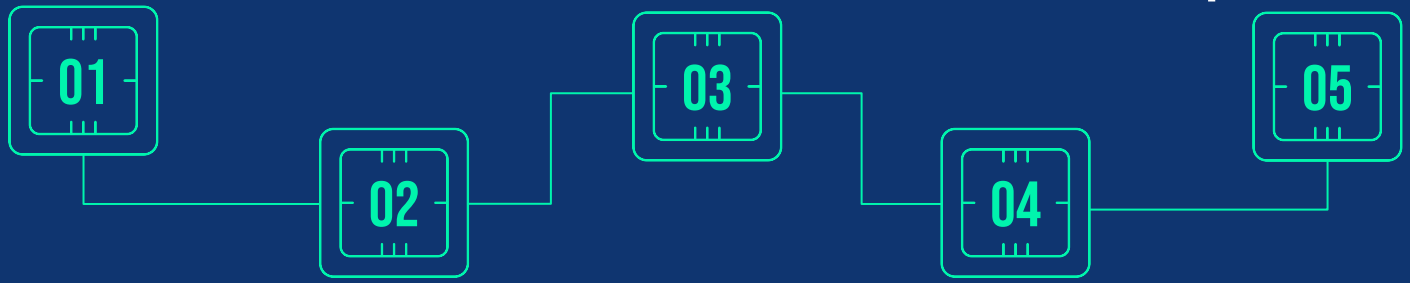
RECIPE DIARY

ENTERPRISE APPLICATION INTEGRATION

Johanna Lipka
Giouri Kilinkaridis

OUTLINE



Description

Functional Requirements

Architecture

Architecture in terms of EIP

Mapping Patterns to Implementation

DESCRIPTION

Recipe Diary

The idea was to create an application that allows the user to quickly find a recipe based on an any ingredient he/she wants to try and cook.

DESCRIPTION

1. Fast and easy way to find recipes

- Save time and effort on discovering new recipes.

DESCRIPTION

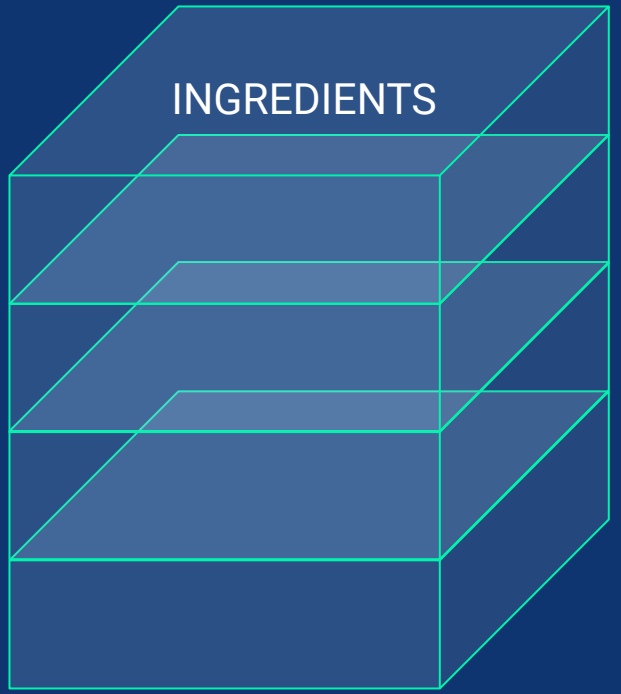
2. Over 2.3 millions of recipes to look for.

- Recipes from all over the world with a click of a button

DESCRIPTION

3. Full list of ingredients

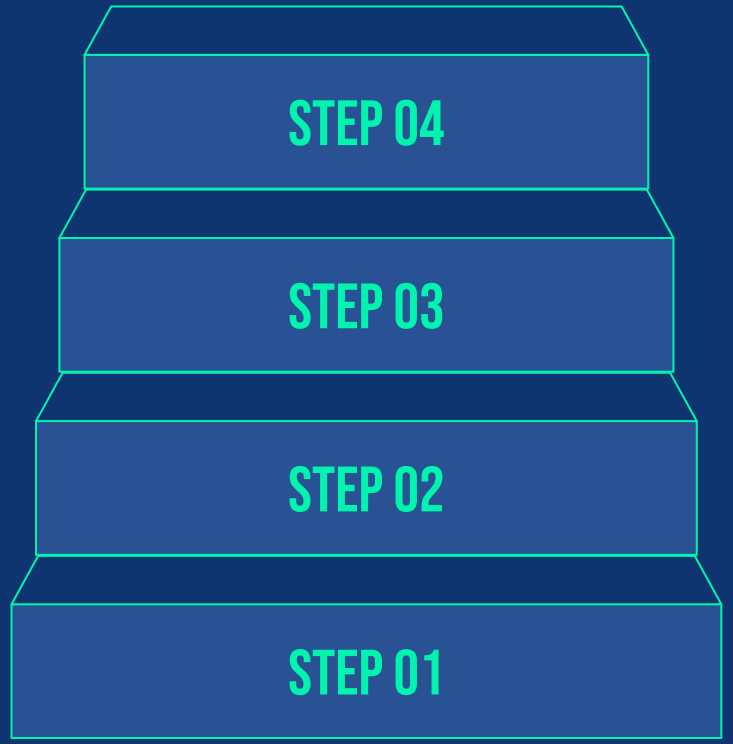
- Ingredients are scattered all over the video, making the user pause and rewind to find them and write them down.



DESCRIPTION

4. A YouTube video showing the cooking process.

- A video offering a step by step guide on how to prepare your meal is always helpful.



DESCRIPTION

5. A local map of available supermarkets based on user’s position

- An embedded map with supermarkets near you.

FUNCTIONAL REQUIREMENTS

- User Registration

  Users should be able to create accounts and log in.

FUNCTIONAL REQUIREMENTS

- Friendly User Interface

  UI should be simple and practical for usage.

FUNCTIONAL REQUIREMENTS

- Ingredient Input

  User should be input just the ingredient he wants without the need to describe the entire recipe.

FUNCTIONAL REQUIREMENTS

- Recipe Details

  The app should display details regarding each recipe.

FUNCTIONAL REQUIREMENTS

- Display Recipes

  The app should display an image of the recipe and the ingredients.

FUNCTIONAL REQUIREMENTS

- Attach YouTube Video

  The app should display with each recipe a YouTube video of how it is made.

FUNCTIONAL REQUIREMENTS

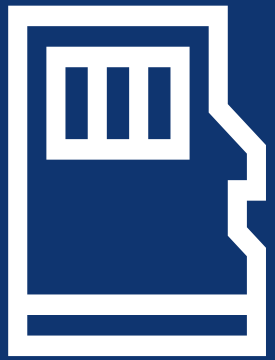
- Embed a Map with Supermarkets

  The app should display with each recipe a map with marks representing supermarkets close to users position.

FUNCTIONAL REQUIREMENTS

- Saving User’s Searched Recipes

  The app should save the recipes a user looked for or give an option to save only the recipes the user chose to save.



ARCHITECTURE

CLIENT-SIDE

SERVER-SIDE

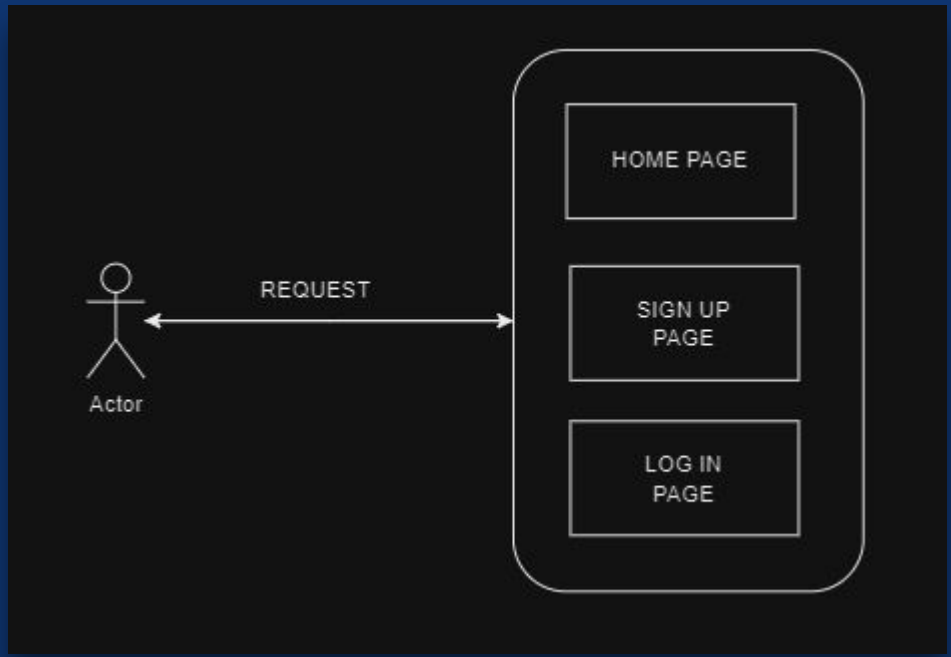
DATABASE

ARCHITECTURE

CLIENT-SIDE

Types of requests:

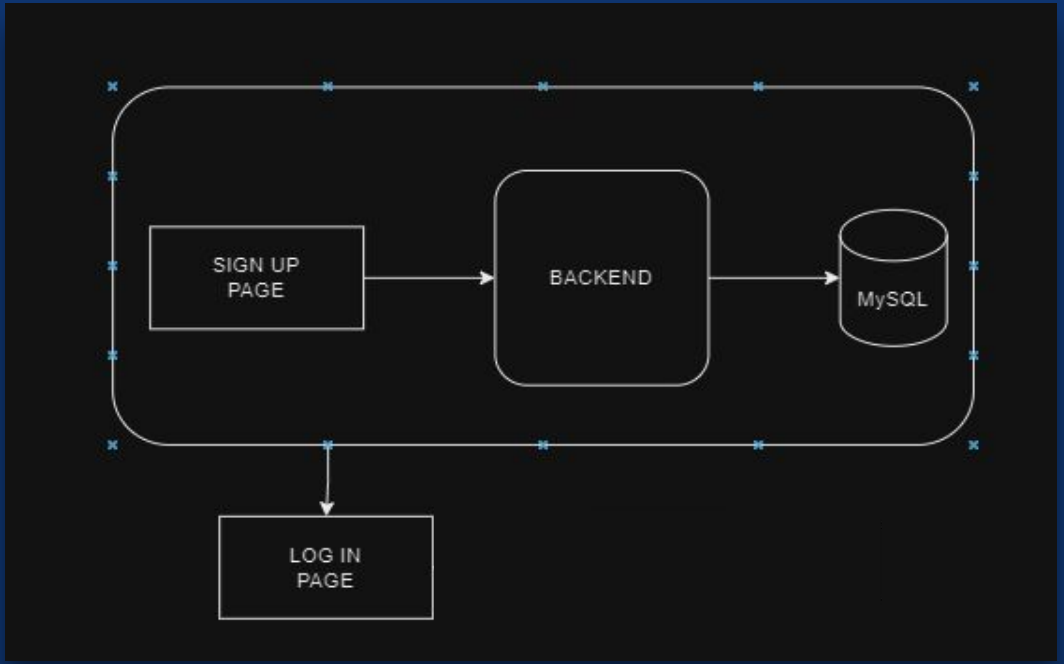
- Sign up
- Log in
- Search a recipe



ARCHITECTURE

CLIENT-SIDE

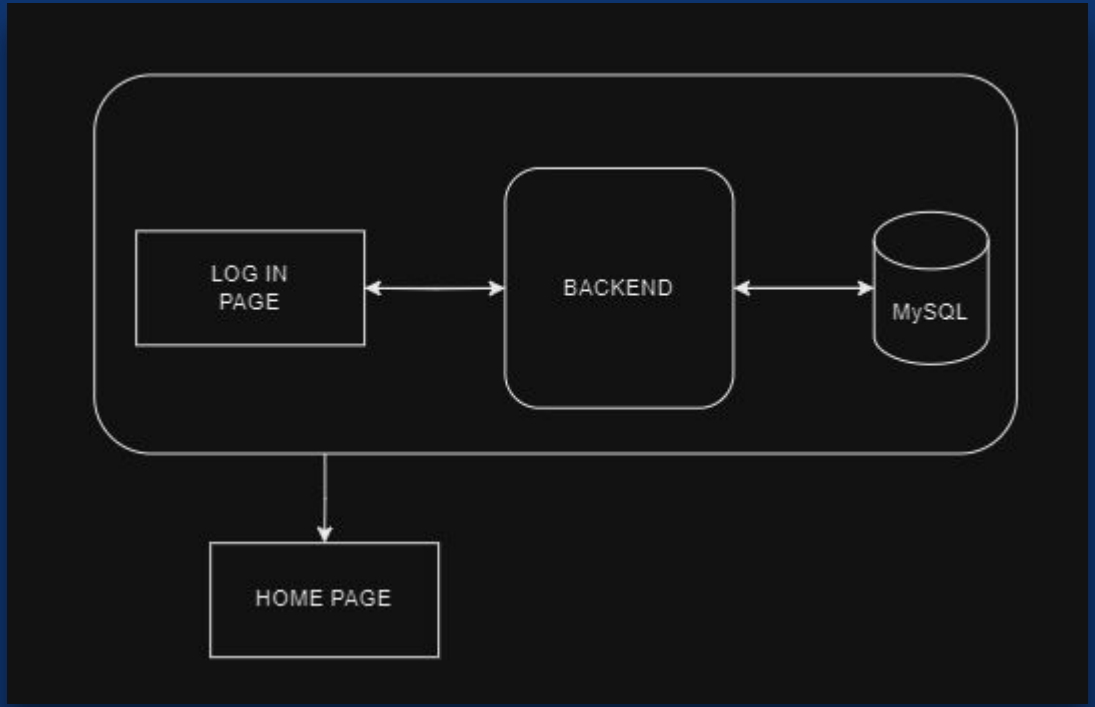
- Sign up request



ARCHITECTURE

CLIENT-SIDE

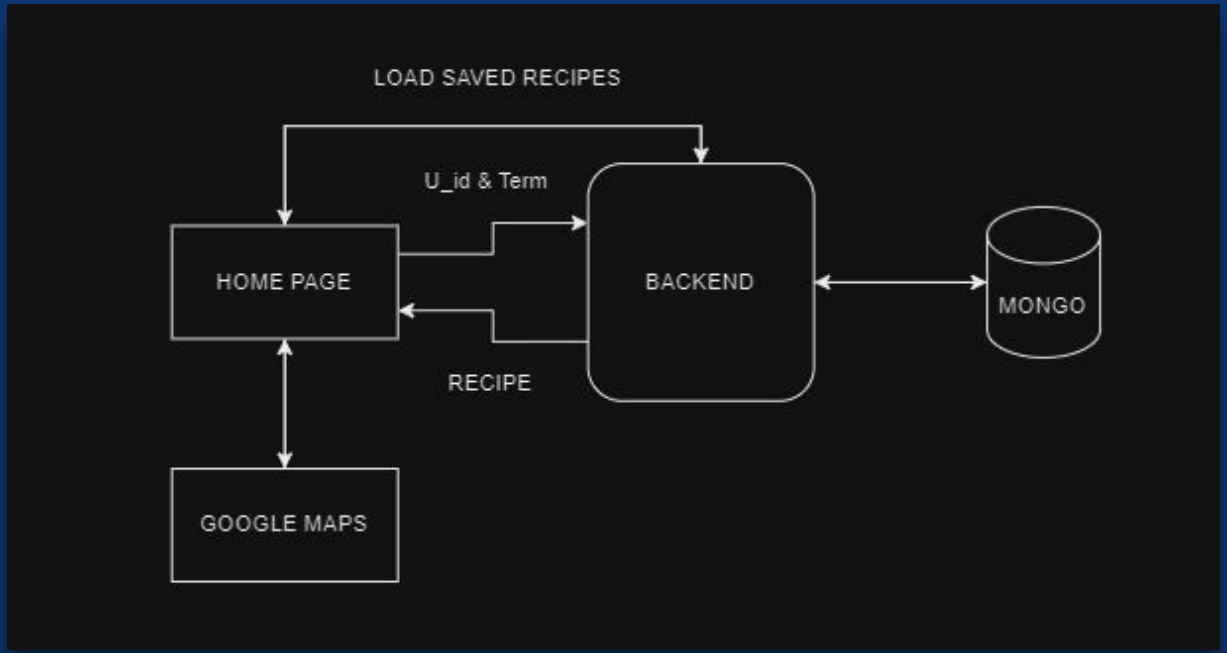
- Log in request



ARCHITECTURE

CLIENT-SIDE

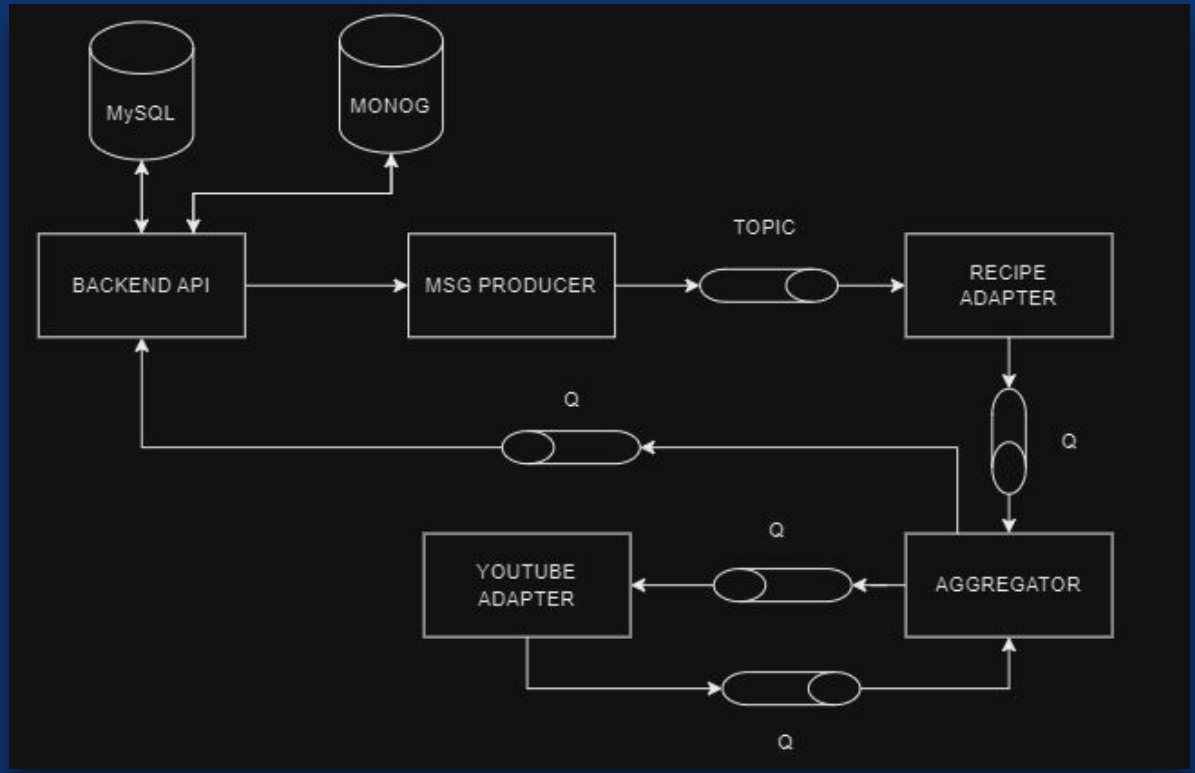
- Search Recipe



ARCHITECTURE

SERVER-SIDE

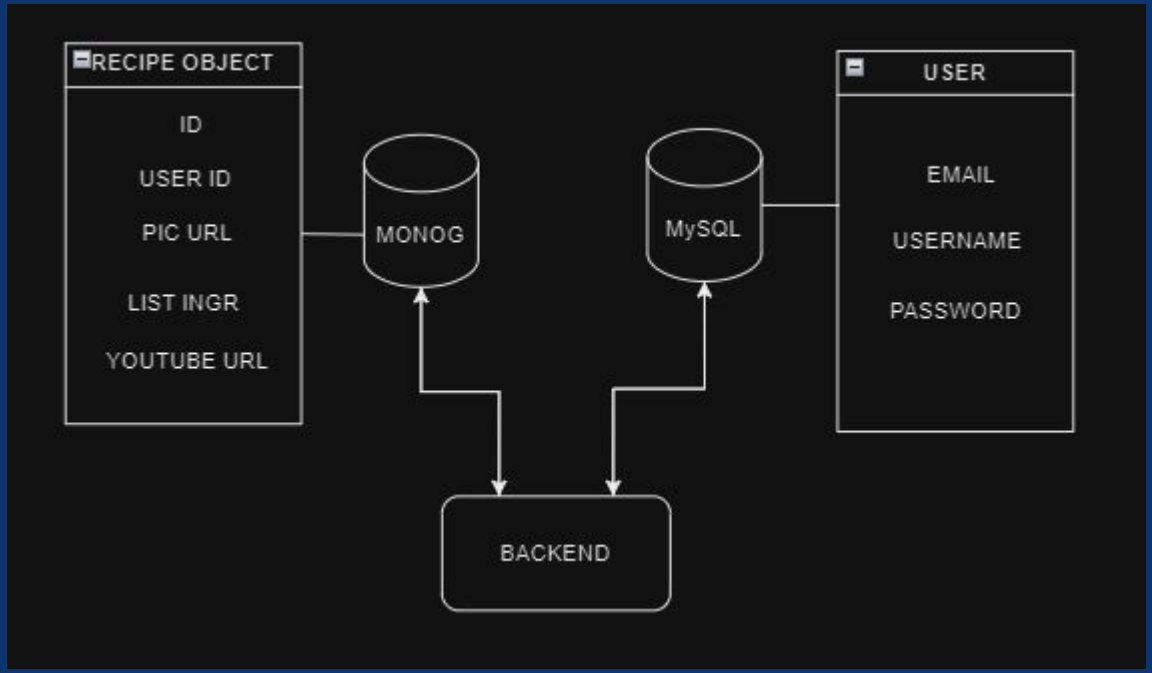
- Backend components



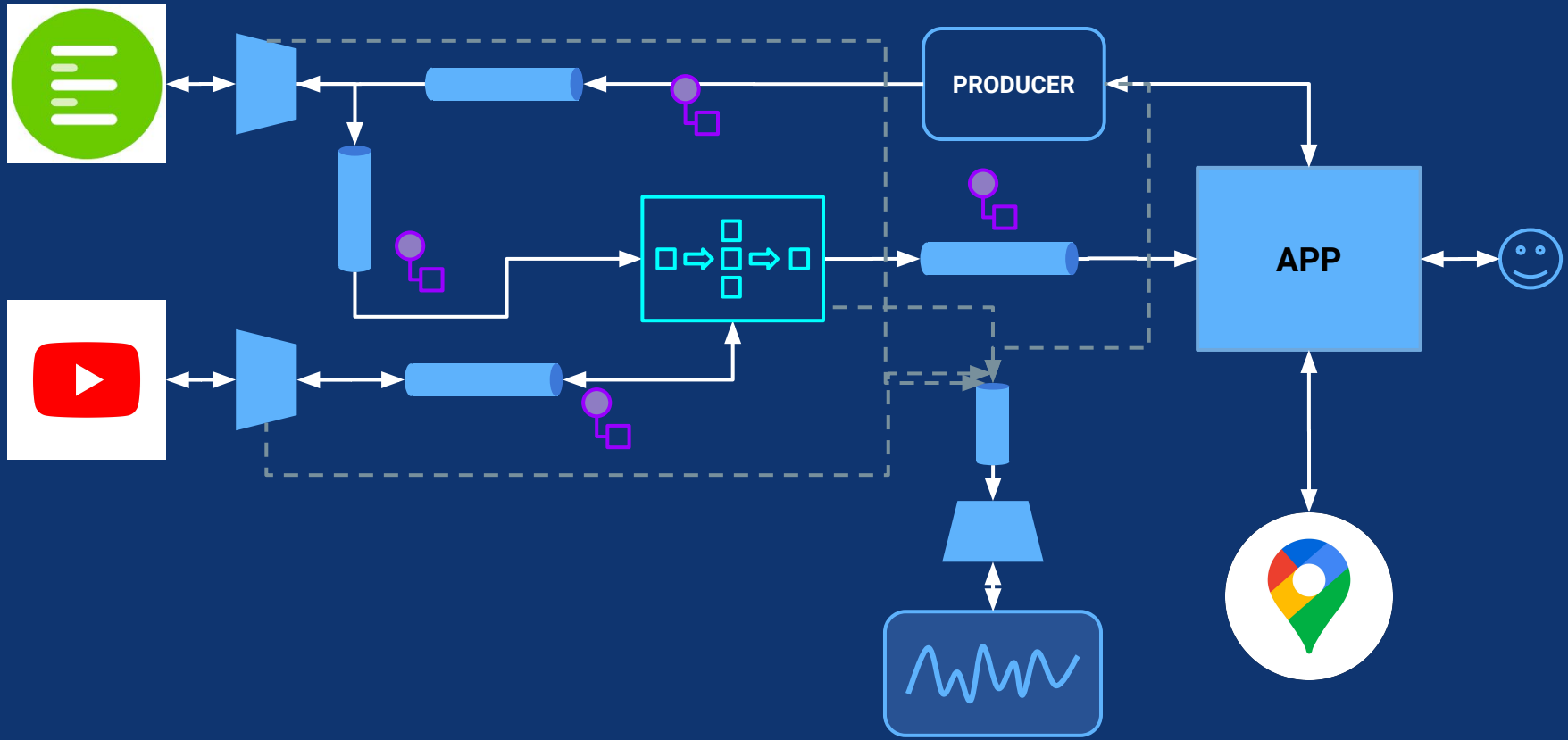
ARCHITECTURE

DATABASE

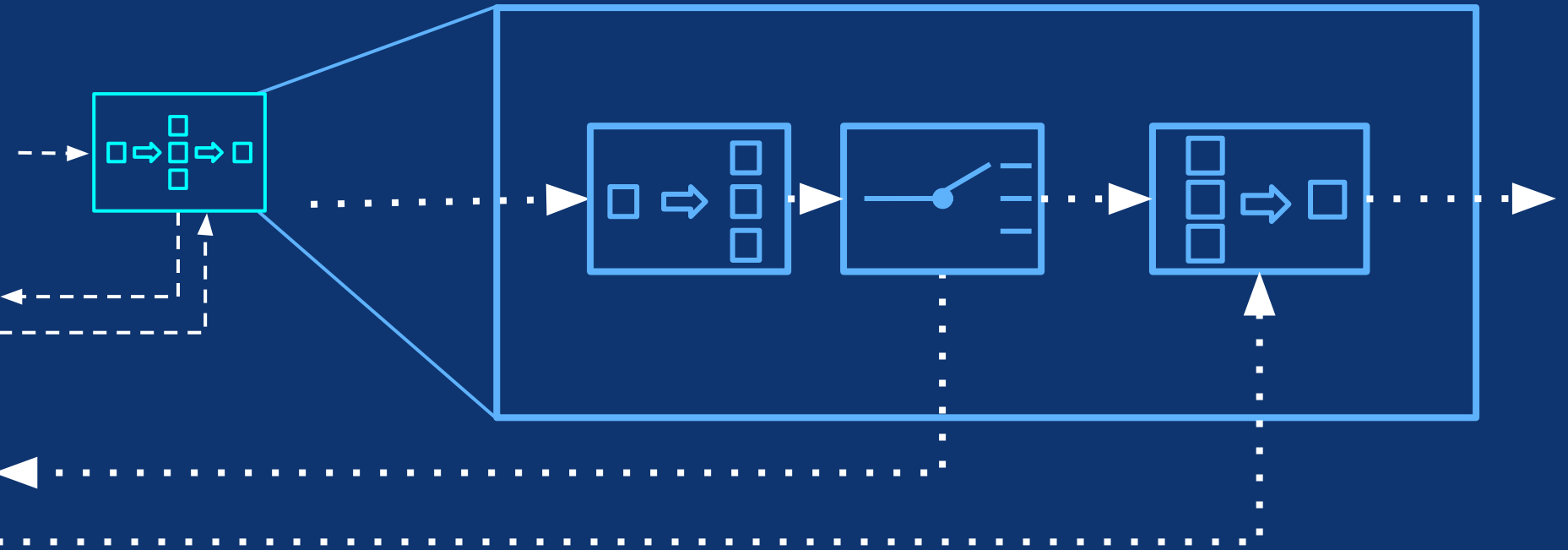
- MySQL for users
- Mongo for recipes



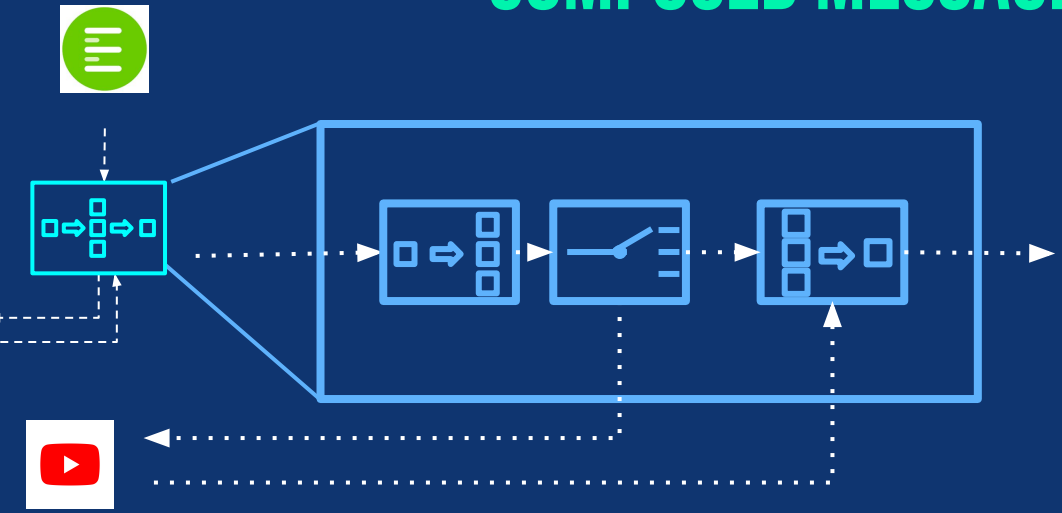
ARCHITECTURE IN TERMS OF EIP



COMPOSED MESSAGE PROCESSOR

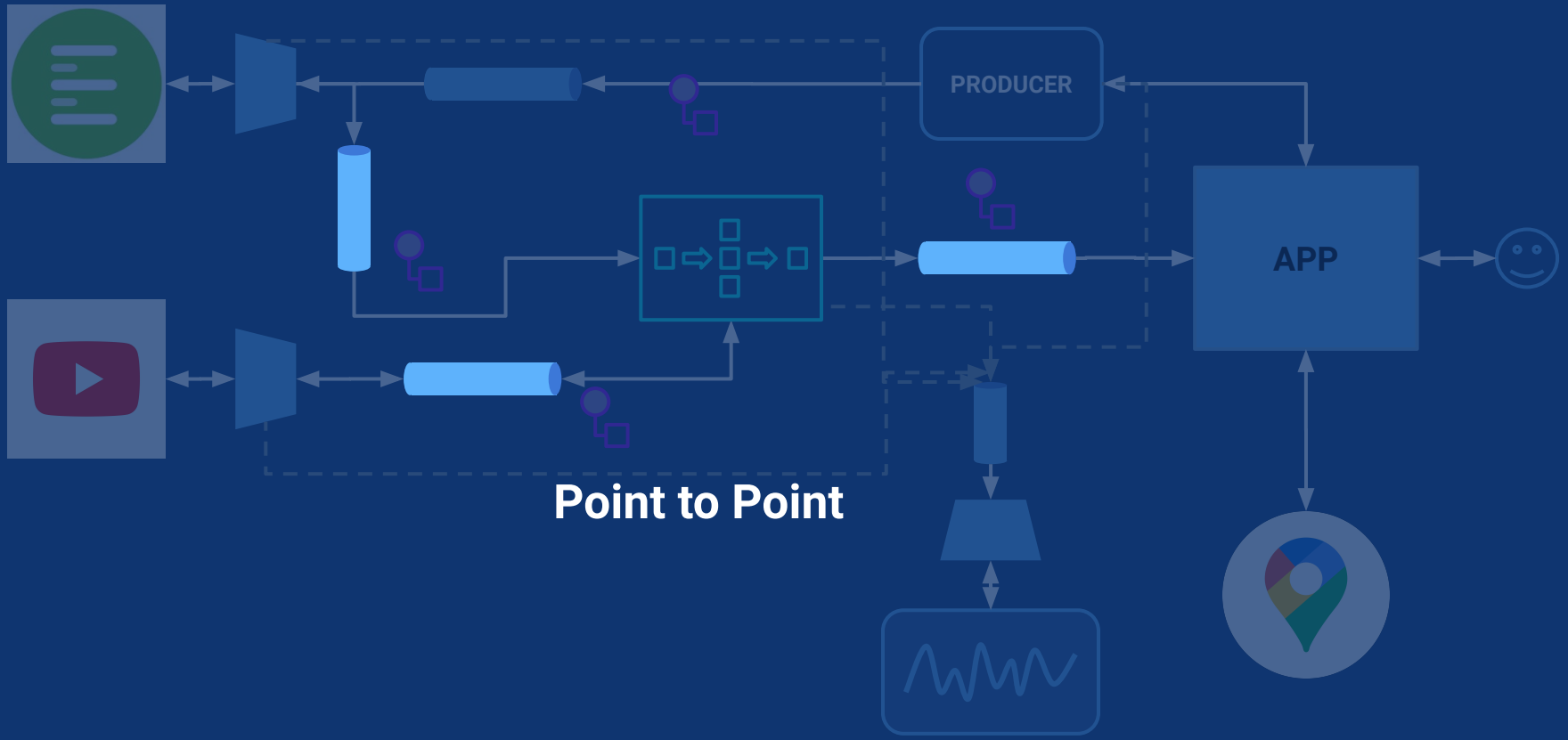


COMPOSED MESSAGE PROCESSOR



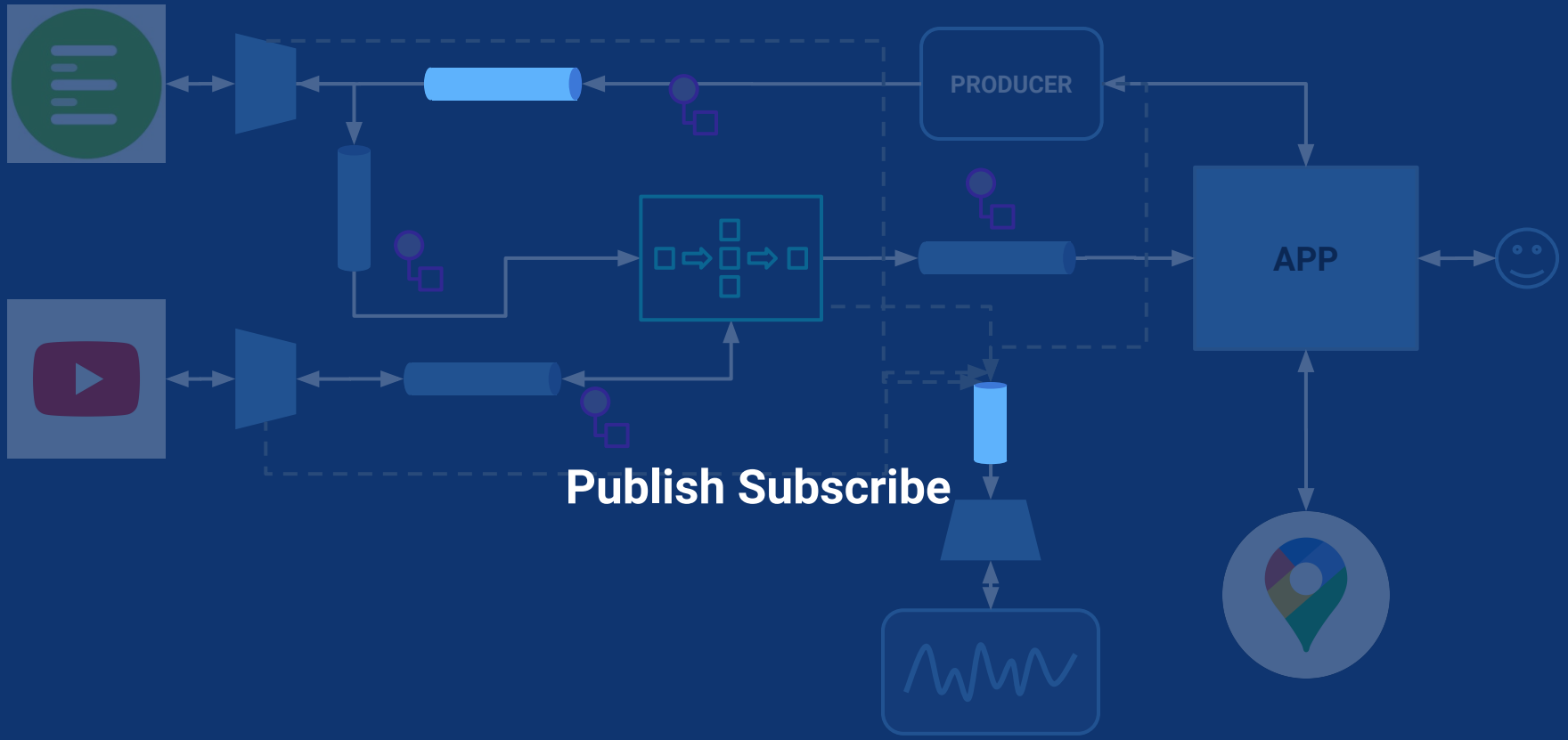
- Recipe message is sent to splitter
- Splitter extracts title and the rest of the message
- Router routes “title message” to the youtube adapter
- The “rest message” is sent directly to the aggregator
- The youtube adapter returns video details for the recipe title to the aggregator
- The aggregator aggregates bothe messages based on a message ID

CHANNELS

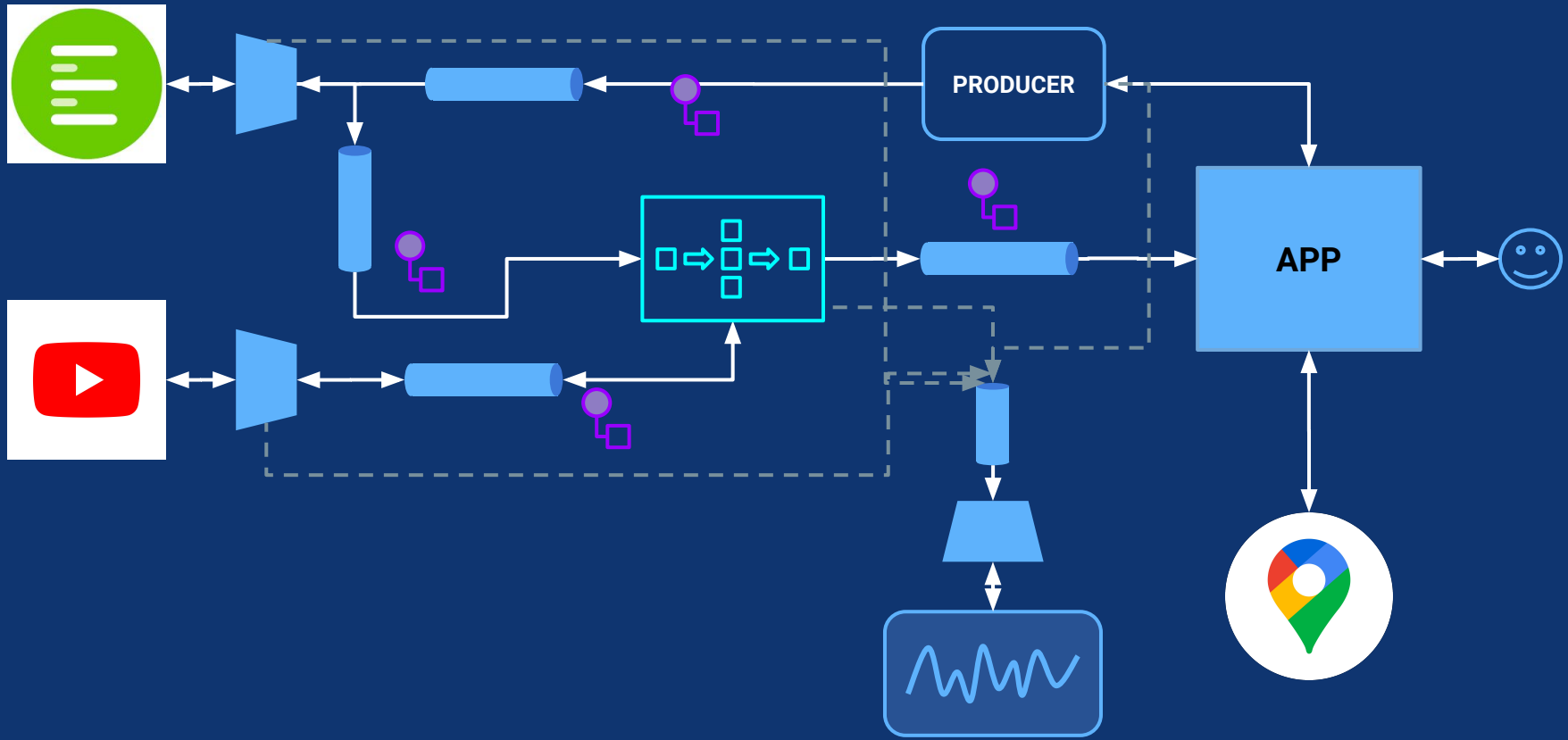


Point to Point

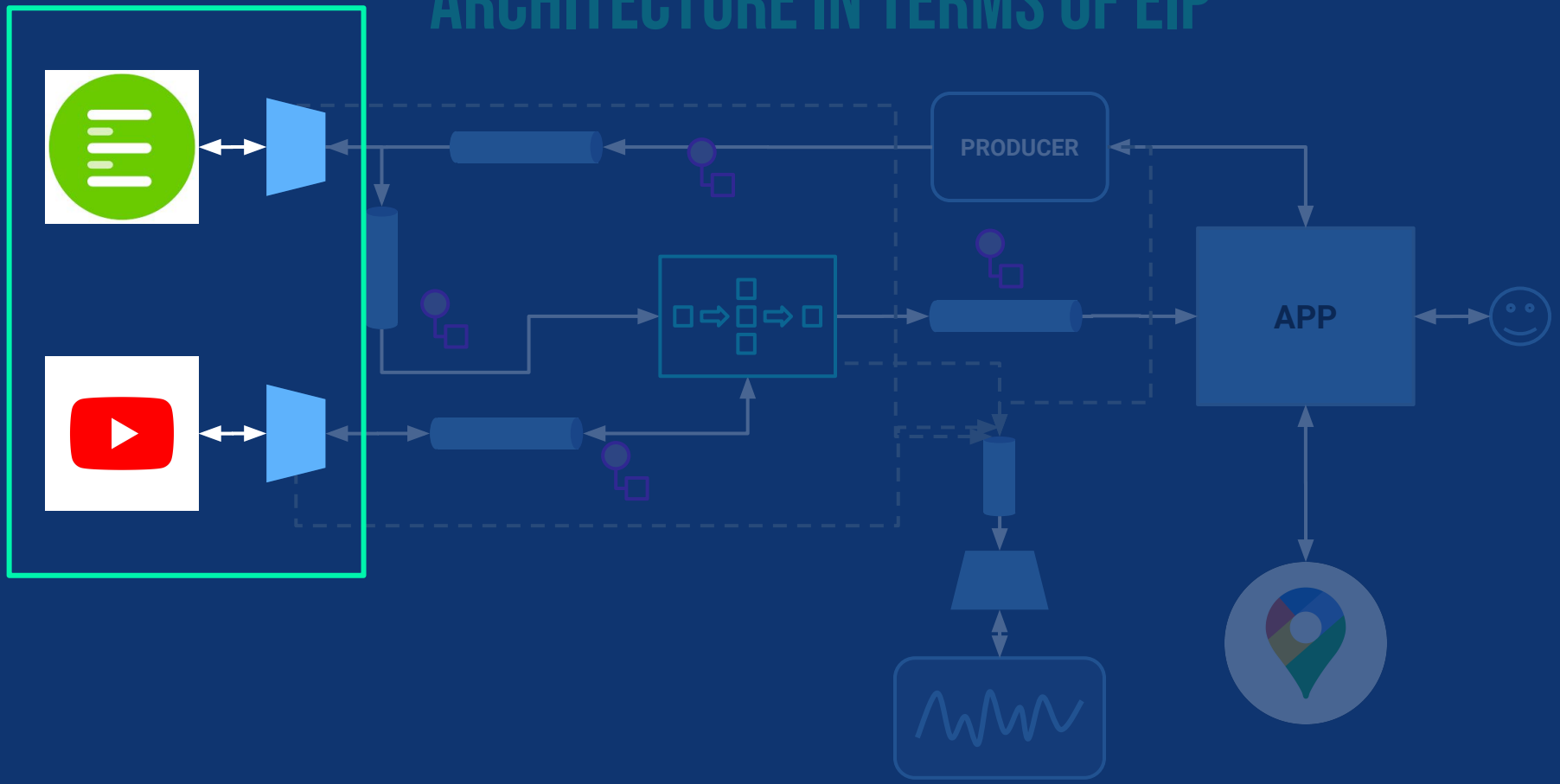
CHANNELS



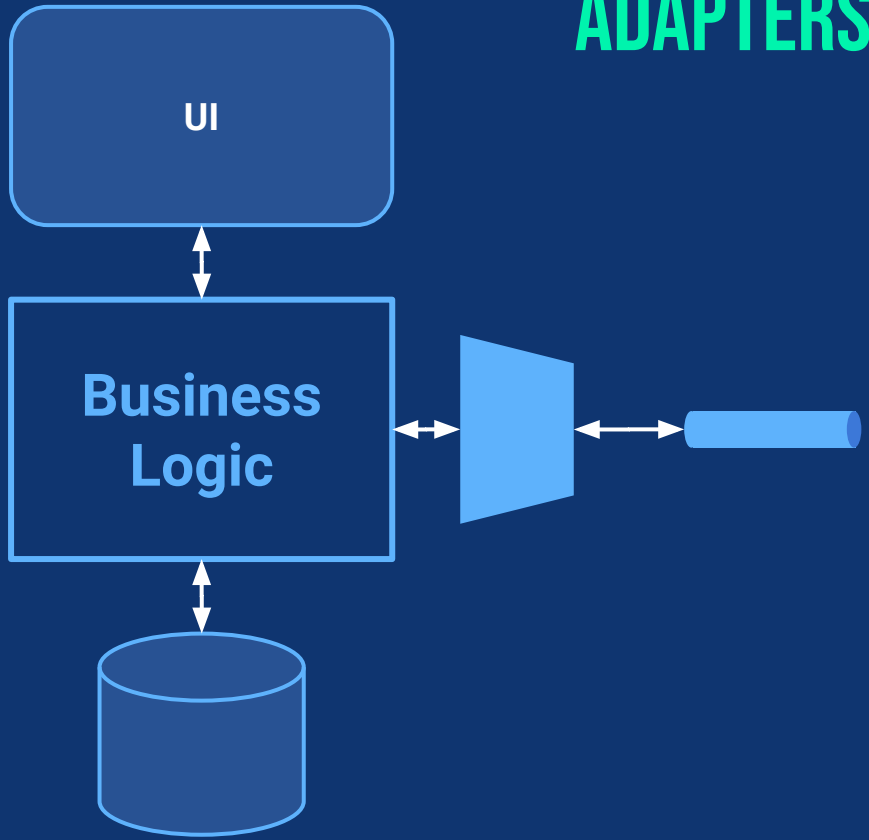
ARCHITECTURE IN TERMS OF EIP



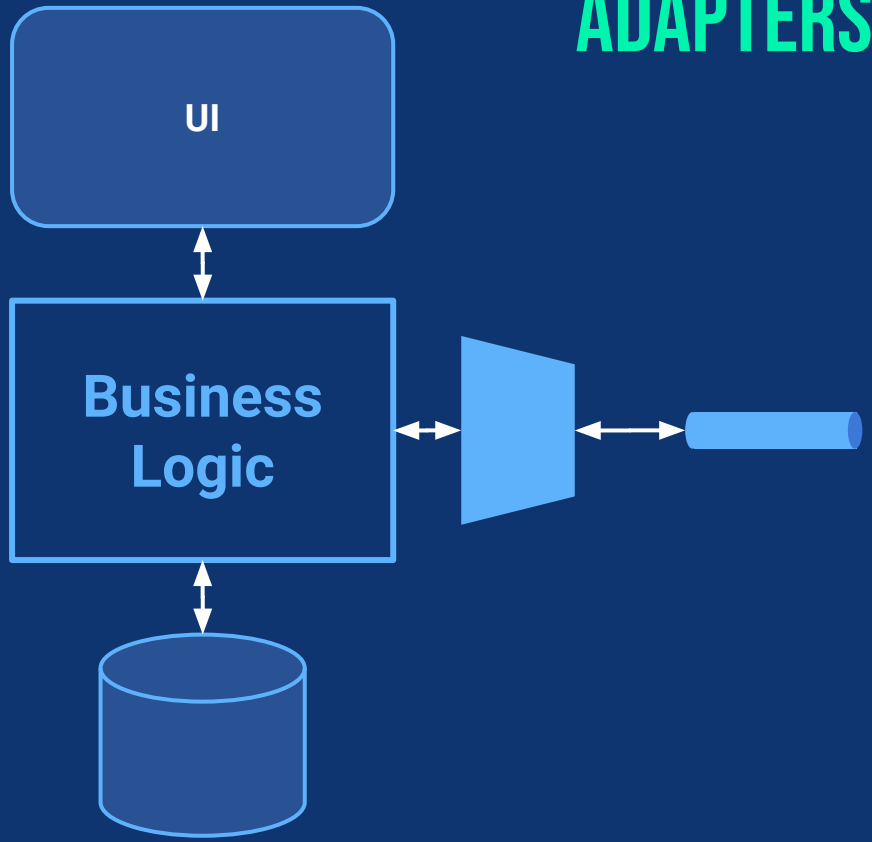
ARCHITECTURE IN TERMS OF EIP



ADAPTERS

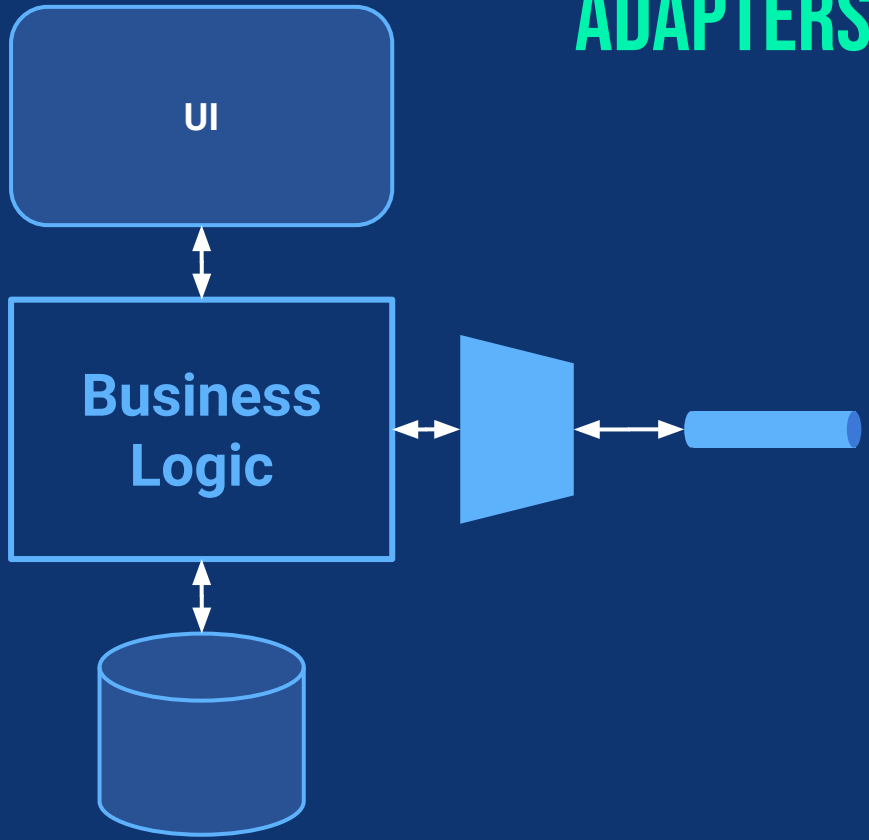


ADAPTERS



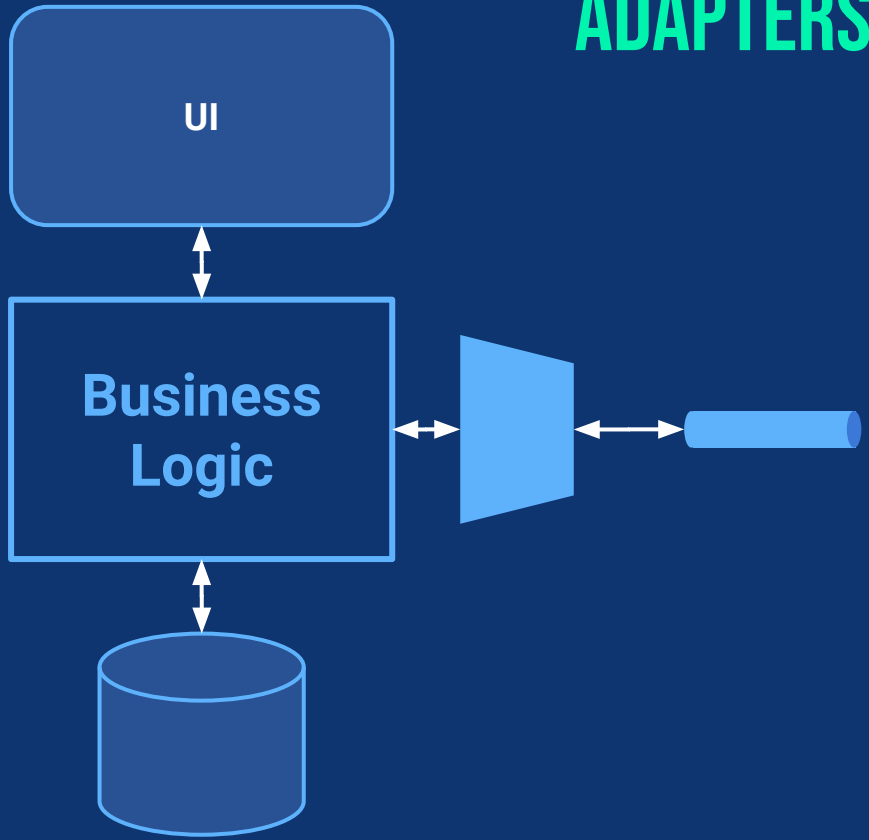
- Consumer
- Producer

ADAPTERS



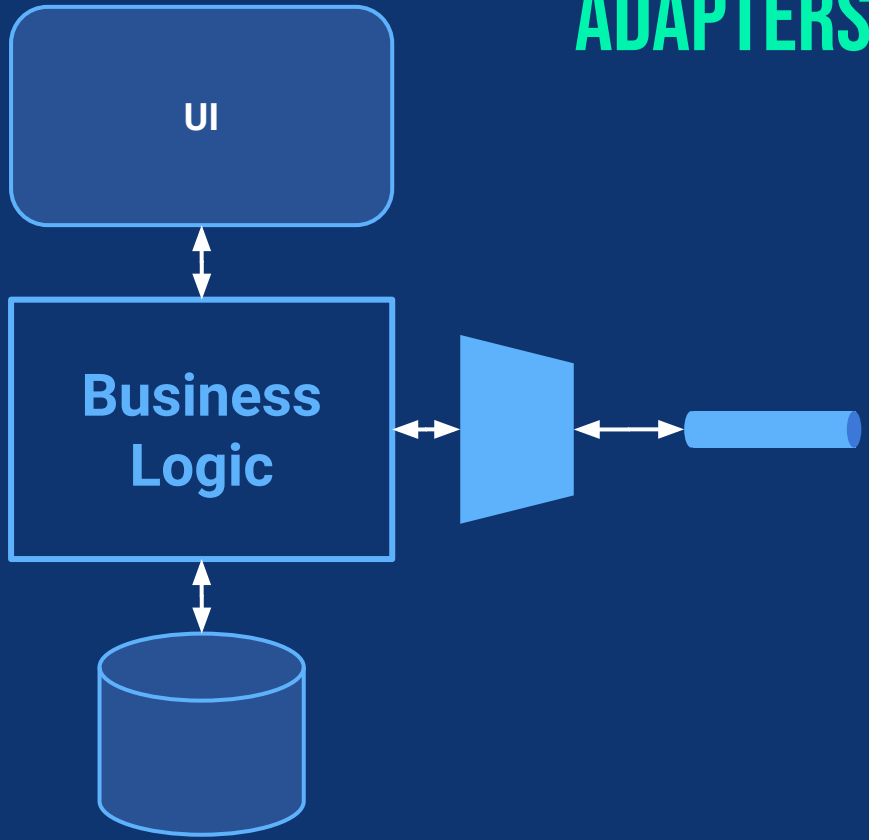
- Consumes Recipe Requests
- Produces Recipe Data

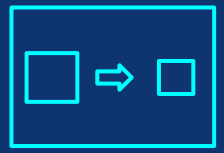
ADAPTERS



- Consumes Video Requests
- Produces Video Data

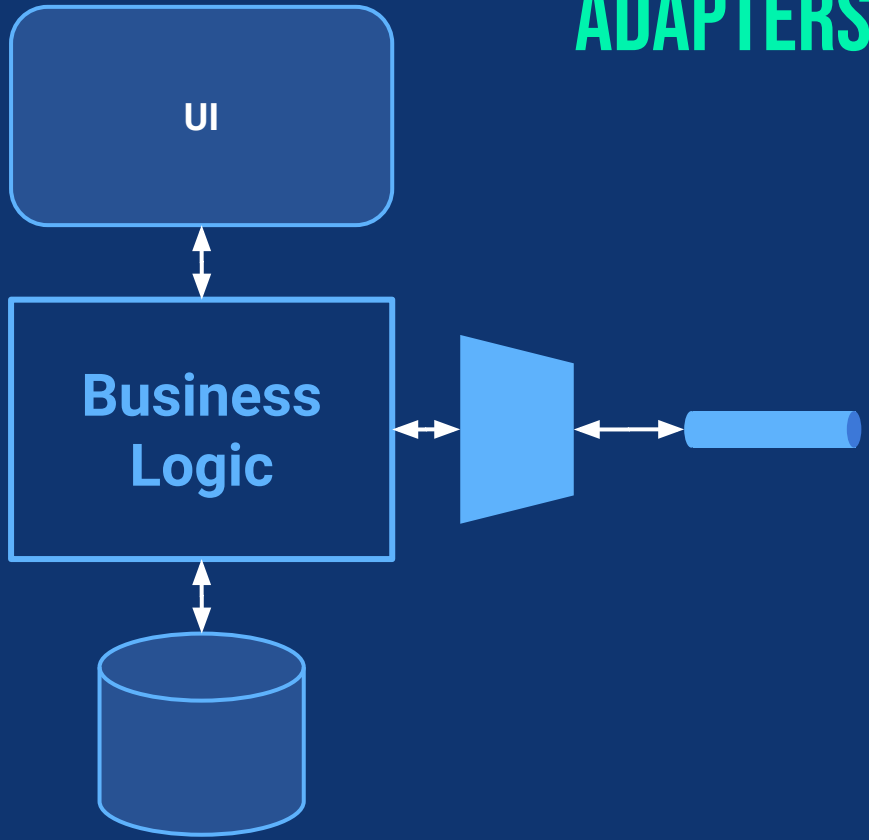
ADAPTERS

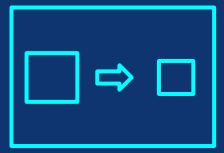




Act as content filters

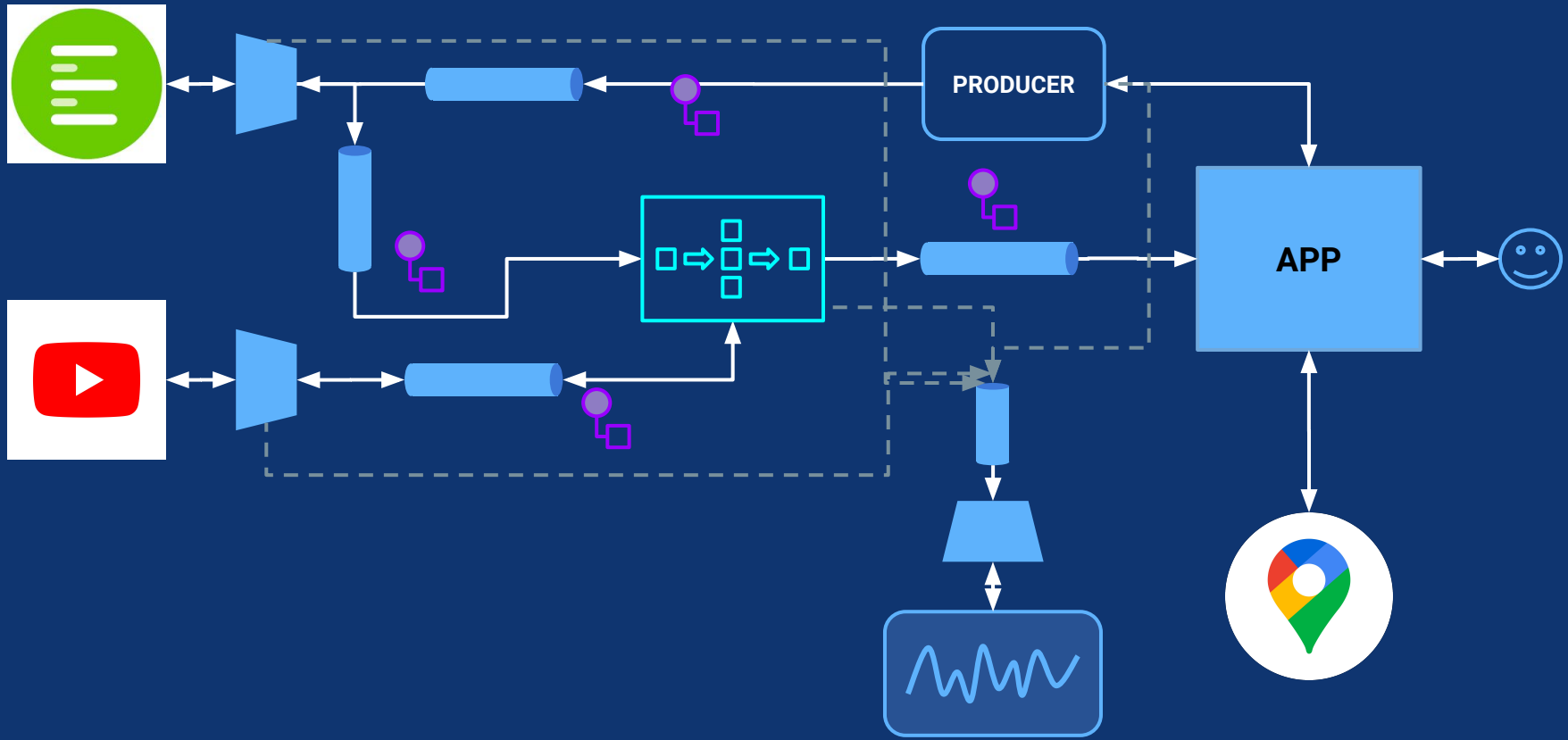
ADAPTERS



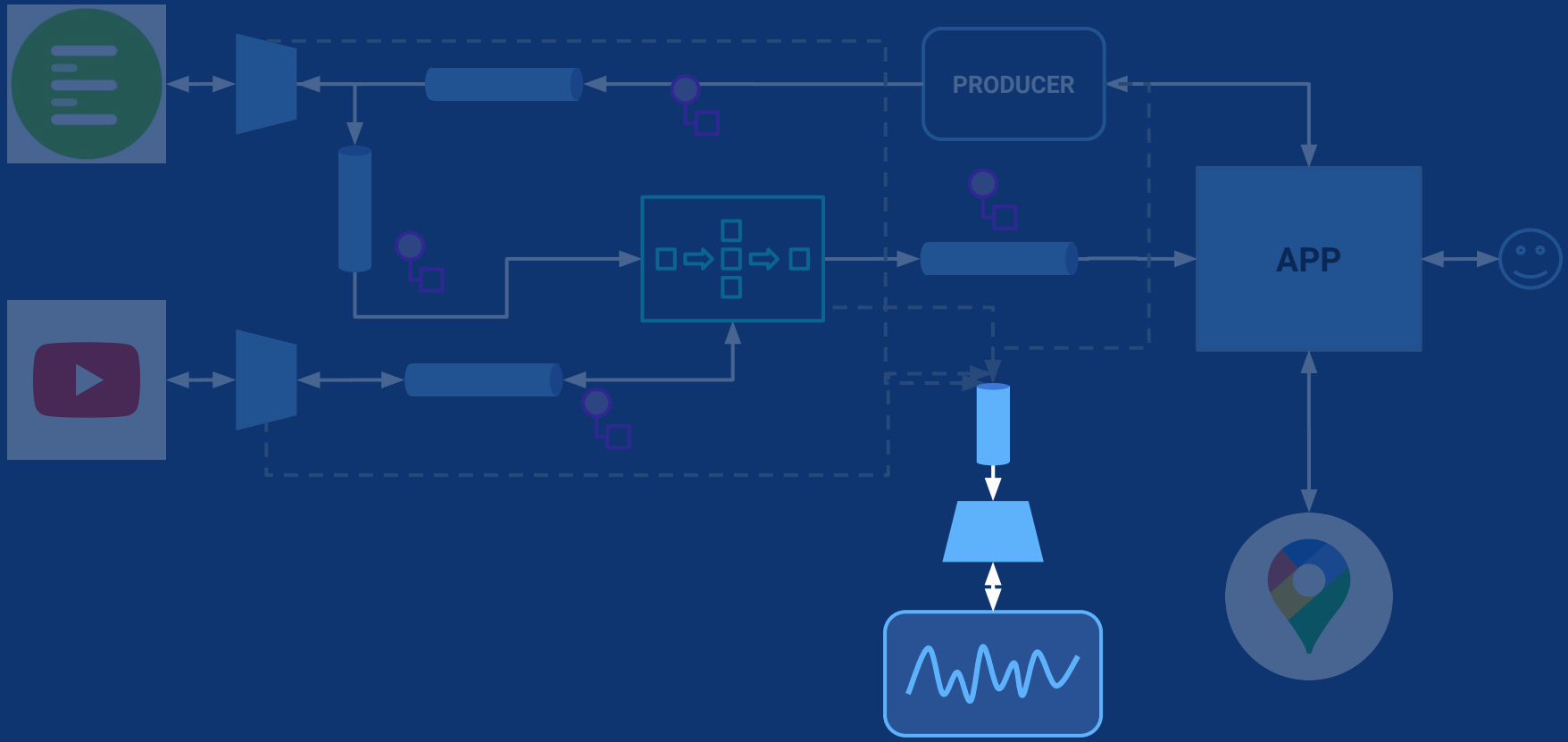


Act as content filters

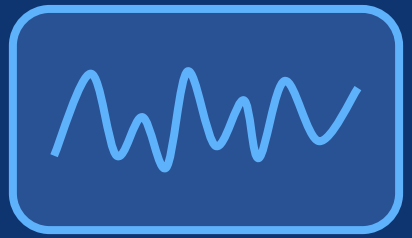
ARCHITECTURE IN TERMS OF EIP



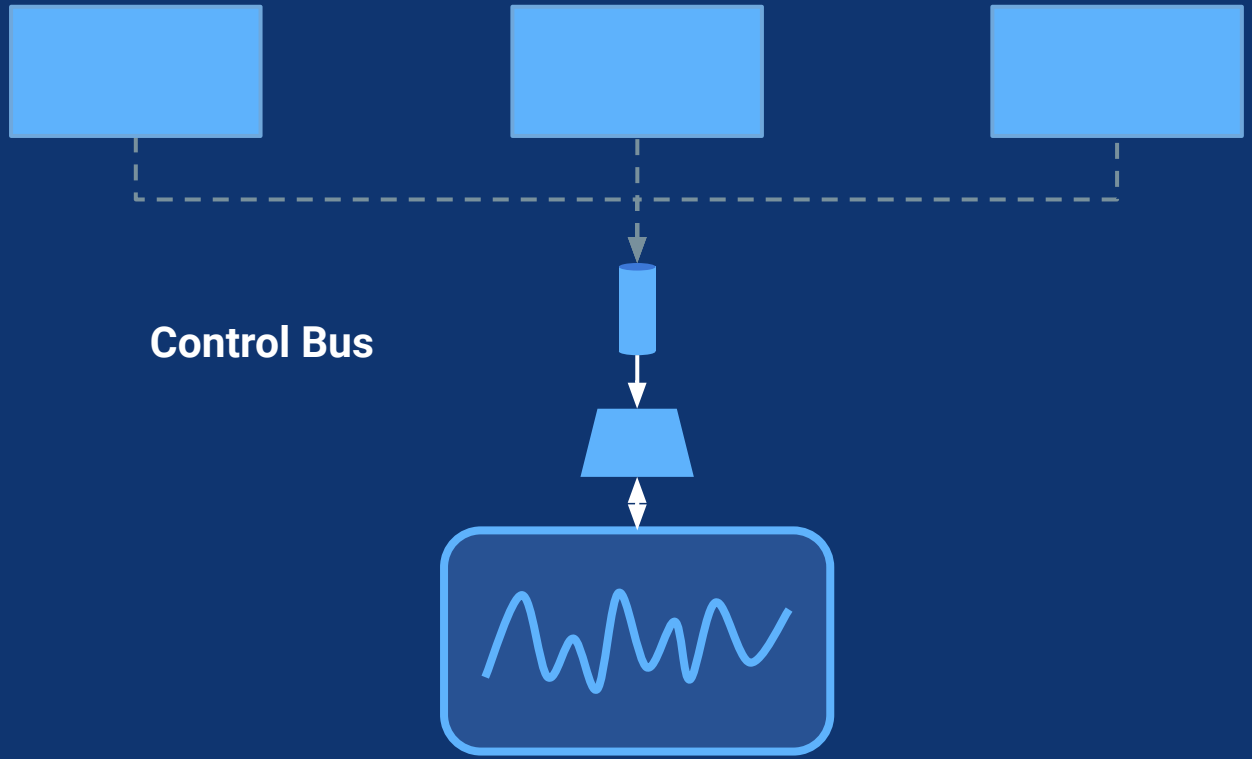
ARCHITECTURE IN TERMS OF EIP



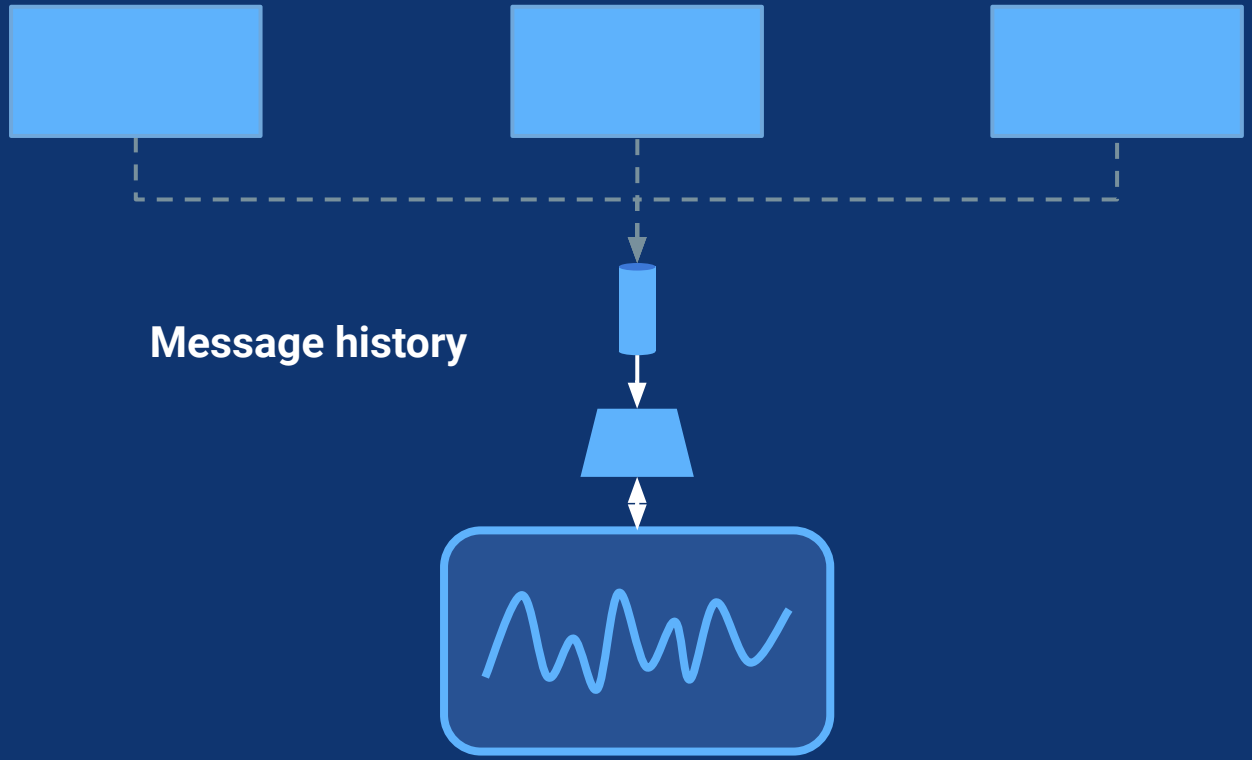
MONITORING



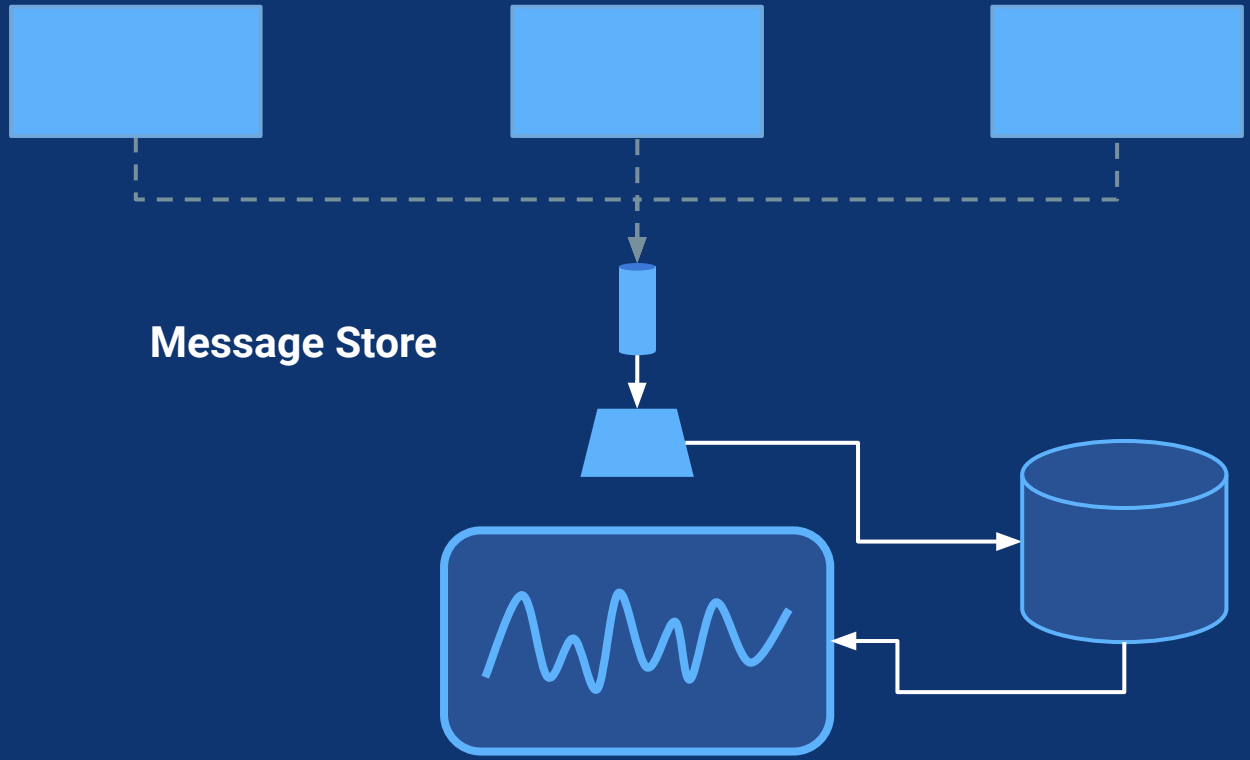
MONITORING



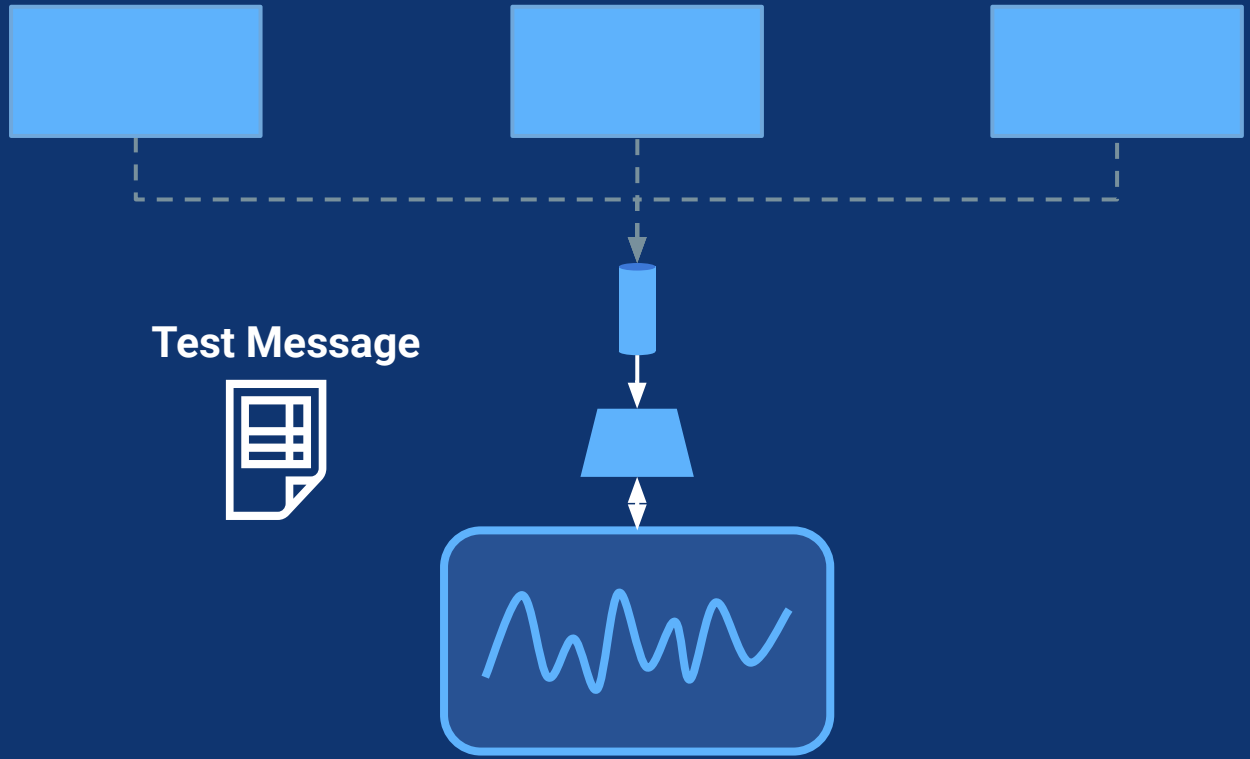
MONITORING



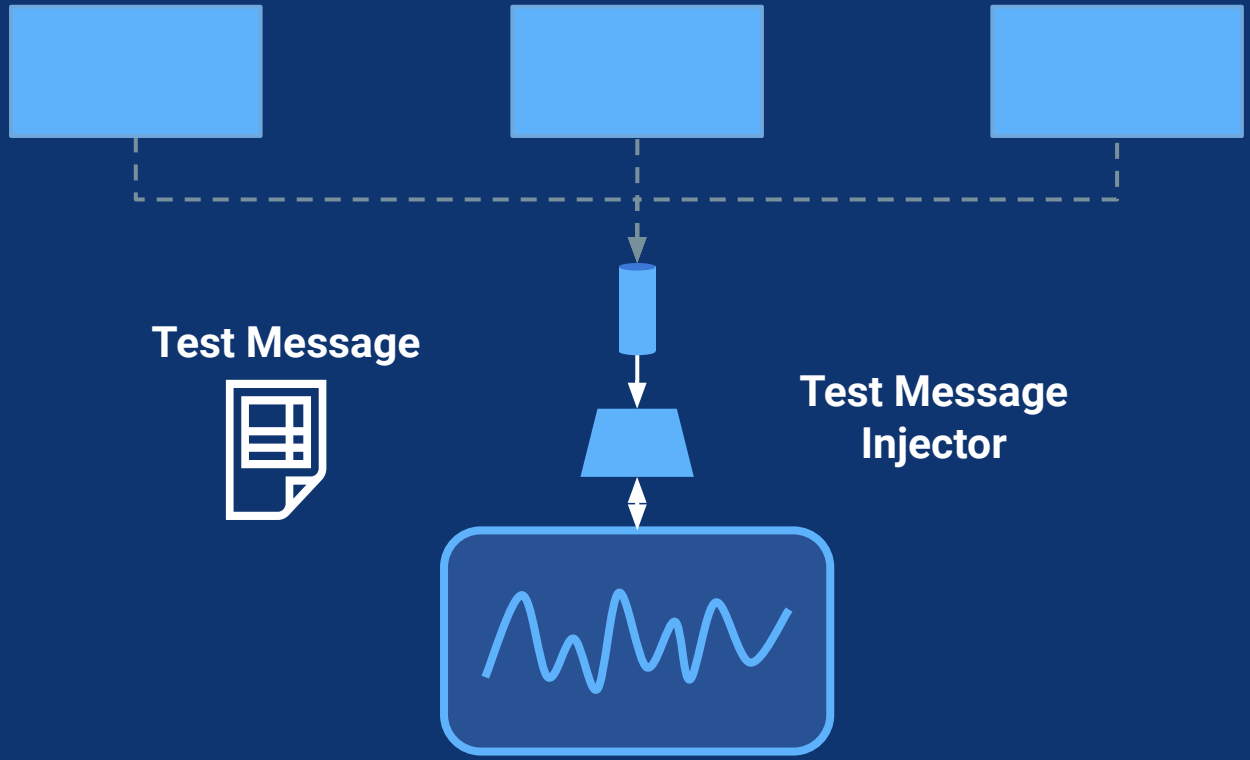
MONITORING



MONITORING

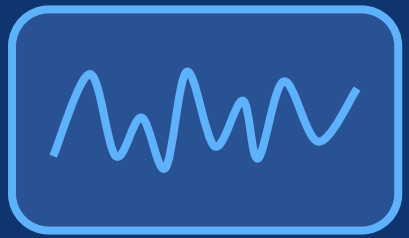


MONITORING



MONITORING

General Queue Metrics are scraped from the message broker



MAPPING PATTERNS TO IMPLEMENTATION

- P2P and publish subscribe channels
- Adapters: producer/ consumer, filter
- Messages
- Composed Message Processor
- Control Bus
- Message History
- Message Store
- Test Message

MAPPING PATTERNS TO IMPLEMENTATION

P2P and publish subscribe channels

Aggregator.java

```java
@JmsListener(destination = "from-yt-consumer-queue")
public void processMessageFromYTconsumer(VideoID message) {
    try {
        System.out.println("Agg got from YT cons "+ message.getVideo_id());
        List<String> history = message.getHistory();
        history.add(COMPONENT_NAME);
        message.setHistory(history);
        if (messageBuffer.containsKey(message.getMsg_id())){
            messageBuffer.get(message.getMsg_id()).setVid_url(message.getVideo_id());
            sendMessageToQueue(messageBuffer.get(message.getMsg_id()), queueName: "from-agg-to-controller-queue");
            messageBuffer.get(message.getMsg_id()).setType("Video ID message");
            messageBuffer.get(message.getMsg_id()).setTestMessage(message.isTestMessage());
            LogMessage logMessage = new LogMessage(messageBuffer.get(message.getMsg_id()));
            sendMessageToQueue(logMessage, queueName: "topic.control-bus");
            messageBuffer.remove(message.getMsg_id());
        }

    } catch (Exception e) {
        System.out.println("Error agg from youtube reading queue");
    }
}
```

MAPPING PATTERNS TO IMPLEMENTATION

P2P and publish subscribe channels

- **Two publish-subscribe channels** (Topics):
  - topic.control-bus
  - topic.compositeMsg2
- **Four P2P queues:**
  - to-yt-consumer-queue
  - to-aggregator-queue
  - from-yt-consumer-queue
  - from-agg-to-controller-queue

```
                message) {

                + message.getVideo_id());
                ();


                sg_id())){
                .setVid_url(message.getVideo_id());
                essage.getMsg_id()), queueName: "from-agg-to-controller-queue");
                .setType("Video ID message");
                .setTestMessage(message.isTestMessage());
                (messageBuffer.get(message.getMsg_id()));
                me: "topic.control-bus");
                ());
            }

        } catch (Exception e) {
            System.out.println("Error agg from youtube reading queue");
        }
    }
```

MAPPING PATTERNS TO IMPLEMENTATION

Adapters: filter

RecipeAdapter.java

```
String title = jsonNode.get("hits").get(0).get("recipe").get("label").asText();
String picture = jsonNode.get("hits").get(0).get("recipe").get("image").asText();
List<String> ingredients = new ArrayList<>();

if (jsonNode.get("hits").get(0).get("recipe").get("ingredientLines").isArray()){
    for(JsonNode item : jsonNode.get("hits").get(0).get("recipe").get("ingredientLines")){
        ingredients.add(item.asText());
    }
}
```

MAPPING PATTERNS TO IMPLEMENTATION

Composed Message Processor

ComposedMessageProcessor.java

```
@JmsListener(destination = "to-aggregator-queue")
public void processMessageFromQueue(ResponseMessage message)
```

Receives messages from recipe adapter

```
System.out.println("Agg got from Recipe consumer title"+message.getTitle());
messageBuffer.put(message.getMsg_id(), message);
```

“Routes” message to aggregator storage

```
VideoID videoObj = new VideoID(message.getMsg_id(), message.getTitle(), history);
sendMessageToQueue(videoObj, queueName: "to-yt-consumer-queue");
```

Extracts recipe title and message id and routes it to youtube adapter

```
@JmsListener(destination = "from-yt-consumer-queue")
public void processMessageFromYTconsumer(VideoID message) {
```

Receives messages from youtube adapter

MAPPING PATTERNS TO IMPLEMENTATION

Composed Message Processor

ComposedMessageProcessor.java

```
if (messageBuffer.containsKey(message.getMsg_id())){
    messageBuffer.get(message.getMsg_id()).setVid_url(message.getVideo_id());
    sendMessageToQueue(messageBuffer.get(message.getMsg_id()), queueName: "from-agg-to-controller-queue");
```

Correlates messages by id and sends them to the controller queue

MAPPING PATTERNS TO IMPLEMENTATION

Messages

```
public class RequestMessage extends Message implements Serializable{

    private String id;
    private String term;
    private List<String> history;
}
```

```
public class VideoID extends Message implements Serializable{
    private String msg_id;
    private String video_id;
    private List<String> history;
}
```

MAPPING PATTERNS TO IMPLEMENTATION

Messages

```
public class RequestMessage extends Message implements Serializable{

    private String id;
    private String term;
    private List<String> history;
}
```

```
public class VideoID extends Message implements Serializable{
    private String msg_id;
    private String video_id;
    private List<String> history;
}
```

5 types of messages

MAPPING PATTERNS TO IMPLEMENTATION

Messages: History

```
public class RequestMessage extends Message implements Serializable{

    private String id;
    private String term;
    private List<String> history;
}
```

```
public class VideoID extends Message implements Serializable{
    private String msg_id;
    private String video_id;
    private List<String> history;
}
```

5 types of messages

History is kept and updated by each component

MAPPING PATTERNS TO IMPLEMENTATION

Control bus

- Apache active mq topic
- Monitor Adapter is the only subscriber
- All other components are producers

MAPPING PATTERNS TO IMPLEMENTATION

Control bus

- Apache active mq topic
- Monitor Adapter is the only subscriber
- All other components are producers

MAPPING PATTERNS TO IMPLEMENTATION

Message store

- Monitor Adapter writes all log messages to mysql

Control bus topic

-> Monitor Adapter ->

MAPPING PATTERNS TO IMPLEMENTATION

Test Message

- Frontend sends request to /test endpoint specifying which component to test
- TestMessageService.java sends message to the correct queue

-> backend -> produces test message -> sends messages to queue

THANK YOU

CONTRIBUTION

Giouri Kilinkaridis:

1) Frontend
2) Backend (except anything related to Monitoring)

Johanna

- Monitoring components