Real-Time Quiz application

1. Overview

Welcome to the Real-Time Quiz coding challenge! Your task is to create a technical solution for a real-time quiz feature for an English learning application. This feature will allow users to answer questions in real-time, compete with others, and see their scores updated live on a leaderboard.

2. Business solution

2.1. Requirement

- Users should be able to join a quiz session using a unique quiz Id (quiz No).
- Users' scores should be updated in real-time as they submit answers.
- A leaderboard should display the current standings of all participants in real-time.
- Quiz and questions can be easily configured.
- Users able to move/view to questions they have already submitted.
- Users can resume a quiz with validated constraints.
- The application has the ability to automatically submit questions when quiz time is expired.

Exceptional / assumption case:

> Users can resume a quiz session if the test is still valid.

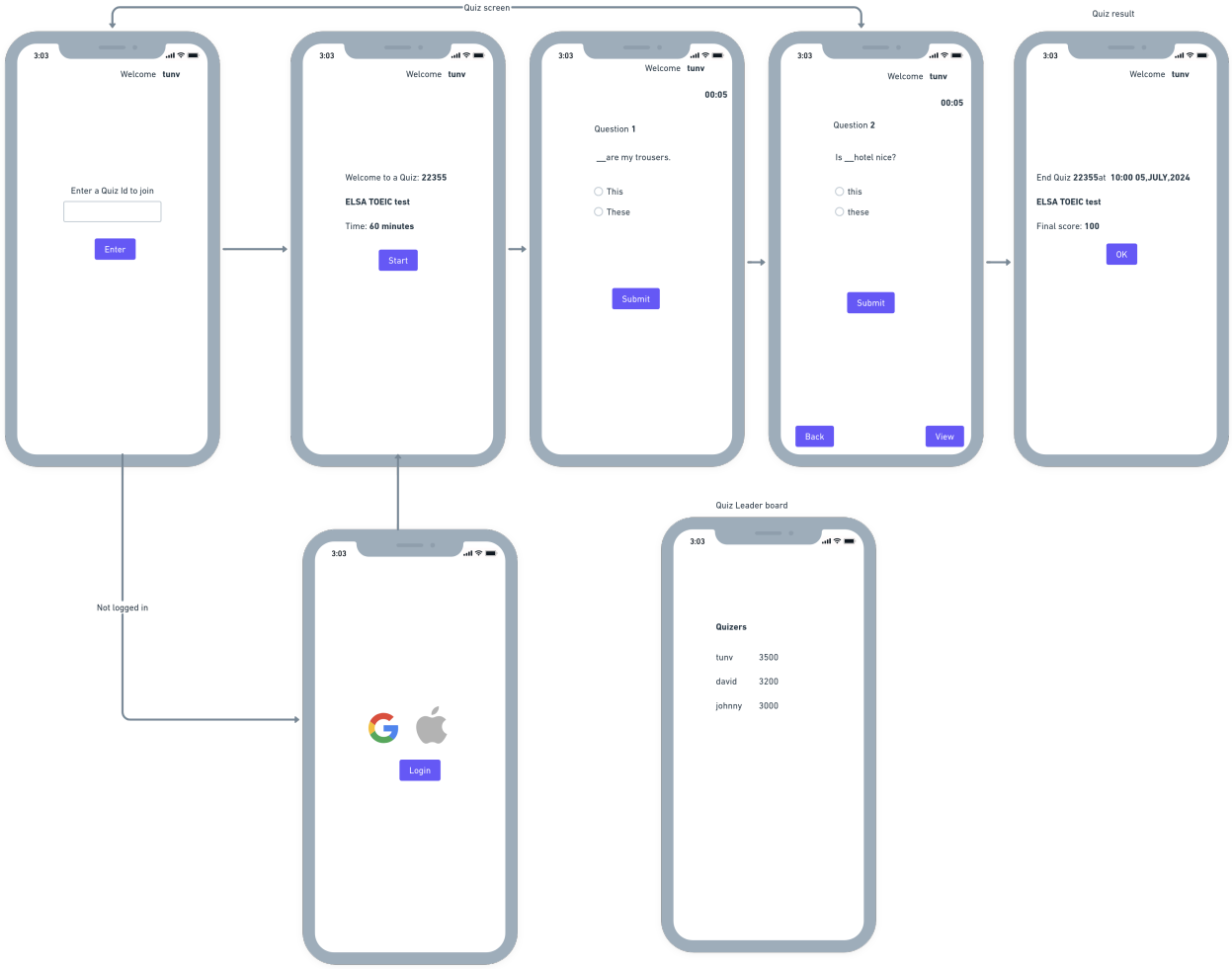
2.2. Non-functional requirements

- Traffic: 10 million active users per day, transactions per second (TPS): 100K
- **100K active quizzes in database.**
- Maximum file size: 5 MB for images, 10 MB for videos per quiz.

2.3. Business Flow

Assumes user has logged into the portal to do a quiz

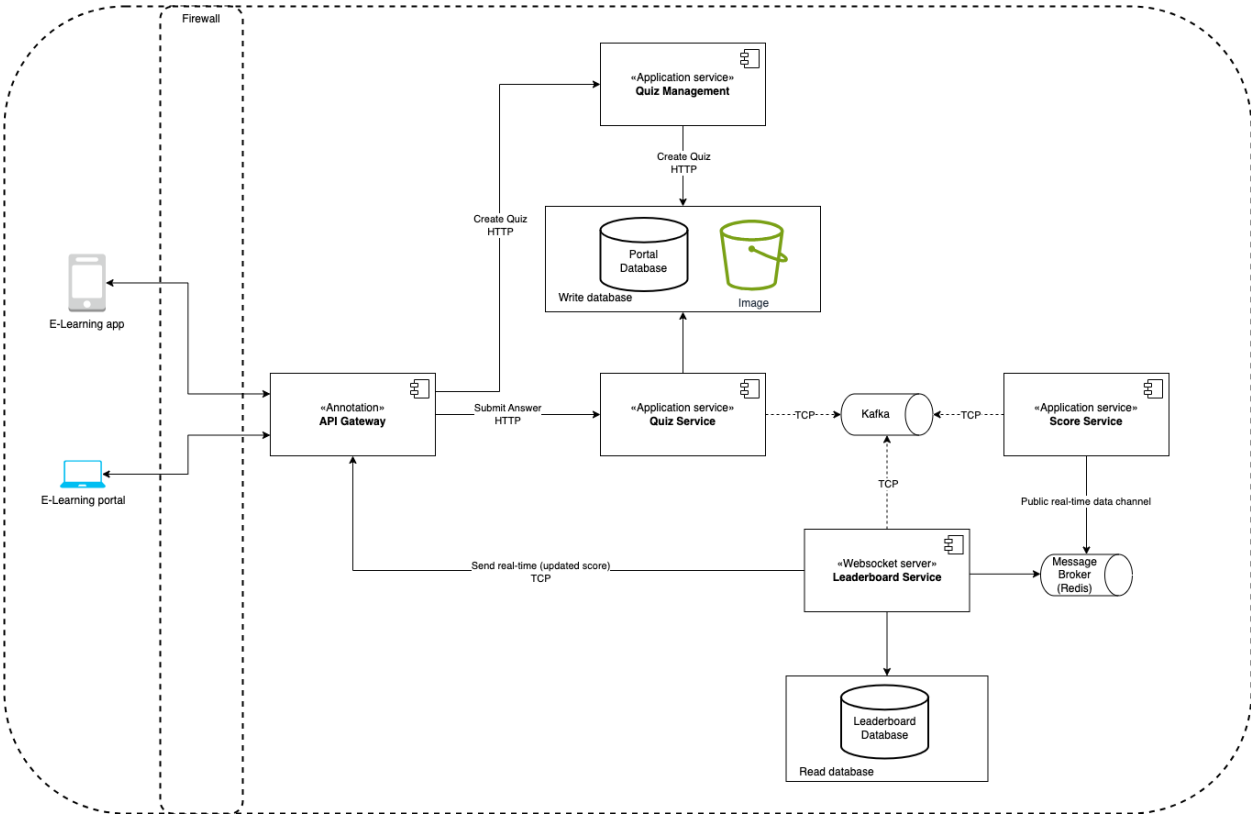
Front end will responsible for render question/answer



3. Technical solution

3.1. System context

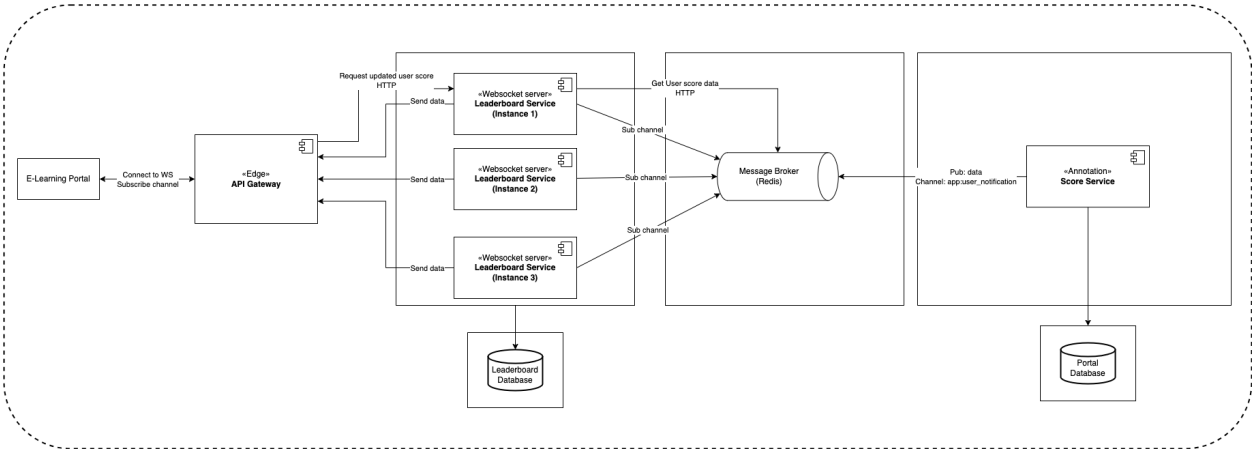
3.2. High Level - Logical Application Architecture



System Component	Description	Note	
API Gateway	The e-learning API gateway provides support for various features: - Security: Authentication and authorization - Data transformation - Rate limit - Request routing and load balancing - Monitoring and logging ...	- Java (version 21) - Spring boot, spring cloud Netflix, spring cloud stacks - JWT token cloud stack: - AWS API Gateway and AWS Cognito	
Quiz Service	**Real-time Quiz Participation**: This component is allowing users to be able to join a quiz session using a unique quiz ID	Java (version 21), spring boot, Kafka, AWS S3	
Score Service	**Real-time score updates**: This component is responsible for updating and syncing score when the user submit answers for a quiz	Java (version 21), spring boot, Kafka	
Leaderboard Service	**Real-time Leaderboard**: This component is responsible for the following tasks: - Web socket server: Use socket protocol - Displaying the current standings of all participants in real-time - Showing the highest user scores	Node JS, Web socket, Redis, Kafka	
Quiz Management	- Manage resource such as: Quiz, Question - Quiz/Question configuration	Java (version 21), spring boot, AWS S3	

E-learning app	It is web portal, a single-page web application (SPA) based front end application, mobile application (iOS, Android)	Javascript, typescript, React JS, Websocket Mobile native	
Monitors + Logging component			
	TBD		

Leaderboard server - Web socket Architecture on EKS cluster:



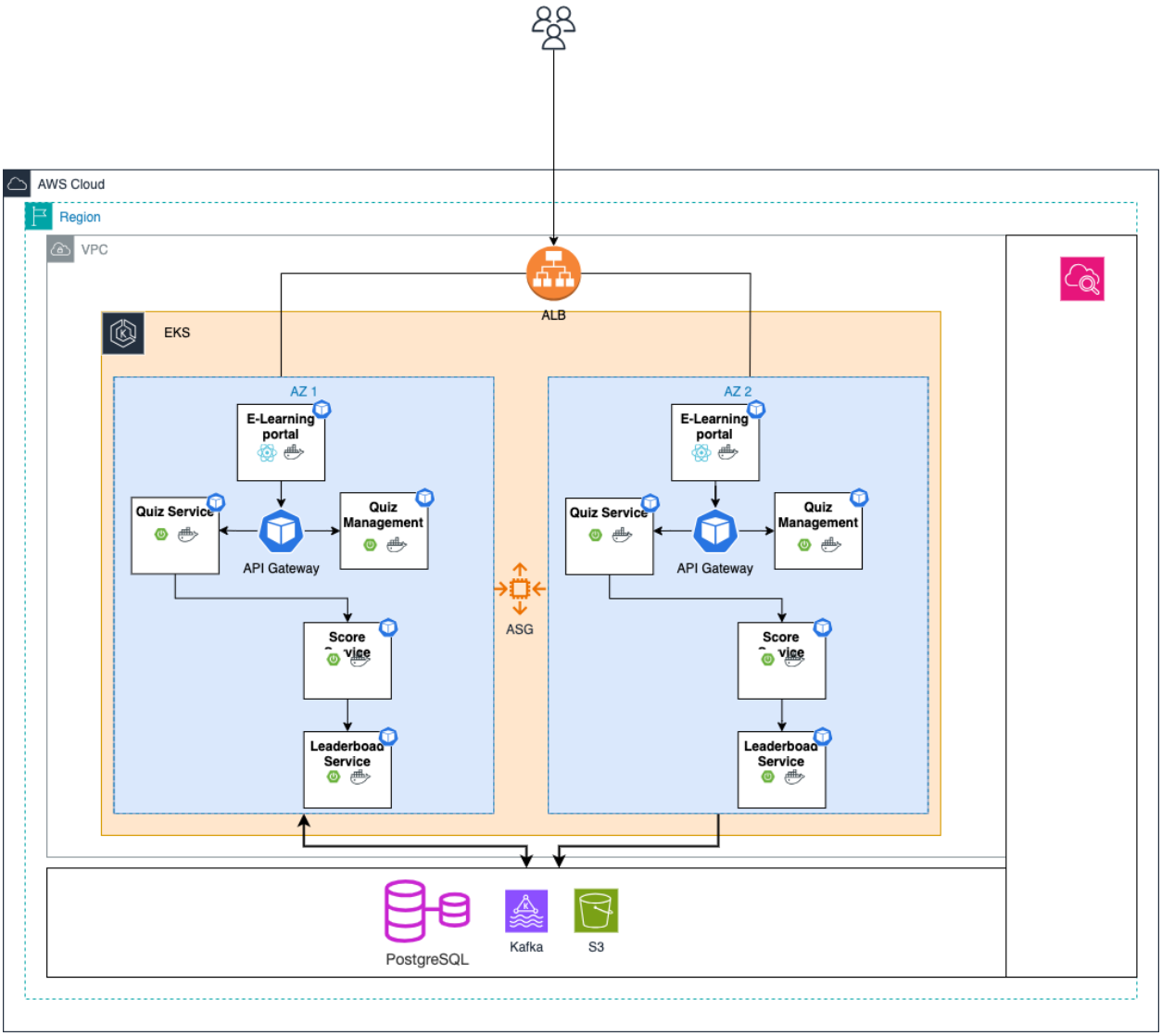
:

3.3. Deployment Architecture

Cloud solution

- Container orchestration: Deploy application to AWS EKS
- Database: RDS PostgreSQL with high Availability architecture with multiple Availability Zone (AZ)
- Kafka cluster: AWS MSK
- Redis cluster: N/A
- S3: Use to store image/audio of quiz, no need version data
- Availability Zone: 3 AZ, use auto scaling group (ASG)
- Resources:
  - 1 nginx web server: for portal web
  - Need 1 pod for API Gateway
  - Need 3 pods: quiz service, score service
  - Need 2 pods: leaderboard service

- Need 1 pod for Quiz Management



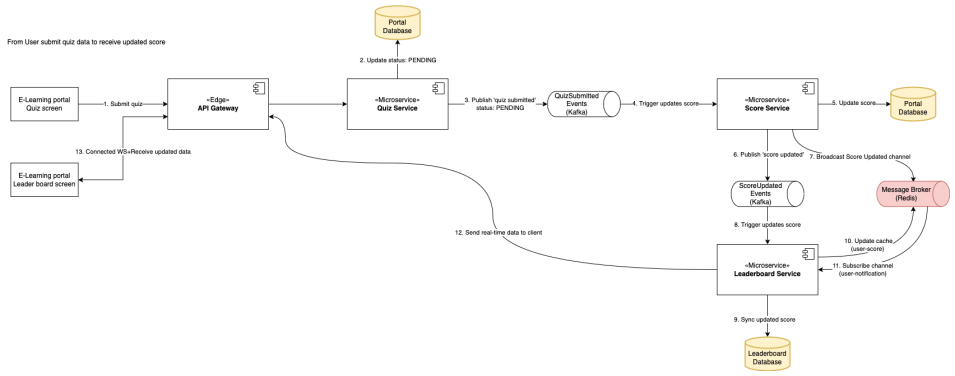
On premise solution

- Container orchestration: Deploy application to Openshift Platform
- Redis cluster sentinel

Source	Protocol	Port	Destination	Port	Authentication		
E-Learning portal	HTTP		API Gateway		JWT Web token		

			Use authentication code flow with PKCE		(Accesstoken)		

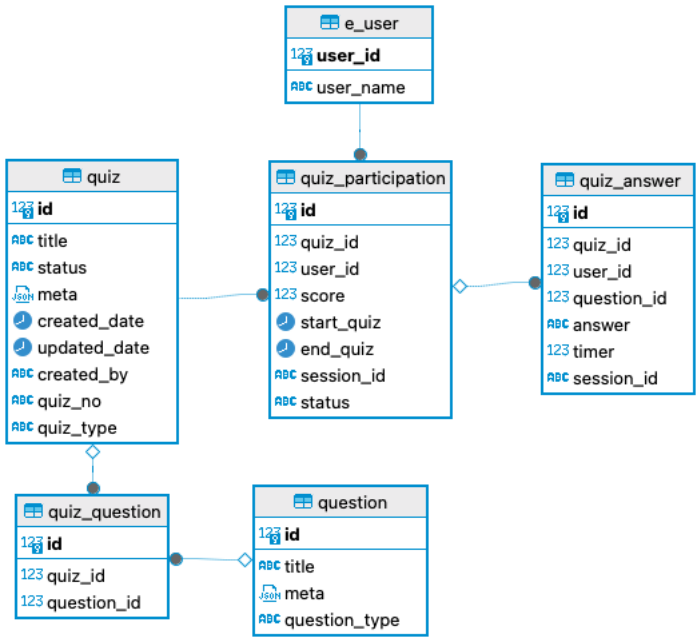
3.4. Data Flow



3.5. Database Design

- Portal database: The primary database, used for domain services such as Quiz Service, Score Service, and Quiz Management.
- Leaderboard database: This database is utilized for statistics, historical data, and should be used for kind of query operation.

> Portal database:




```
CREATE TABLE portal.e_user (
    user_id int4 GENERATED BY DEFAULT AS IDENTITY( INCREMENT BY 1 MINVALUE 1 MAXVALUE 2147483647
    user_name varchar NULL,
    CONSTRAINT user_pk PRIMARY KEY (user_id),
    CONSTRAINT user_unique UNIQUE (user_name)
);

CREATE TABLE portal.quiz (
    id int4 GENERATED BY DEFAULT AS IDENTITY( INCREMENT BY 1 MINVALUE 1 MAXVALUE 2147483647 STAR
    title varchar NOT NULL,
    status varchar NOT NULL,
    meta jsonb NULL,
    created_date timestamp NULL,
    updated_date timestamp NULL,
    created_by varchar NULL,
    quiz_no varchar NOT NULL,
    quiz_type varchar NULL,
    CONSTRAINT quiz_pk PRIMARY KEY (id),
    CONSTRAINT quiz_unique UNIQUE (quiz_no)
);

CREATE TABLE portal.question (
    id int4 GENERATED BY DEFAULT AS IDENTITY( INCREMENT BY 1 MINVALUE 1 MAXVALUE 2147483647 STAR
    title text NULL,
    meta jsonb NULL,
    question_type varchar NULL,
    CONSTRAINT question_pk PRIMARY KEY (id)
);
```

```
CREATE TABLE portal.quiz_question (
    id int4 GENERATED BY DEFAULT AS IDENTITY( INCREMENT BY 1 MINVALUE 1 MAXVALUE 2147483647 STAR
    quiz_id int4 NULL,
    question_id int4 NULL,
    CONSTRAINT quiz_question_pk PRIMARY KEY (id)
);


-- portal.quiz_question foreign keys

ALTER TABLE portal.quiz_question ADD CONSTRAINT quiz_question_question_fk FOREIGN KEY (question_
ALTER TABLE portal.quiz_question ADD CONSTRAINT quiz_question_quiz_fk FOREIGN KEY (quiz_id) REFE

-- portal.quiz_participation definition

-- Drop table

-- DROP TABLE portal.quiz_participation;

CREATE TABLE portal.quiz_participation (
    id int4 GENERATED BY DEFAULT AS IDENTITY( INCREMENT BY 1 MINVALUE 1 MAXVALUE 2147483647 STAR
    quiz_id int4 NOT NULL,
    user_id int4 NOT NULL,
    score numeric NULL,
    start_quiz timestamp NULL,
    end_quiz timestamp NULL,
    session_id varchar NULL,
    status varchar NULL,
    CONSTRAINT quiz_participation_pk PRIMARY KEY (id),
    CONSTRAINT quiz_participation_unique UNIQUE (session_id)
);


-- portal.quiz_participation foreign keys

ALTER TABLE portal.quiz_participation ADD CONSTRAINT fk_quiz_participation_quiz_id FOREIGN KEY (
ALTER TABLE portal.quiz_participation ADD CONSTRAINT quiz_participation_e_user_fk FOREIGN KEY (u

CREATE TABLE portal.quiz_answer (
    id int4 GENERATED BY DEFAULT AS IDENTITY( INCREMENT BY 1 MINVALUE 1 MAXVALUE 2147483647 STAR
    quiz_id int4 NOT NULL,
    user_id int4 NOT NULL,
    question_id int4 NOT NULL,
    answer varchar NULL,
    timer int4 NULL,
    session_id varchar NULL,
    CONSTRAINT quiz_answer_pk PRIMARY KEY (id)
);


-- portal.quiz_answer foreign keys

ALTER TABLE portal.quiz_answer ADD CONSTRAINT quiz_answer_quiz_participation_fk FOREIGN KEY (ses
```

> Leaderboard database:

```
CREATE TABLE leaderboard.user_score (
    id int4 GENERATED BY DEFAULT AS IDENTITY NOT NULL,
    user_id int4 NULL,
    username varchar NULL,
    score numeric NULL,
    CONSTRAINT user_score_pk PRIMARY KEY (id)
);
```

2.4. API Details

Component	**Quiz Management**
Feature Id	**Create quiz**
API Code	01
API Endpoint	**POST /api/quizzes**
Description	A CRUD API for creating a quiz. The application will respond with a Quiz Number (quiz No field), this number is used to join the quiz.

```
// Request Body
{
    "tile": "ELSA - IELTS Test",
    "questions": [
        {
            "question": "______ are my trousers.",
            "options": [
                "This",
                "These"
            ],
            "correctOptions": [
                "These"
            ]
            "config": { // where to specify the configuration for a question
                "type": "SINGLE" // SINGLE|MULTIPLE
            }
        }
    ],
    "meta": { // where to specify the configuration for a quiz
        "quizTime": 3600000 // time duration for completing a quiz
        "maximumQuestion": 10
    }
}
```

```
// Response
{
  "id": 1,
  "tile": "ELSA - IELTS Test",
```

```
  "quizNo": "22354" // A generated number by system with a maximum of 5 digits, user use this id
}
```

Component	Quiz Service	
Feature Id	**Get quiz Info**	
API Code	02	
API Endpoint	**GET /api/quizzes?quiz_no=22335**	
Description	Get quiz info by param quiz_no	

```
// Response
{
    "quizId": 1,
    "quizNo": "22354",
    "title": "ELSA - IELTS Test",
    "quizTime": 3600000,
    "note": "Time 60 minutes"
}
```

Component	Quiz Service
Feature Id	**Join quiz**
API Code	03
API Endpoint	**POST /api/quizzes/join**
Description	Allows users to join a quiz using a provided quiz number

> Currently, system only supports joining a quiz using the quiz id

```
// Request Body
{
     "quizNo": "22354",
     "userName": "tunv"
}
```

```
// Response
{
    "sessionId": "653d93e4-add8-42e6-942a-a0ebadf828ca",
    "quizId": 1,
    "quizNo": "22354",
    "userId": 1,
```

```
    "userName": "tunv",
    "startTime": "2024-07-10T11:25:15.091763",
    "endTime": null,
    "score": 0.0,
    "questions": [
        {
            "questionId": 1,
            "question": "______ are my trousers.",
            "options": [
                "This",
                "These"
            ],
            "config": {
                "type": "SINGLE"
            }
        },
        {
            "questionId": 2,
            "question": "Is ______ hotel nice?",
            "options": [
                "this",
                "that"
            ],
            "config": {
                "type": "SINGLE"
            }
        },
        {
            "questionId": 3,
            "question": "Are ______ your friends?",
            "options": [
                "these",
                "those"
            ],
            "config": {
                "type": "SINGLE"
            }
        }
    ]
}
```

Component	Score Service
Feature Id	**Submit answers**
API Code	04
API Endpoint	**POST /api/quizzes/submit**
Description	- Submit answers for a quiz - Auto submit answers when quiz time is expired

```
// Request body
{
```

```
  "quizId": 1,
  "userId": 1,
  "sessionId": "653d93e4-add8-42e6-942a-a0ebadf828ca",
  "data": [
    {
      "questionId": 1,
      "selectedOptions": [
        "These"
      ],
      "timer": 30000
    },
    {
      "questionId": 2,
      "selectedOptions": [
        "that"
      ],
      "timer": 20000
    },
    {
      "questionId": 3,
      "selectedOptions": [
        "these"
      ],
      "timer": 10000
    }
  ]
}

// Response
{
    "quizId": 1,
    "quizNo": "22354",
    "quizTitle": "ELSA - IELTS Test",
    "sessionId": "653d93e4-add8-42e6-942a-a0ebadf828ca",
    "endQuiz": "2024-07-10T11:39:52.371789"
}
```

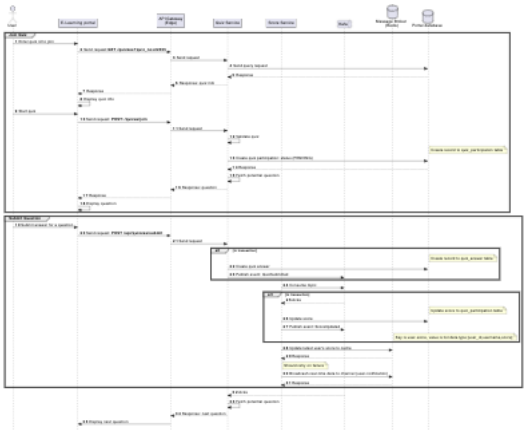
Component	Leaderboard Service
Feature Id	**Get all user score**
API Code	05
API Endpoint	**GET /api/quizzes/score**
Description	Fetch user score

```
// Request body
[
    {
```

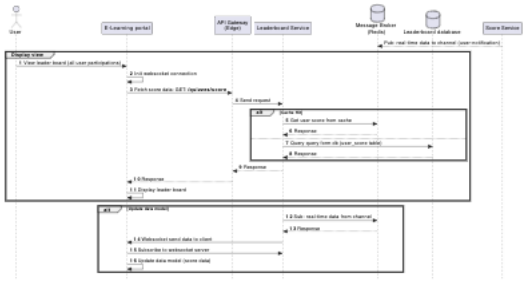
```
      "userId": 1,
      "userName": "tunv",
      "score": 3500.00
    },
    {
      "userId": 2,
      "userName": "david",
      "score": 320.00
    }
]
```

3.6. Sequence Diagram

- User start quiz (From user starting a quiz to ending quiz)



- Auto ending quiz (user run out of time, the application will automatically submit the quiz)
- Fetching user score



3.7. Message Design

Topic/channel	Description	Permission	Producer	Consumer	
QuizSubmitted	Kafa topic Payload: {	Write	Quiz Service	Score Service	

	"quizId": 1,
	"userId": 1,
	"sessionId": "653d93e4-add8-42e6-942a-a0ebadf828ca",
	"data": [
	{
	"questionId": 1,
	"selectedOptions": [
	"These"
	],
	"timer": 30000
	},
	{
	"questionId": 2,
	"selectedOptions": [
	"that"
	],
	"timer": 20000
	},
	{
	"questionId": 3,
	"selectedOptions": [
	"these"
	],
	"timer": 10000
	}
	]
	}				

ScoreUpdated	Kafka topic **{** **"quizId": 1,** **"userId": 1,** **"score": 10.00,** **}**		Score Service	Leaderboard Service	
user-score	Redis data store: List Payload value: **[** **{** **"userId": 1,** **"userName": "tunv",** **"score": 3500.00** **},** **{** **"userId": 2,** **"userName": "david",** **"score": 3200.00** **},** **{** **"userId": 3,** **"userName": "johnny",** **"score": 3000.00** **}** **]**	Write	Leaderboard Service		

user-notification	Redis channel Payload: **[** **{** **"userId": 1,** **"userName": "tunv",** **"score": 3500.00** **},** **{** **"userId": 2,** **"userName": "david",** **"score": 3200.00** **},** **{** **"userId": 3,** **"userName": "johnny",** **"score": 3000.00** **}** **]**	Write	Pub: Score Service	Sub: Leaderboard Service	

3.8. Development requirements

Following to principle design:

- Microservice based on data driven architecture
- Database per service
- Service communication: REST API, Asynchronous message
- Pub/Sub design pattern
- CQRS design pattern
- API Gateway and authentication with Oauth2

4. DevSec ops

TBC