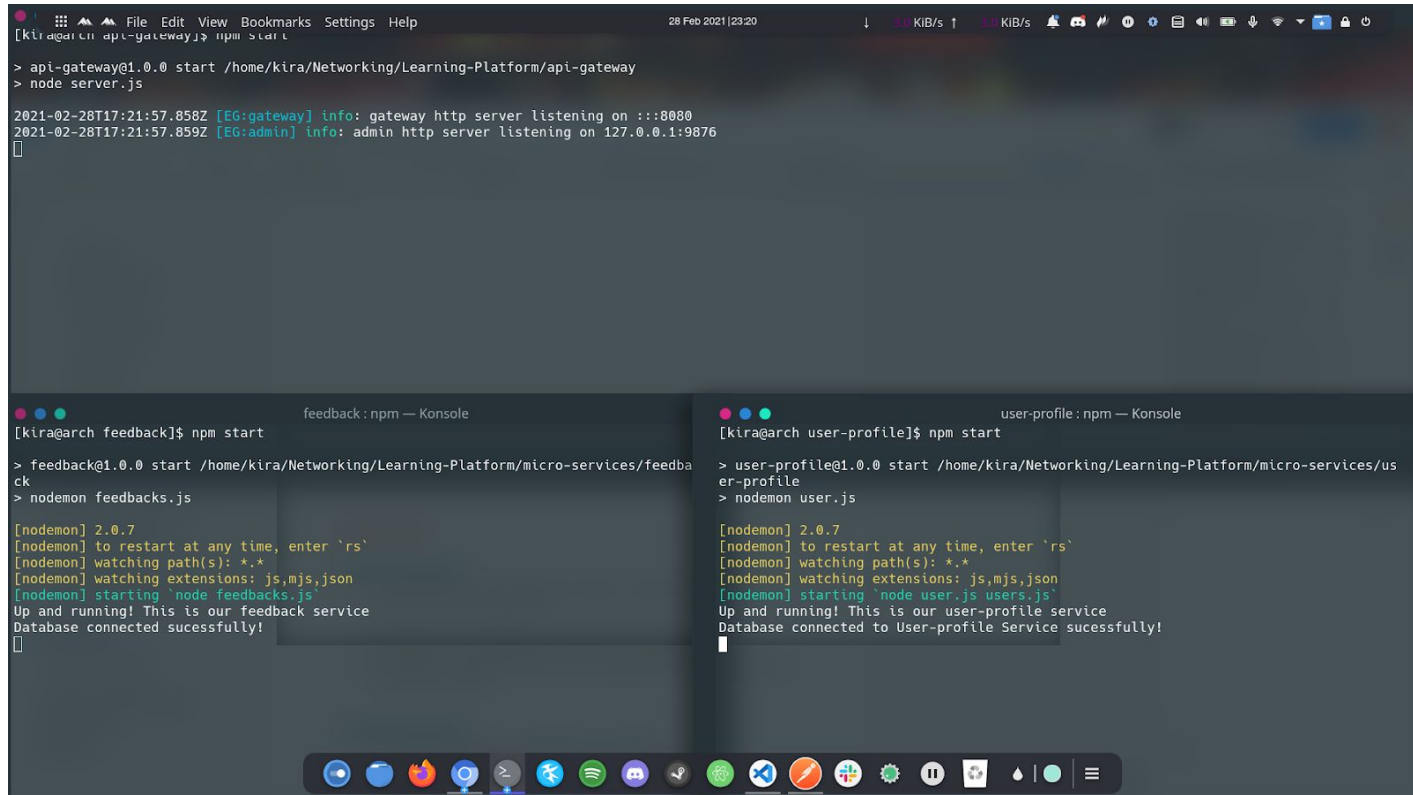
API Testing:

- All the APIs created for software under-tested.

Current Status

Backend Development:

- Image of our running API Gateway server and 2 microservices namely feedback and user-profile service



```
[kira@arch api-gateway]$ npm start
> api-gateway@1.0.0 start /home/kira/Networking/Learning-Platform/api-gateway
> node server.js

2021-02-28T17:21:57.858Z [EG:gateway] info: gateway http server listening on :::8080
2021-02-28T17:21:57.859Z [EG:admin] info: admin http server listening on 127.0.0.1:9876
[]
```

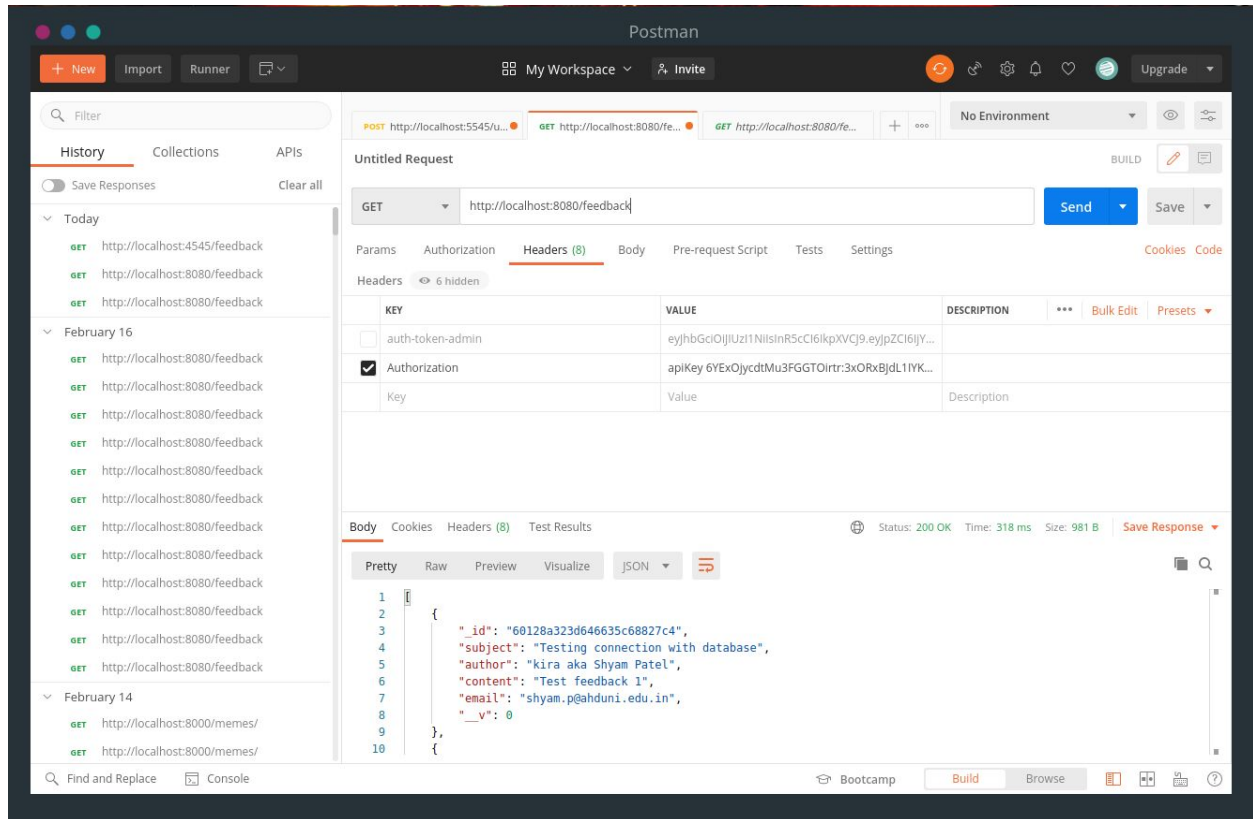
```
[kira@arch feedback]$ npm start
> feedback@1.0.0 start /home/kira/Networking/Learning-Platform/micro-services/feedback
> nodemon feedbacks.js

[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node feedbacks.js`
Up and running! This is our feedback service
Database connected successfully!
[]
```

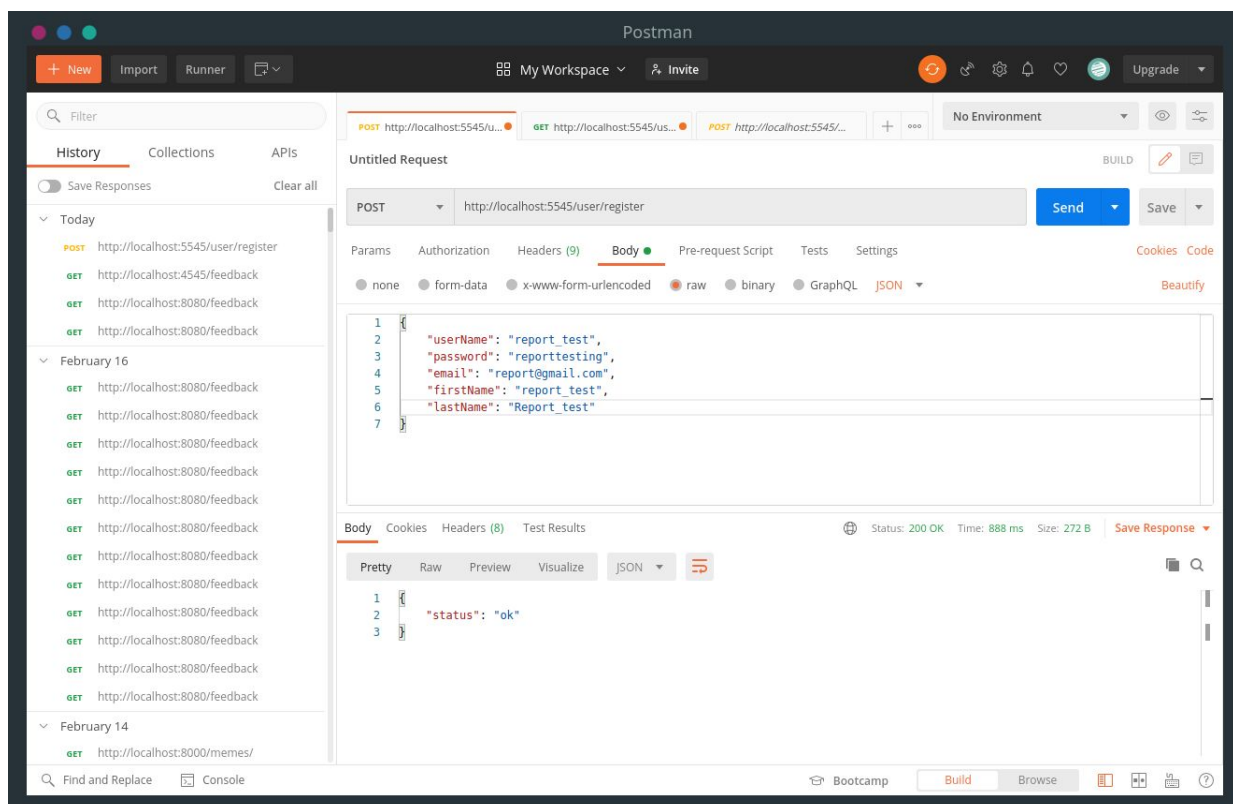
```
[kira@arch user-profile]$ npm start
> user-profile@1.0.0 start /home/kira/Networking/Learning-Platform/micro-services/user-profile
> nodemon user.js

[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node user.js users.js`
Up and running! This is our user-profile service
Database connected to User-profile Service successfully!
[]
```

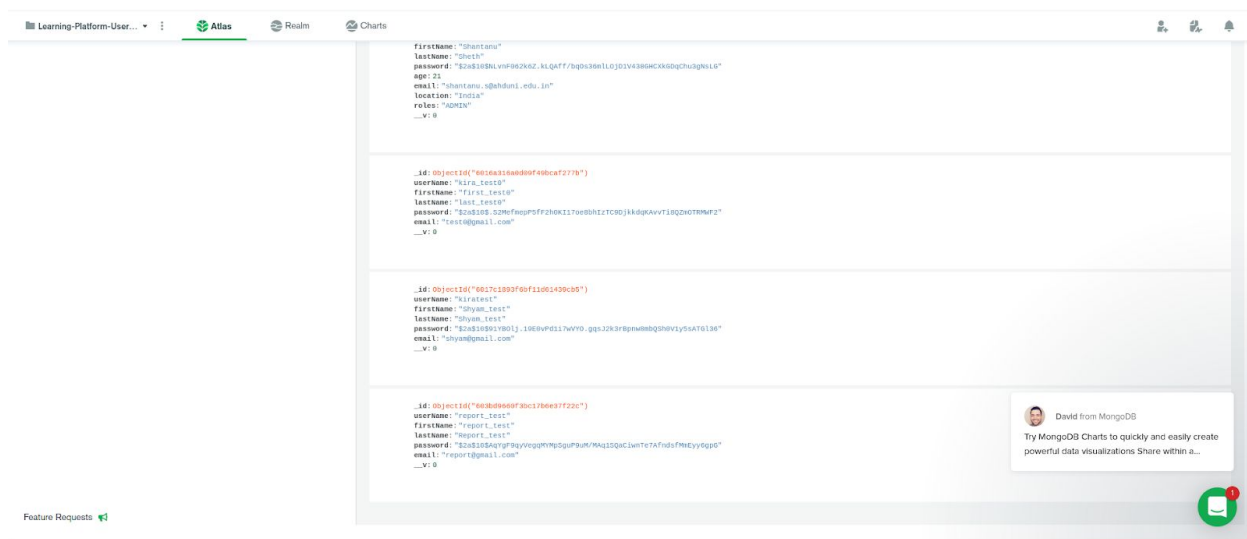
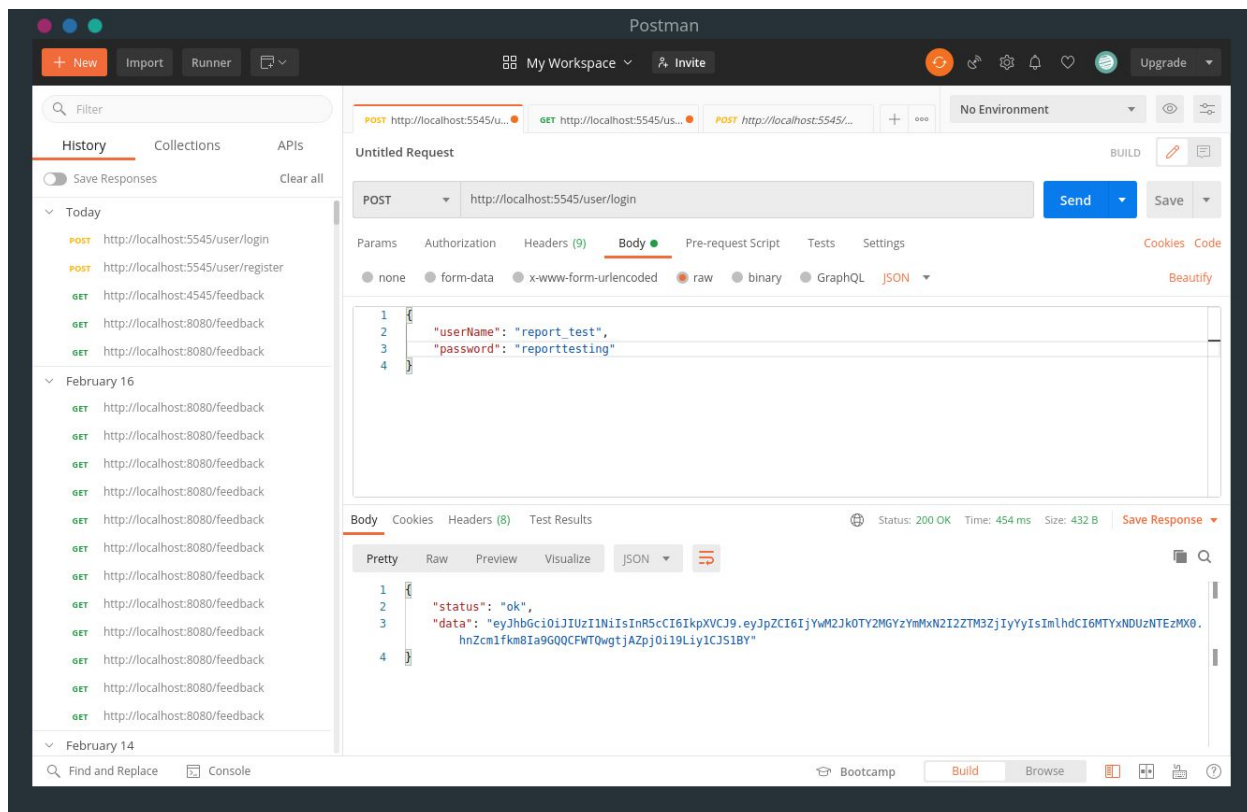
- Feedback Service is developed and its API is tested using Postman.



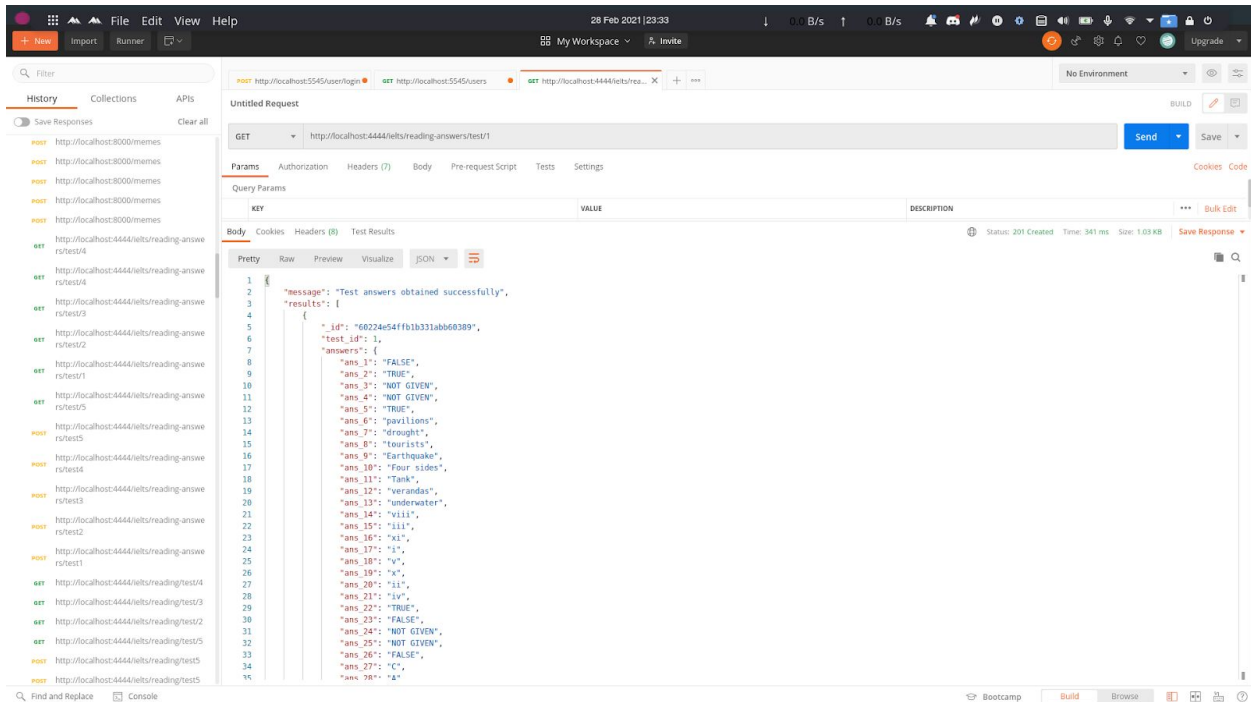
- User Profile service is developed and its API is tested using Postman.



- Authentication using JWT(Json Web Tokens) and password encryption is developed and tested in Postman.



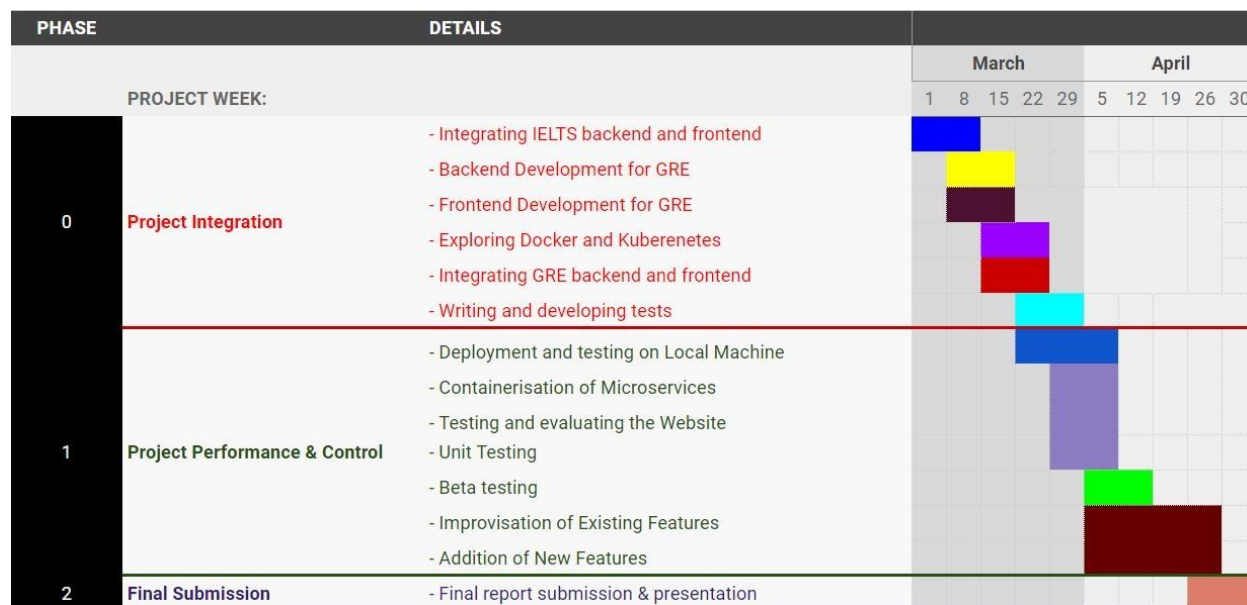
- Backend for IELTS (Listening, Reading & Writing) is completed and API's are successfully tested using Postman.



Frontend Development:

- Prototyping in Figma and mockup pages for user interface.
- Landing Page is developed along with Login and Registration Page.
- Feedback Page is developed.
- Logo of our platform is designed and is needed to be integrated on the landing page.
- Page of IELTS is under development and needs to be integrated with the backend for proper testing.

Timeline to complete remaining tasks



Future Direction

For BTP purpose we are focusing on a single aspect of Learning Management system i.e. Preparation for IELTS and GRE like Exams. For Future works we can make a mobile App for the same and also implement other aspects of Learning Management.

Hosting online classes, storage options for University and schools, Interactive web app to make learning fun for primary and secondary school students, implementing a hobbies based module that can teach you your favourite hobby for example chess, or correctly exercising or learning how to cook etc.

References

- Virtualization. (2012). Reliability and Availability of Cloud Computing, 16–28. doi: 10.1002/9781118393994.ch2
- A Practical Guide to Deploying Cloud-Based Video Services. (n.d.). Retrieved from https://pages.awscloud.com/GLOBAL_IND_practical_guide_whitepaper_2019.html
- “ReactJS Documentation.” React, <https://reactjs.org/docs/getting-started.html> Accessed 28 Feb. 2021.
- Node. Js. “Documentation.” Node.Js, 2015, <https://nodejs.org/en/docs>

- Alshuqayran, N., Ali, N., Evans, R.: A systematic mapping study in microservice architecture. In: Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on. pp. 44–51. IEEE (2016)
- Mario Villamizar, Oscar Garces, Harold Castro, Mauricio Verano, ´ Lorena Salamanca, Rubby Casallas, and Santiago Gil. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In Computing Col
- Matthias Vianden, Horst Lichter, and Andreas Steffens. Experience on a microservice-based reference architecture for measurement systems. In 2014 21st Asia-Pacific Software Engineering Conference, volume 1, pages 183–190. IEEE, 2014.
- Sudhir Tonse. Microservices at netflix - challenges of scale. <http://www.slideshare.net/stonse/microservices-at-netflix>, August 2014.