Integrating with Backend

Backend Services

Part 1: Node JS and MySQL

Jumail Bin Taliba
School of Computing, UTM
June 2020

Watch on YouTube

Set the playback speed 1.5X

See the timestamp in the description

Agenda

- Design Principles and Design Patterns
- Local Development
- Setting Up Databases
- Writing Code for REST API
- Deployment
- Testing with Flutter App

SOLID Principles

- Single responsibility principle
- Open close principle
- Liskov substitution principle
- Interface segregation principle
- Dependency inversion principle

Why care?

- Software development is a collaborative work
- The intention of the principles is to make software easier to maintain and to extend

Design Patterns

What are design patterns?

- Reusable solutions to repeating problems

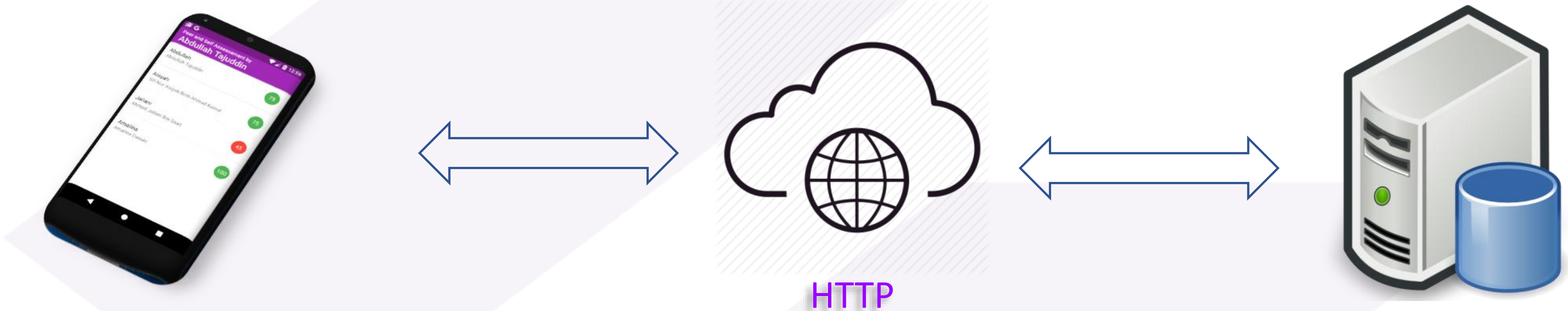
Example of design pattern?

- Creational patterns: Builder, Singleton, Dependency Injection, etc
- Structural patterns: Facade, Adapter, etc
- Behavioural patterns: Command, Memento, State, Iterator, etc
- *Architectural patterns: MVC, MVP, MVVM, etc

System Architecture

Mobile App

Back-end



Programming:
Node, PHP, Python,
Perl,
Deno

Database:
MySQL, MongoDB,
Firebase

MVVM
Architecture

HTTP
REST
GraphQL

MVC
Architecture

Example: Get the profile of a given user

HTTP : http://www.mywebsite.com/getprofile.php? uid=1213

REST : GET http://www.mywebsite.com/profiles/1213

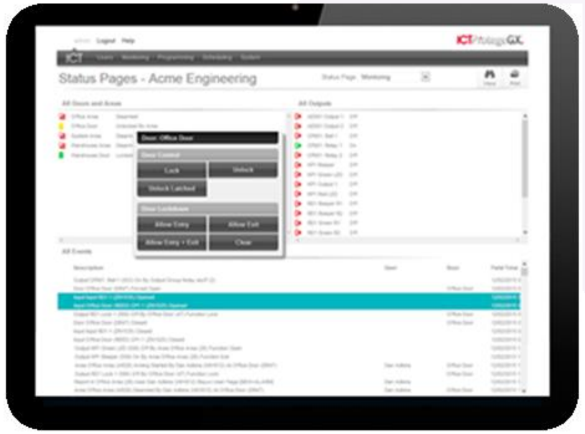
GraphQL:

```
query{
    User( uid: 1213){
        fullName
        role
    }
}
```

Back-end Architecture

Model View Controller (MVC)

Front-end



1. Client Request

4. Server Response

Back-end

Controller

- Handles client requests: GET, POST, etc
- Gets data from the models and passes data to the views
- Gets presentations from the views and passes to the client

2. Get Data

Model

- Handles data related logic
- Talks to the database, e.g SELECT, INSERT, UPDATE, etc

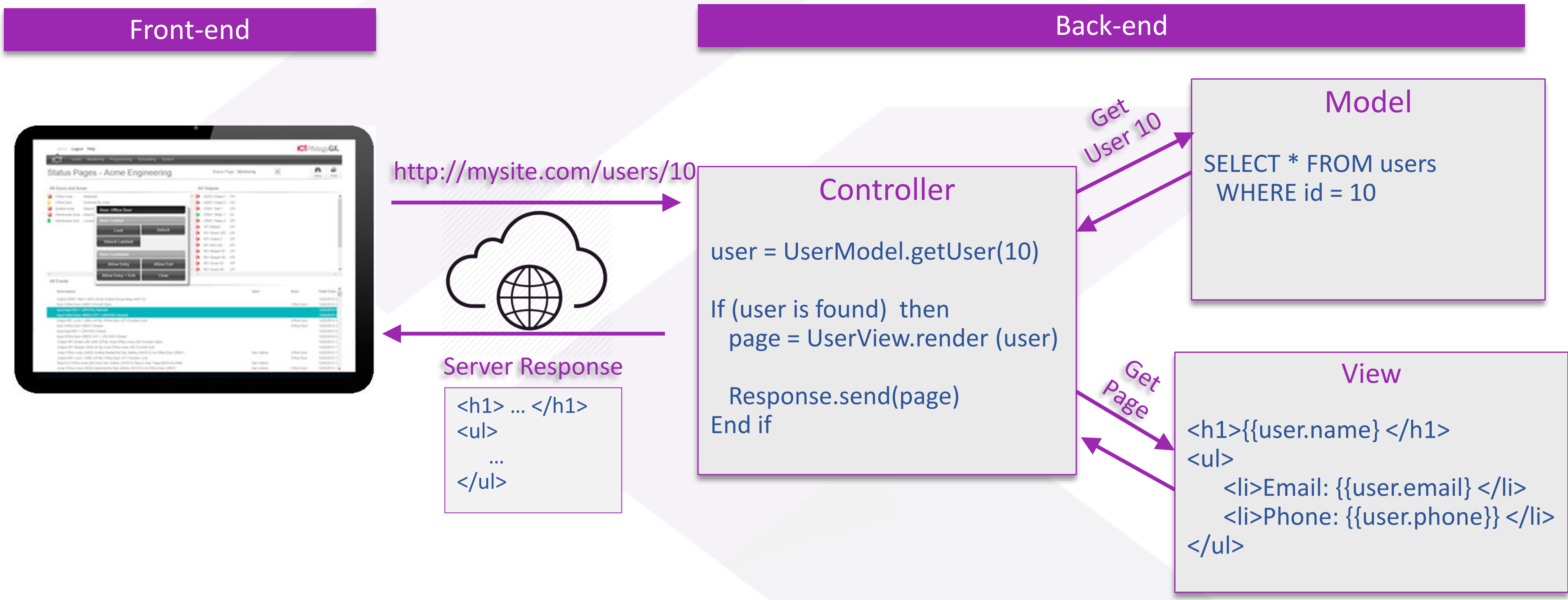
3. Get Presentation

View

- Handles data presentation
- Dynamic UI / pages
- Template engines

Back-end Architecture (2)

Example: Show user profile



```
<h1> … </h1>
<ul>
     …
</ul>
```

Controller

```
user = UserModel.getUser(10)

If (user is found)  then
   page = UserView.render (user)

   Response.send(page)
End if
```

Model

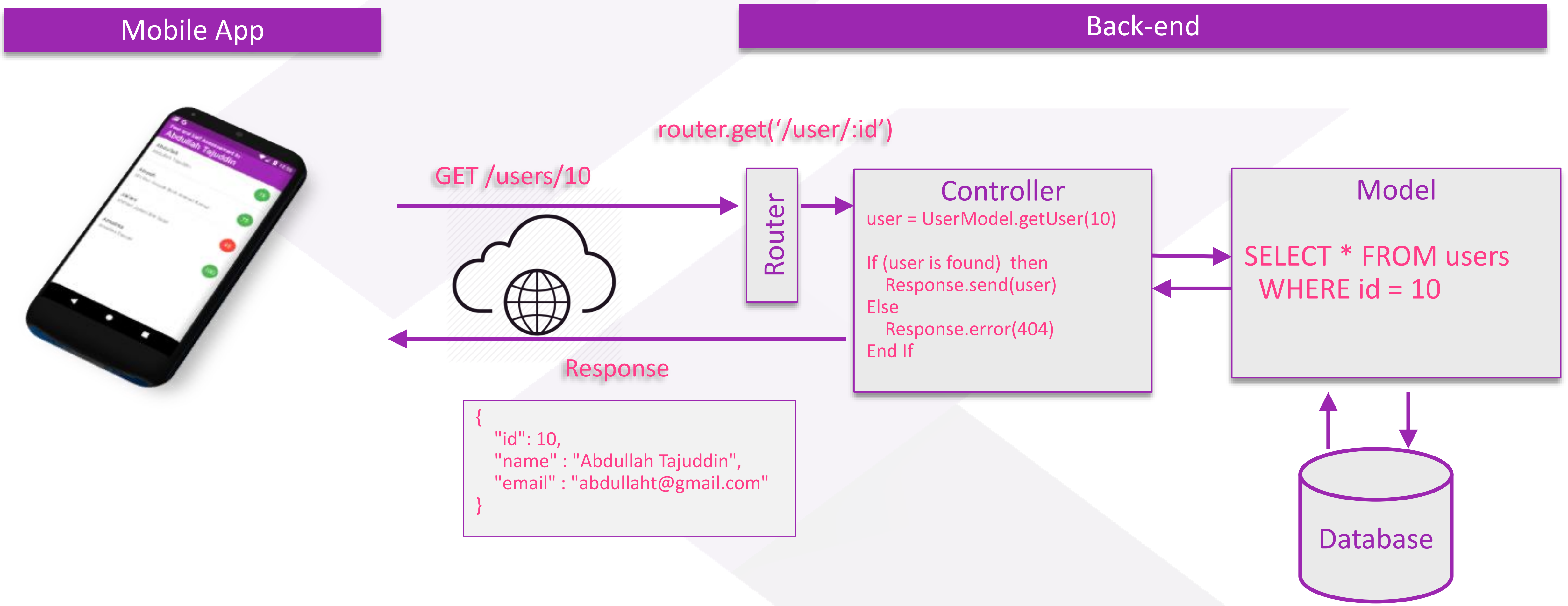
```
SELECT * FROM users
  WHERE id = 10
```

View

```
<h1>{{user.name} </h1>
<ul>
     <li>Email: {{user.email} </li>
     <li>Phone: {{user.phone}} </li>
</ul>
```

Back-end Architecture (3)

Our architecture for REST API Server



```
user = UserModel.getUser(10)

If (user is found)  then
   Response.send(user)
Else
   Response.error(404)
End If
```

```
SELECT * FROM users
  WHERE id = 10
```

```
{
   "id": 10,
   "name" : "Abdullah Tajuddin",
   "email" : "abdullaht@gmail.com"
}
```

Project Source Code

backend

https://github.com/jumail-utm/backend_node_mysql

Frontend (Flutter App – Todo List)

https://github.com/jumail-utm/flutter_todo_rest

Setting Up for Local Development

- Install Xampp
  https://www.apachefriends.org/download.html

- Install MySQL Client

  phpMyAdmin

  MySQL Workbench
  https://www.mysql.com/products/workbench

  MySQL Admin (Chrome Extension)
  https://chrome.google.com/webstore/detail/chrome-mysql-admin/ndgnpnpakfcdjmpgmcaknimfgcldechn

- Install Node JS
  https://nodejs.org/en/download

Creating Database

Create database from any MySQL Client tool

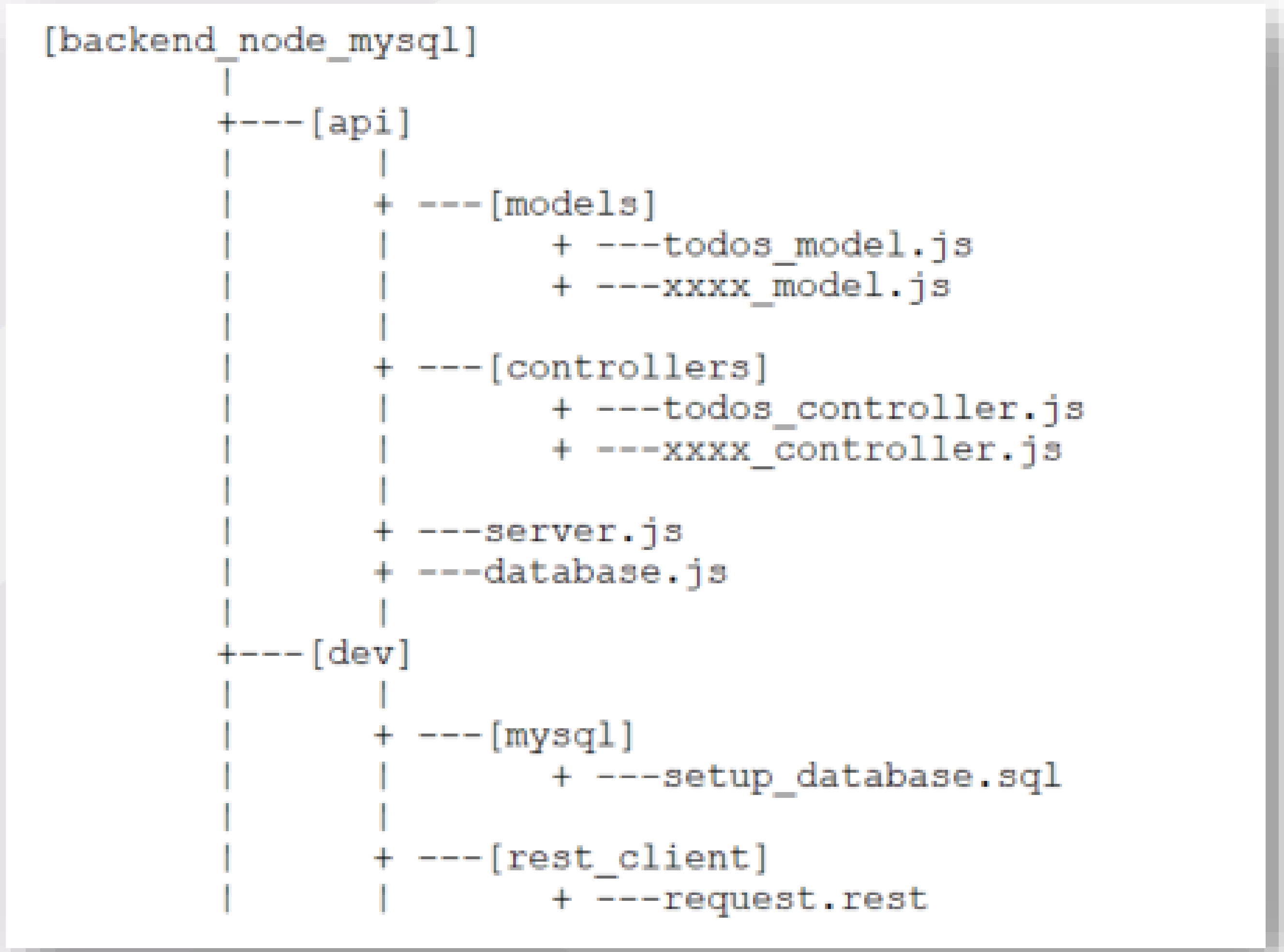
Use SQL script rather than using GUI

Sample script on my github repo

https://github.com/jumail-utm/backend_node_mysql/blob/master/dev/mysql/setup_database.sql

Writing Code for REST API Service with Node JS

- Setup Project
  - Project structure
  - Install dependency packages
- Setup Database connection
- Define model classes
- Define route handlers



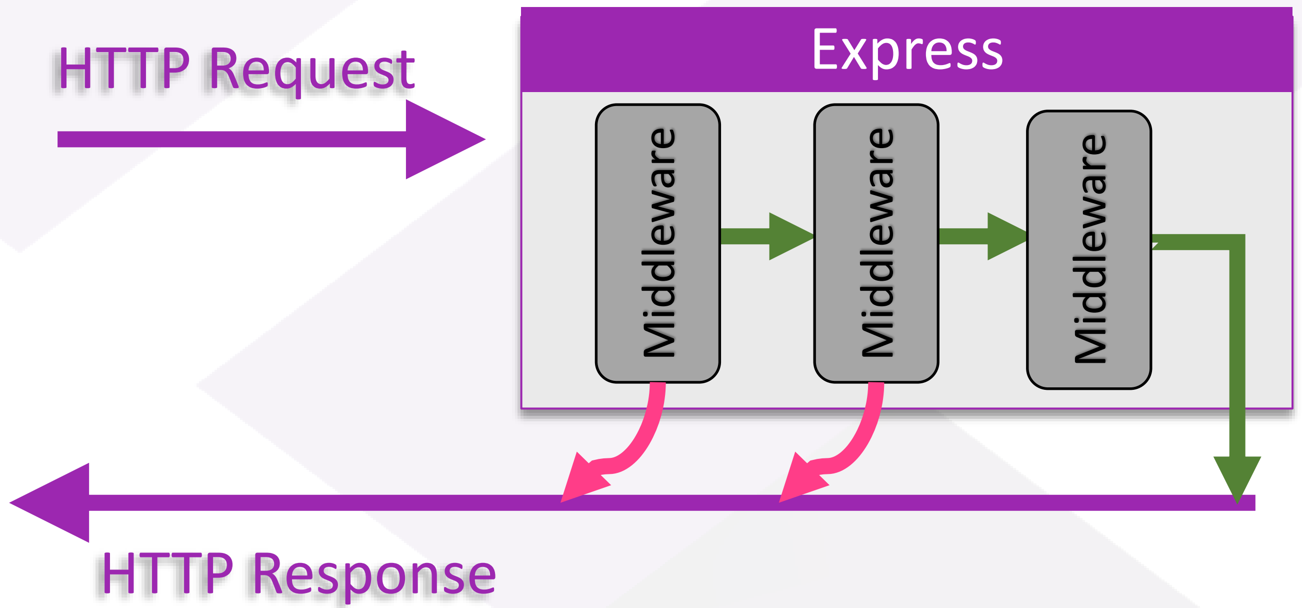
```
[backend_node_mysql]
         |
         +---[api]
         |       |
         |       + ---[models]
         |       |        + ---todos_model.js
         |       |        + ---xxxx_model.js
         |       |
         |       + ---[controllers]
         |       |        + ---todos_controller.js
         |       |        + ---xxxx_controller.js
         |       |
         |       + ---server.js
         |       + ---database.js
         |       |
         +---[dev]
         |       |
         |       + ---[mysql]
         |       |        + ---setup_database.sql
         |       |
         |       + ---[rest_client]
         |       |        + ---request.rest
```

Command line and Code snippet:

https://jumail-utm.github.io/backend_node_mysql/pages/node-mysql-rest-api

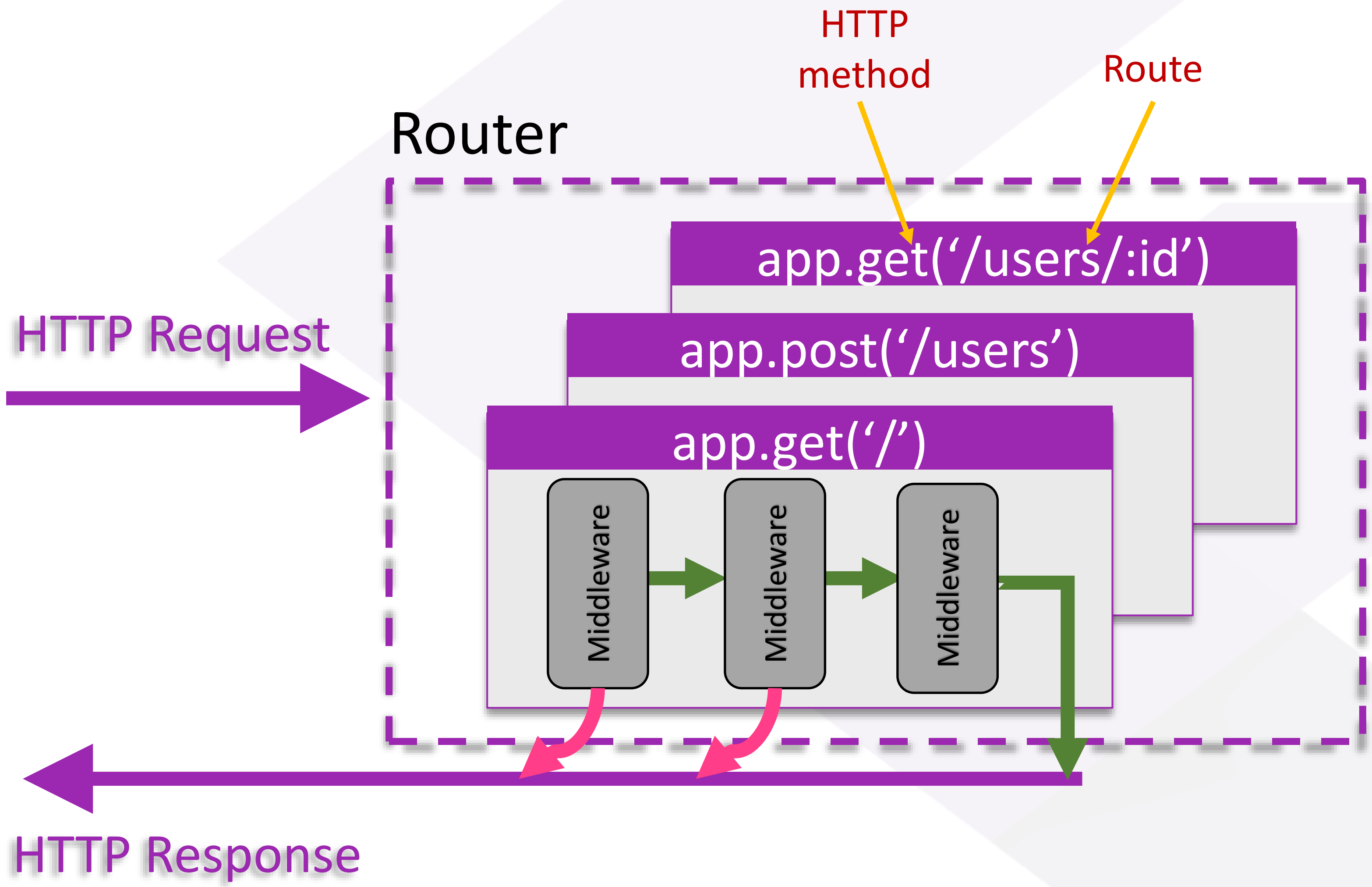
Express JS Middleware

- Express JS is a routing and middleware web framework
- An Express application is a series of function calls (called middlewares) that run between the time of the server gets the request and the time it sends out the response



Express JS Middleware (2)

Terminologies

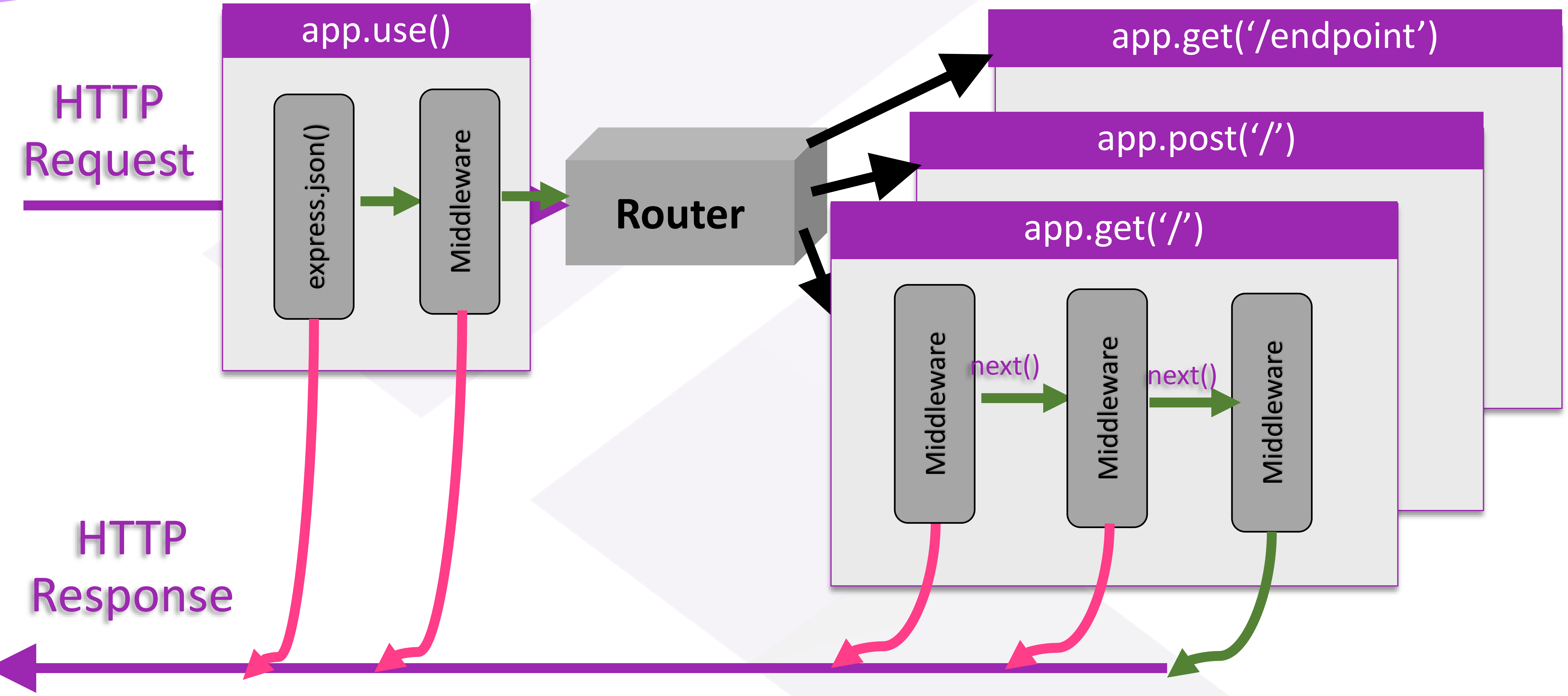


Example express app code

```
const app = express()

app.get( '/', handleRootRoute )

app.post( '/users', handleCreateUser )

app.get( '/users/:id', (req, res, next ) => {
    // code are excluded for brevity
} )

function handleRootRoute(req, res, next){
    // code are excluded for brevity
}

function handleCreateUser(req, res, next){
    // code are excluded for brevity
}
```

Express JS Middleware (3)

How it works?



Express JS Middleware (4)

Middleware Functions

Each middleware function accepts three parameters

- req: Request data from the client
- res: Server response to the client
- next: function to execute the next middleware

```
const app = express()

app.post( '/users', handleCreateUser )

app.get( '/users/:id', (req, res, next ) => {
    // code are excluded for brevity
} )

function handleCreateUser(req, res, next){
    // code are excluded for brevity
}
```

Express JS Middleware (5)

Passing Data To Next Middleware

- A middleware can pass data to the next one by injecting the data to req or res

- Not by passing parameter to next()

So the following does not really work

```
function firstMiddleware(req, res, next){
     next( 10 )
}
```

Example

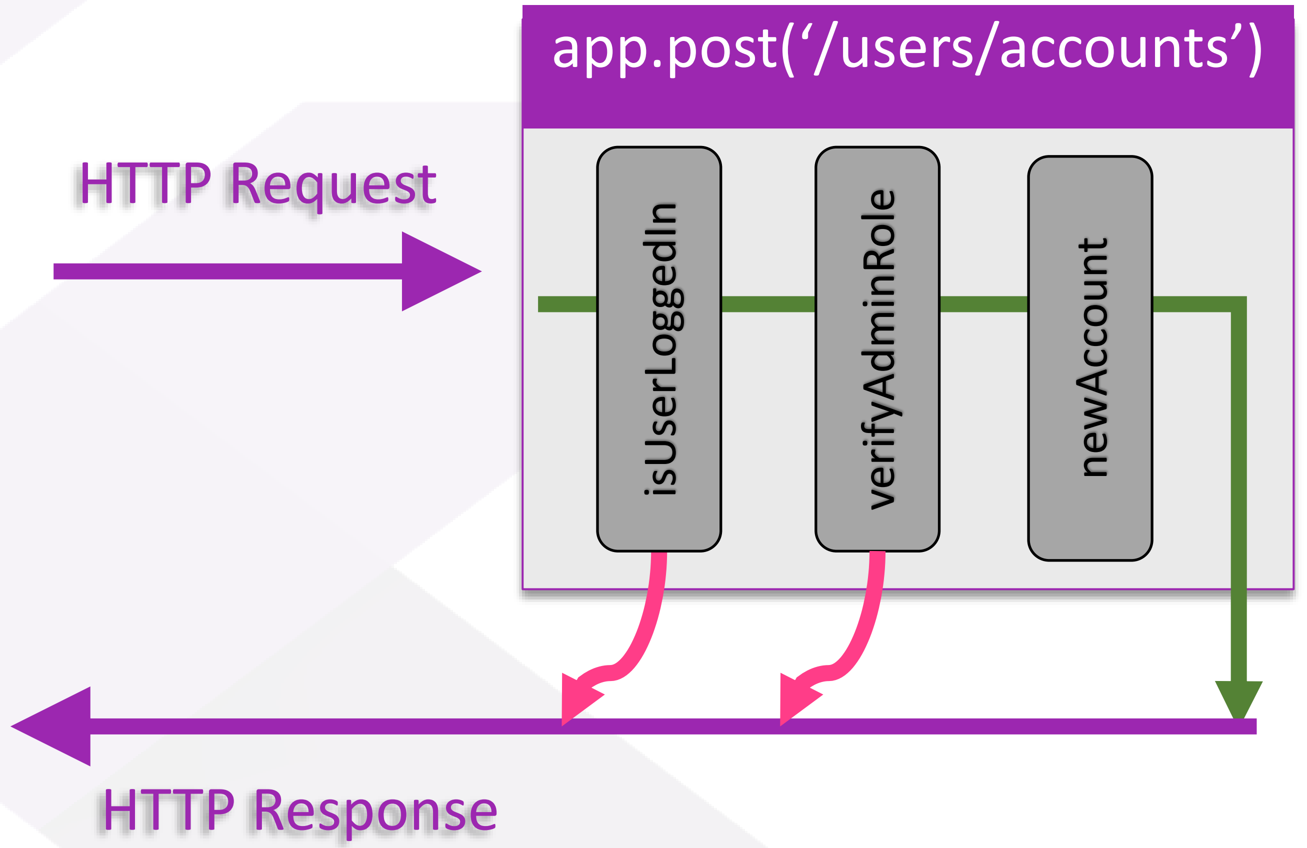
```
app.get( ‘/anyroute’,  firstMiddleware,
                       secondMiddleware,
                       lastMiddleware  )

function firstMiddleware(req, res, next){
   req.firstValue = 10
   next()
}

function secondMiddleware(req, res, next){
   req.secondValue = 20
   next()
}

function lastMiddleware(req, res, next){
   const a  = req.firstValue
   const b  = req.secondValue
   console.log( ‘result = ‘, a + b)
}
```

Express JS Middleware (6)

Chaining Middleware

- Practical implementation of express is by middleware chaining
- It promotes modularity
- Large tasks can be splitted into smallers ones
- Each middleware can focus on its specific task

Example: Register a new account



Express JS Middleware (7)

Example

```
app.post( '/users/accounts',  isUserLoggedIn,
                              verifyAdminRole,
                              newAccount    )

function isUserLoggedIn(req, res, next){
  const user = req.user
  if (!user) return res.sendStatus(401)  // unauthorized
  next()
}

function verifyAdminRole(req, res, next){
  const user = req.user
  if (user.role !== ADMIN_ROLE) return res.sendStatus(403)  // Forbidden
  next()
}

function newAccount(req, res, next){
  const newAccountData = req.body

  // Code for creating a new account goes here
}
```

Deploy on Heroku for Production

- Sign up for an account on Heroku
  https://signup.heroku.com
- Install the Heroku CLI tools
  https://devcenter.heroku.com/articles/heroku-cli
- Create Heroku Apps
  https://dashboard.heroku.com/apps
- Create ClearDB My SQL
  https://jumail-utm.github.io/backend_node_mysql/pages/heroku-setup-mysql
- Deploy Node JS project to Heroku – Prepare, Deploy and Troubleshoot
  https://jumail-utm.github.io/backend_node_mysql/pages/heroku-deploy-node
- Test the REST API Server

Project Source Code

backend

https://github.com/jumail-utm/backend_node_mysql

Frontend (Flutter App – Todo List)

https://github.com/jumail-utm/flutter_todo_rest

Summary

- Setting up databases
- MVC pattern for REST API
- Writing REST API service with Node JS and MySQL
- Deploy Node apps to Heroku

Integrating with Backend

Backend Services

Part 2: REST Service on Firebase

Jumail Bin Taliba
School of Computing, UTM
June 2020

Watch on YouTube

Set the playback speed 1.5X

See the timestamp in the description

Agenda

- Architecture Setups
- Introduction to Firebase
- Setting Up Local Firebase
- Developing REST Service
- Deploying to Firebase

Introduction

Architecture Setups

Each application has several types of code:

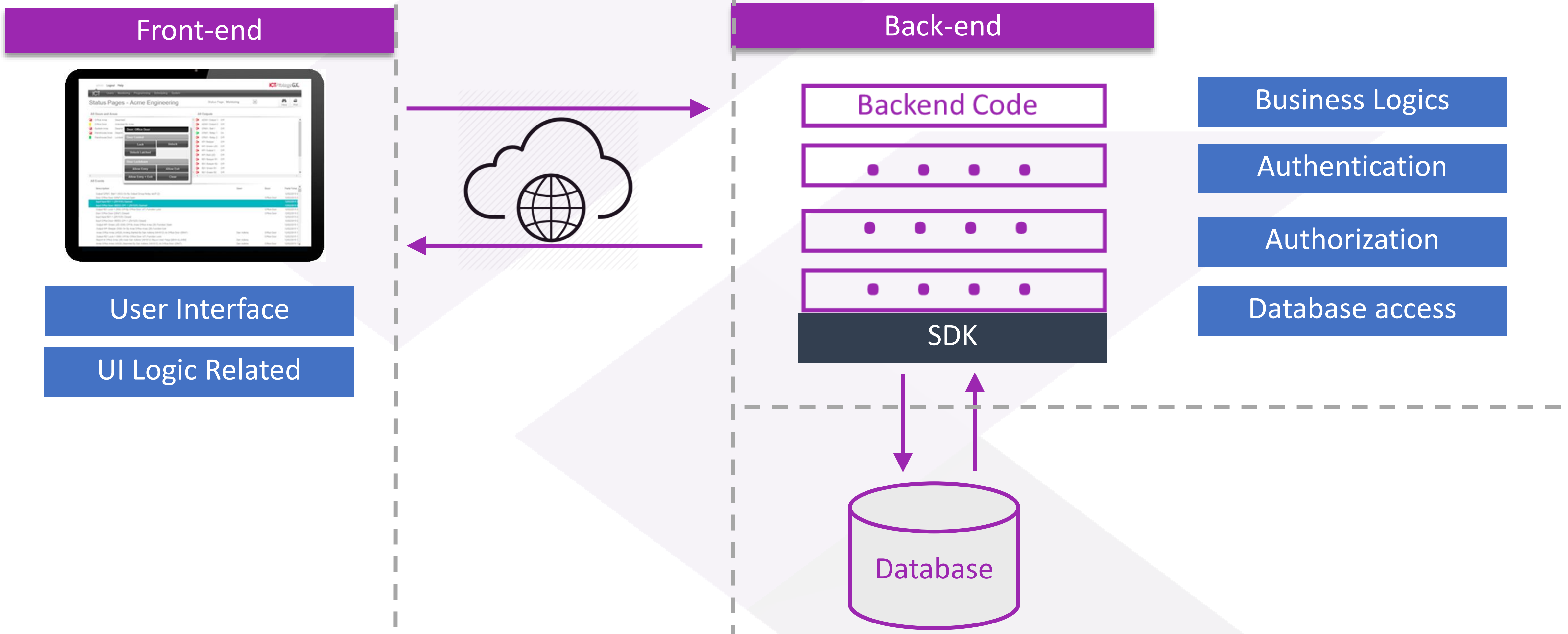
Task	Example
UI or presentation	Show screen, manage layout, etc
Presentation logic related	Conditional UI, state management, etc
Authentication	Verify who a user is
Authorization	Verify what a user has access to
Database related	CRUD operations
Local resource related	Access to device's camera, local files, sensors, etc
API access	Geo location API

Question: Where should you put these code?

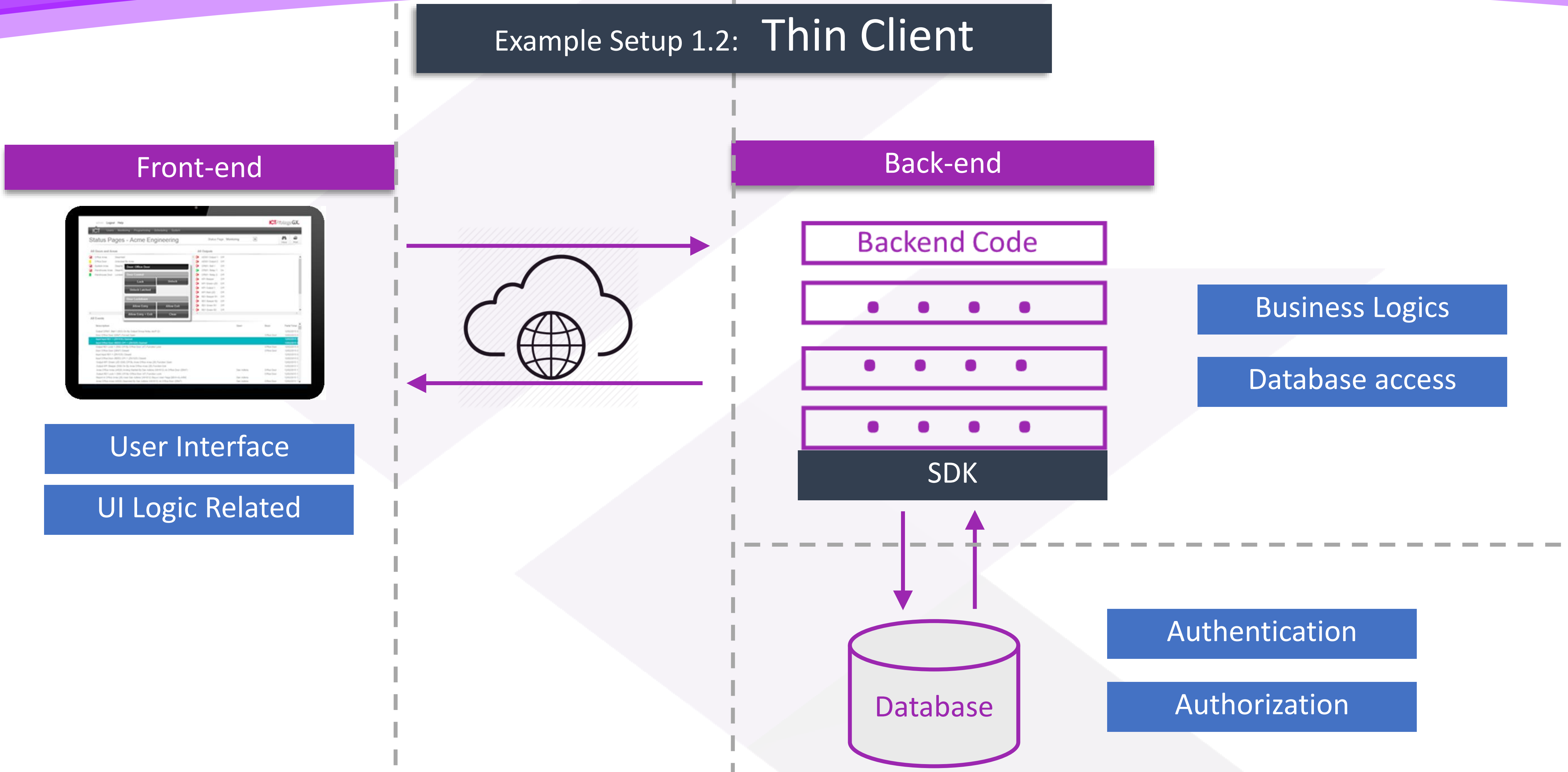
Architecture Setups (2)

Example Setup 1.1: Thin Client

e.g. web-based apps, mobile apps
REST API service, Firebase apps, etc



Architecture Setups (2)

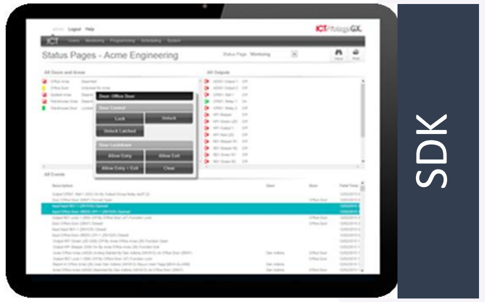


Architecture Setups (3)

Example Setup 2.1: Thick Client

e.g. desktop apps, Java apps, .Net apps, Firebase apps, mobile apps, etc

Front-end



Business logics

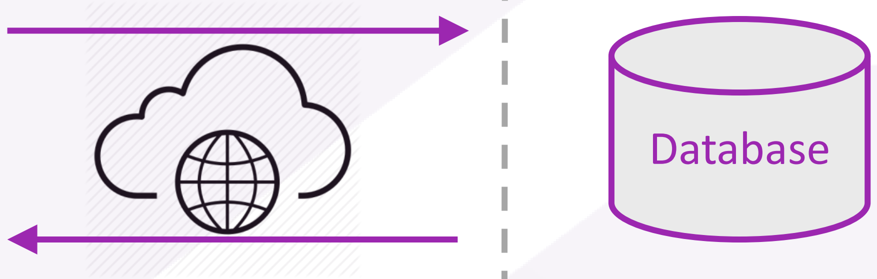
User Interface

UI Logic Related

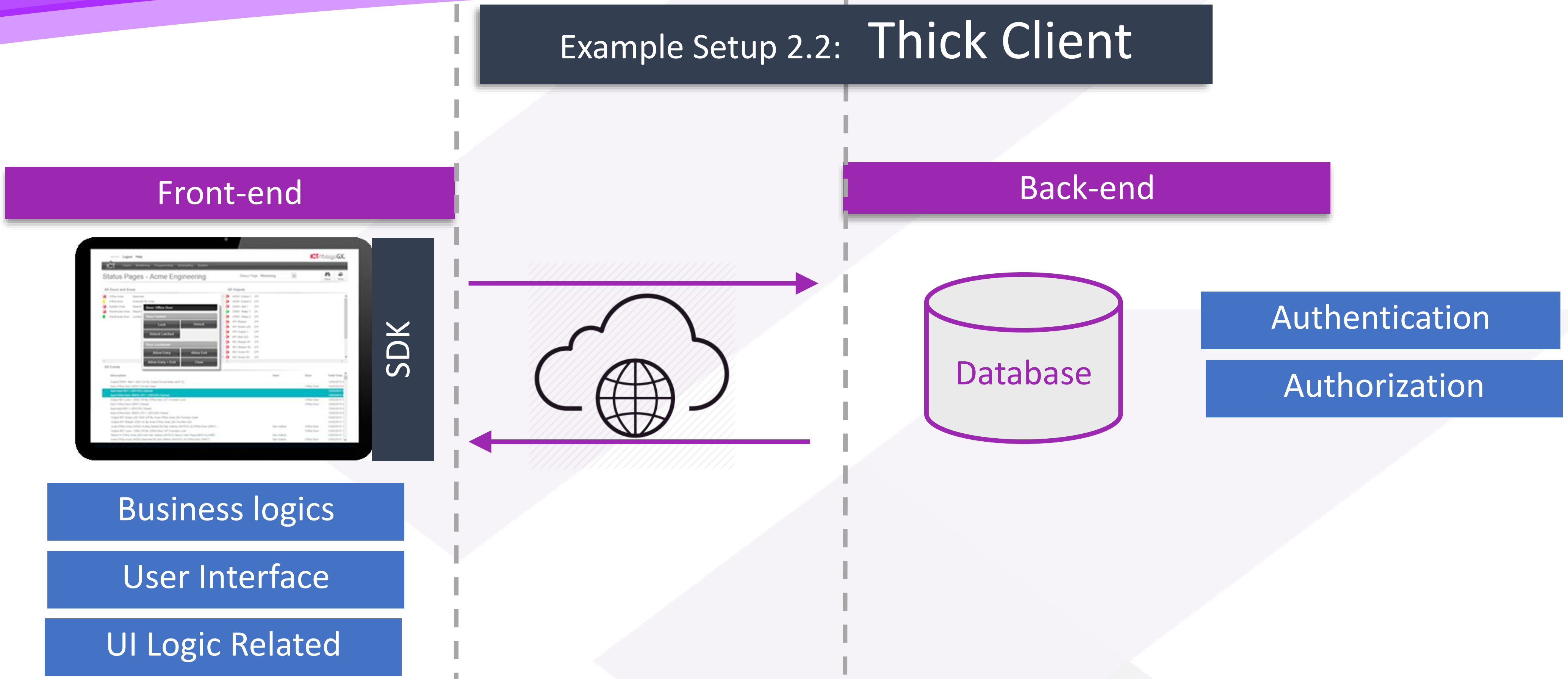
Authentication

Authorization

Back-end



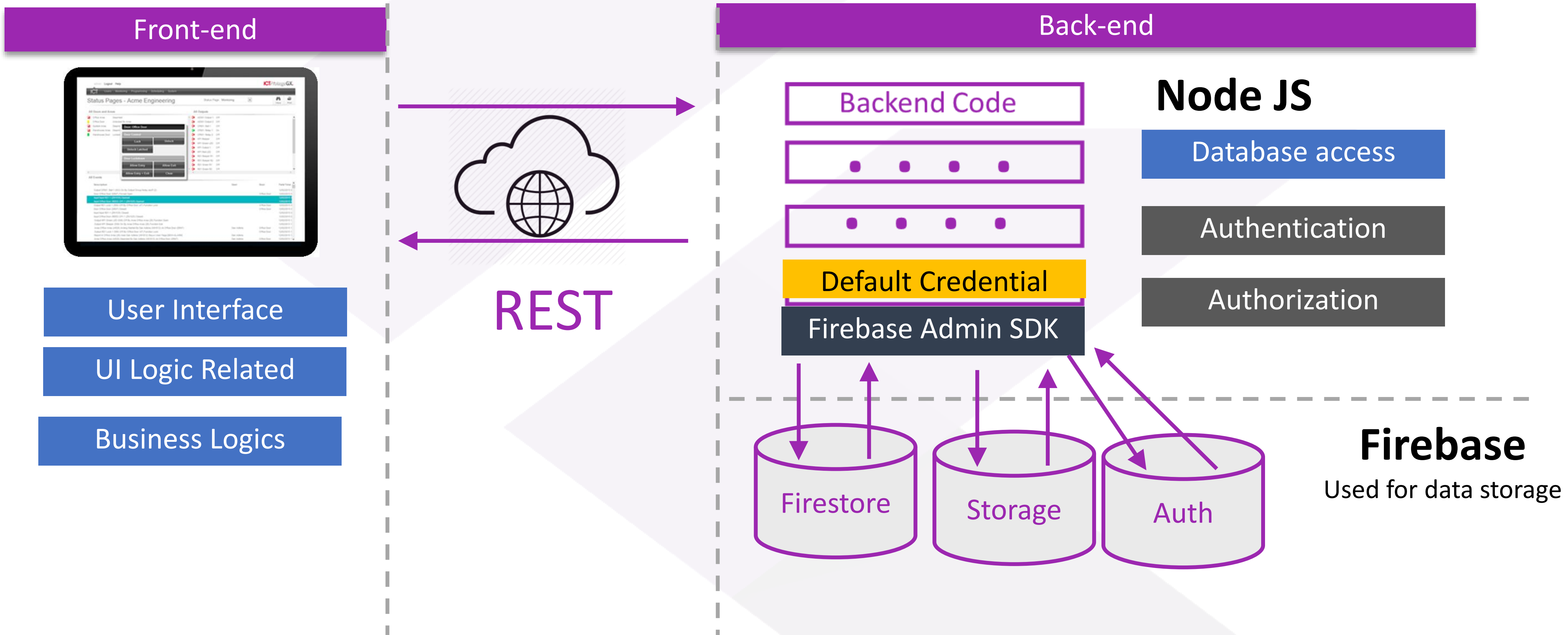
Architecture Setups (3)



Architecture Setups (4)

Firebase Setup for REST API Service (Example 1)

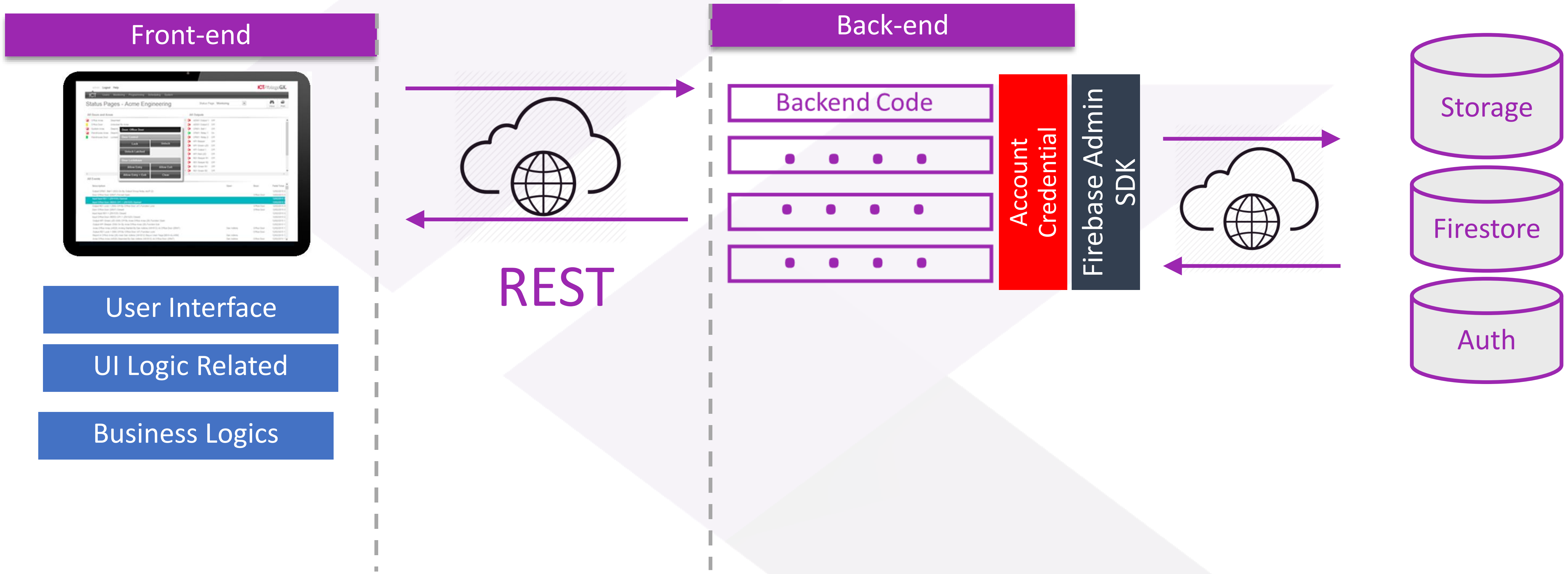
Backend code and Firebase are at the same environment



Architecture Setups (4)

Firebase Setup for REST API Service (Example 2)

Backend code and Firebase are at different environment



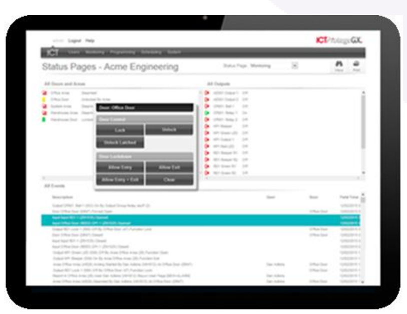
Architecture Setups (5)

Firebase Setup for Client App (Example 1)

Front-end

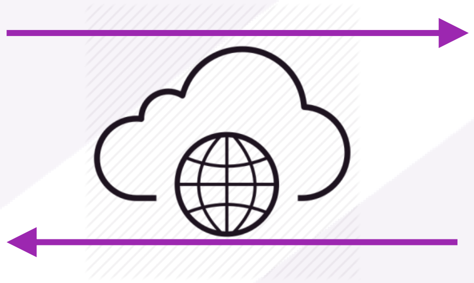
e.g.
SPA web app,
Flutter app,
mobile app

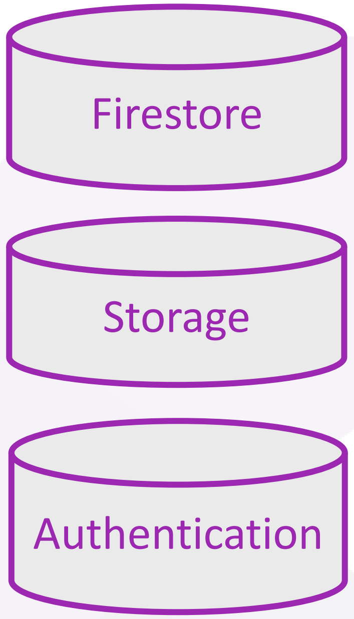
Back-end



Firebase Config

Firebase SDK





Firebase Security Rules

To control access to firebase resources

User Interface

UI Logic Related

Business Logics

One advantage:

Allow server push

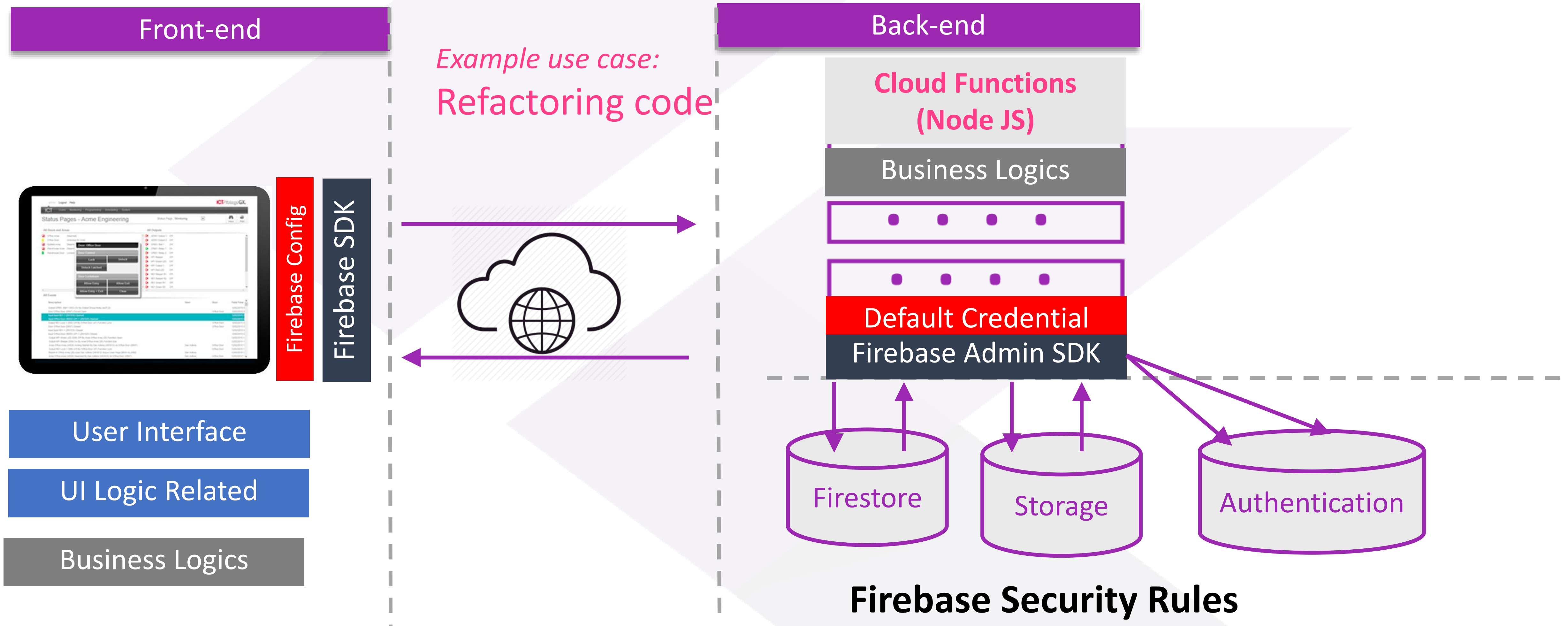
Architecture Setups (6)

Example of Firebase Security Rules

```
rules_version = '2';
service cloud.firestore {
match /databases/{database}/documents {
    // Some examples of firestore rules:
    // Always allow read and write
    match /{document=**} {
     allow read, write;
    }

    // Allow only signed-users to have access
    match /{document=**} {
      allow read, write: if request.auth.uid != null ;
    }

    // Allow only signed-in users can post comments,
    //   a comment can only be edited by the user who posted it,
    //   and anyone can read comments

    match /comments/{commentId}/{userId} {
      allow create: if request.auth.uid != null;
      allow write: if request.auth.uid == userId;
      allow read;
    }
}}
```

Architecture Setups (7)

Firebase Setup for Client App (Example 2)



Introduction to Firebase

- What is Firebase?
  - A BaaS solution
  - Backend made easy
- Firebase Services:
  - Database: Firestore, Realtime (document-based NoSQL)
  - Cloud Storage
  - Authentication
  - Authorization
  - Web Hosting
  - Cloud Functions
  - Cloud Messaging
  - Many more ….

What is NoSQL?

- “Not Only SQL”
- Non-Relational database
- No schema
- Allow unstructured data
- No normalization
- Does not guarantee data integrity and consistency
- Types of NoSQL databases:
  - Document-based *(e.g. Firebase’s Firestore and Realtime, MongoDB)*
  - Column-based *(e.g. Apache Cassandra)*
  - Graph-based *(e.g Neo4J)*

SQL vs Document-based NoSQL

SQL

A database contains a list of **tables**
A table contains a list of **records**
A table may have relationships to other tables

Table Users

uid	name	email
11	Abdullah Razali	abdullah.r@gmail.com
53	Ali Bakar	ali2020@gmail.com
211	Siti Aminah Rashid	saminahr@gmail.com

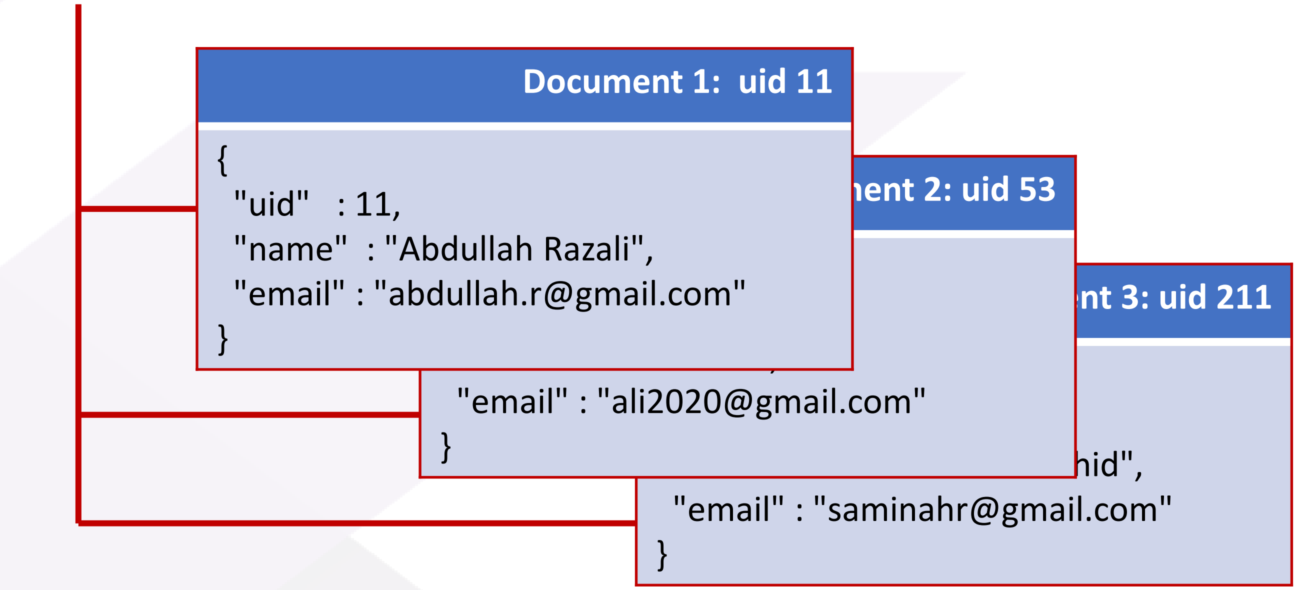
Example query:

```
SELECT * FROM users
WHERE uid = 11
```

Document-based NoSQL

A database contains a list of **collections**
A collection contains a list of **documents**

Collection Users



```
{
  "uid"   : 11,
  "name"  : "Abdullah Razali",
  "email" : "abdullah.r@gmail.com"
}
```

```
  "email" : "ali2020@gmail.com"
}
```

```
hid",
  "email" : "saminahr@gmail.com"
}
```

Example query (in Firebase):

```
db.collection('users').doc('11').get()
```

What are Cloud Functions?

- Functions run on Google Cloud Platform, e.g. on Firebase
- Serverless: No server management is required
- Event-driven: a function executes when certain event occurs

Source	Event	Use case example
Firestore	onCreate, on Update, onDelete, onWrite	Send a push notification to users when a new data created
Cloud Storage	onChange [on Objects]	Resize and convert format of an avatar
Authentication	onCreate, onDelete	Create a new document for newly registered user
HTTP	onRequest, onCall	REST API call, Callable functions – Refactoring some client-side code to server-side code
… and many more ….		

How to Write Cloud Functions?

- Install firebase-tools
- Create a firebase project with functions features added

```
$ firebase init functions
```

- Write the code in `./functions/index.js` file for the following tasks:
  - Import Firebase SDK
  - Initialize the SDK with an authorization strategy
  - Export functions to be listened by Firebase

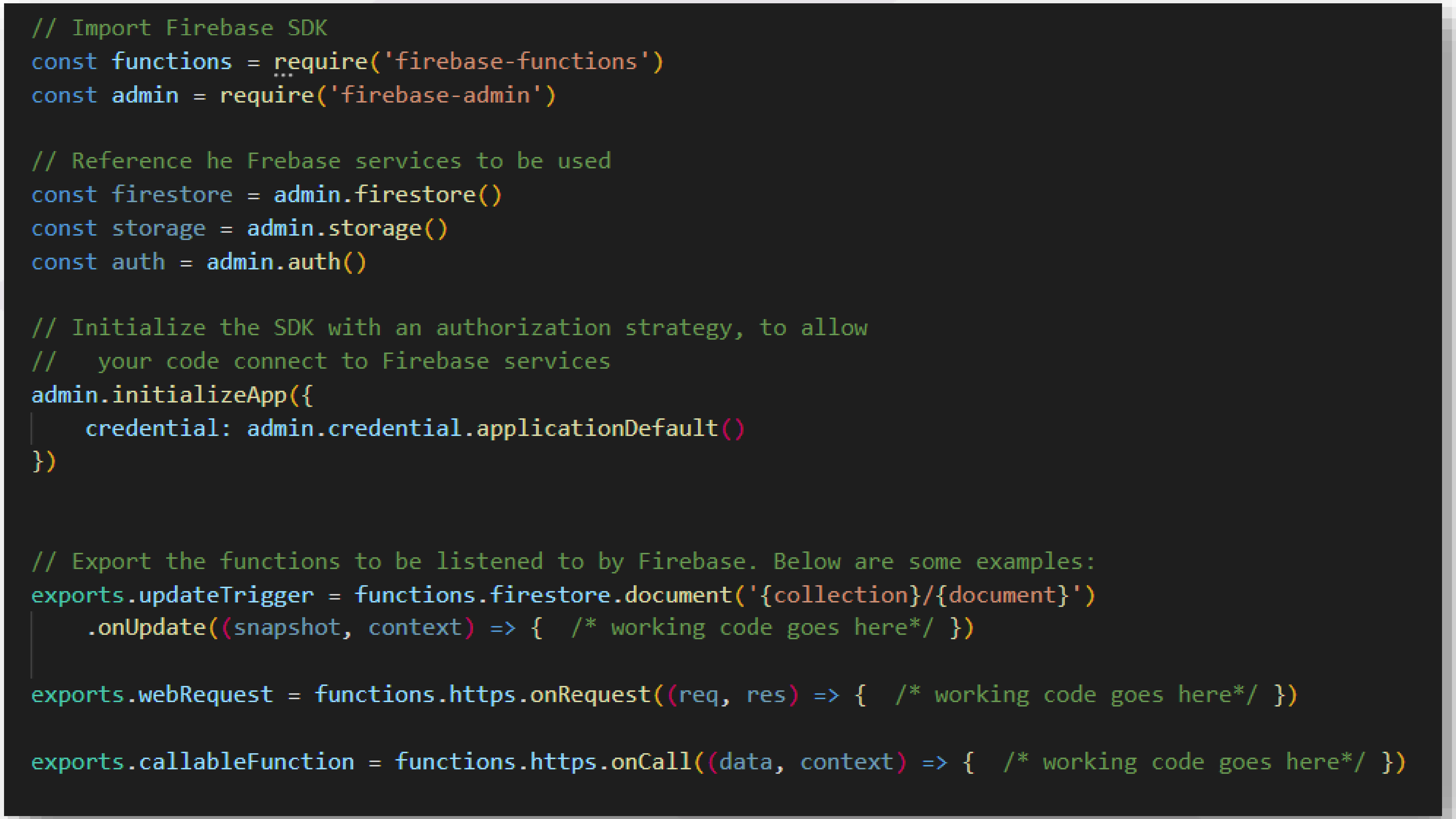
  Example of index.js is in the following slide

- Deploy the functions to Firebase

```
$ firebase deploy --only functions
```

How to Write Cloud Functions? (2)

./functions/index.js

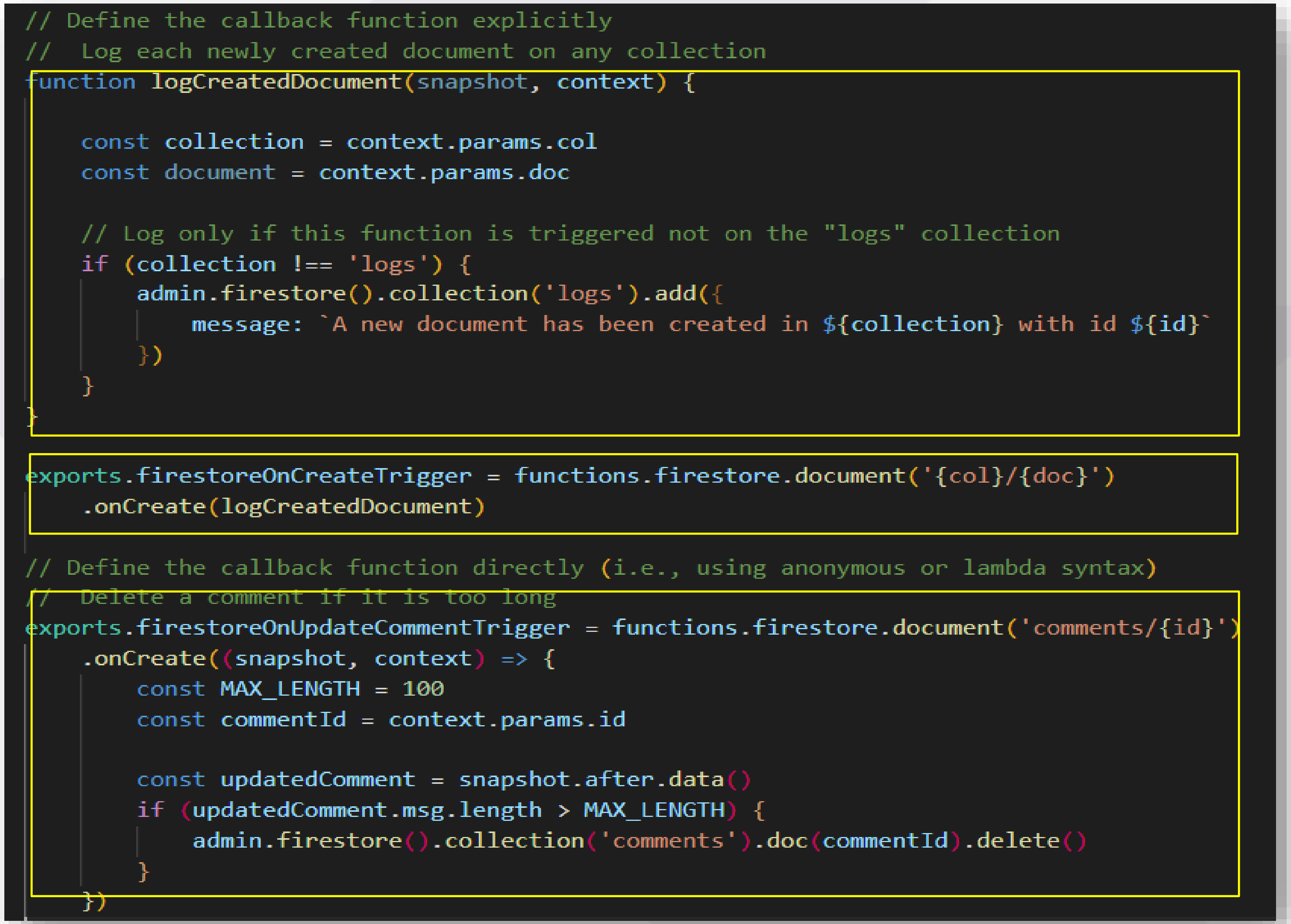


```
// Import Firebase SDK
const functions = require('firebase-functions')
const admin = require('firebase-admin')

// Reference he Frebase services to be used
const firestore = admin.firestore()
const storage = admin.storage()
const auth = admin.auth()

// Initialize the SDK with an authorization strategy, to allow
//   your code connect to Firebase services
admin.initializeApp({
    credential: admin.credential.applicationDefault()
})


// Export the functions to be listened to by Firebase. Below are some examples:
exports.updateTrigger = functions.firestore.document('{collection}/{document}')
    .onUpdate((snapshot, context) => {  /* working code goes here*/ })

exports.webRequest = functions.https.onRequest((req, res) => {  /* working code goes here*/ })

exports.callableFunction = functions.https.onCall((data, context) => {  /* working code goes here*/ })
```

How to Write Cloud Functions? (3)

Example 1: Triggers

No explicit function call required: A trigger will be called automatically by Firebase when the event occurs.



```
// Define the callback function explicitly
//  Log each newly created document on any collection
function logCreatedDocument(snapshot, context) {

    const collection = context.params.col
    const document = context.params.doc

    // Log only if this function is triggered not on the "logs" collection
    if (collection !== 'logs') {
        admin.firestore().collection('logs').add({
            message: `A new document has been created in ${collection} with id ${id}`
        })
    }
}

exports.firestoreOnCreateTrigger = functions.firestore.document('{col}/{doc}')
    .onCreate(logCreatedDocument)

// Define the callback function directly (i.e., using anonymous or lambda syntax)
//  Delete a comment if it is too long
exports.firestoreOnUpdateCommentTrigger = functions.firestore.document('comments/{id}')
    .onCreate((snapshot, context) => {
        const MAX_LENGTH = 100
        const commentId = context.params.id

        const updatedComment = snapshot.after.data()
        if (updatedComment.msg.length > MAX_LENGTH) {
            admin.firestore().collection('comments').doc(commentId).delete()
        }
    })
```

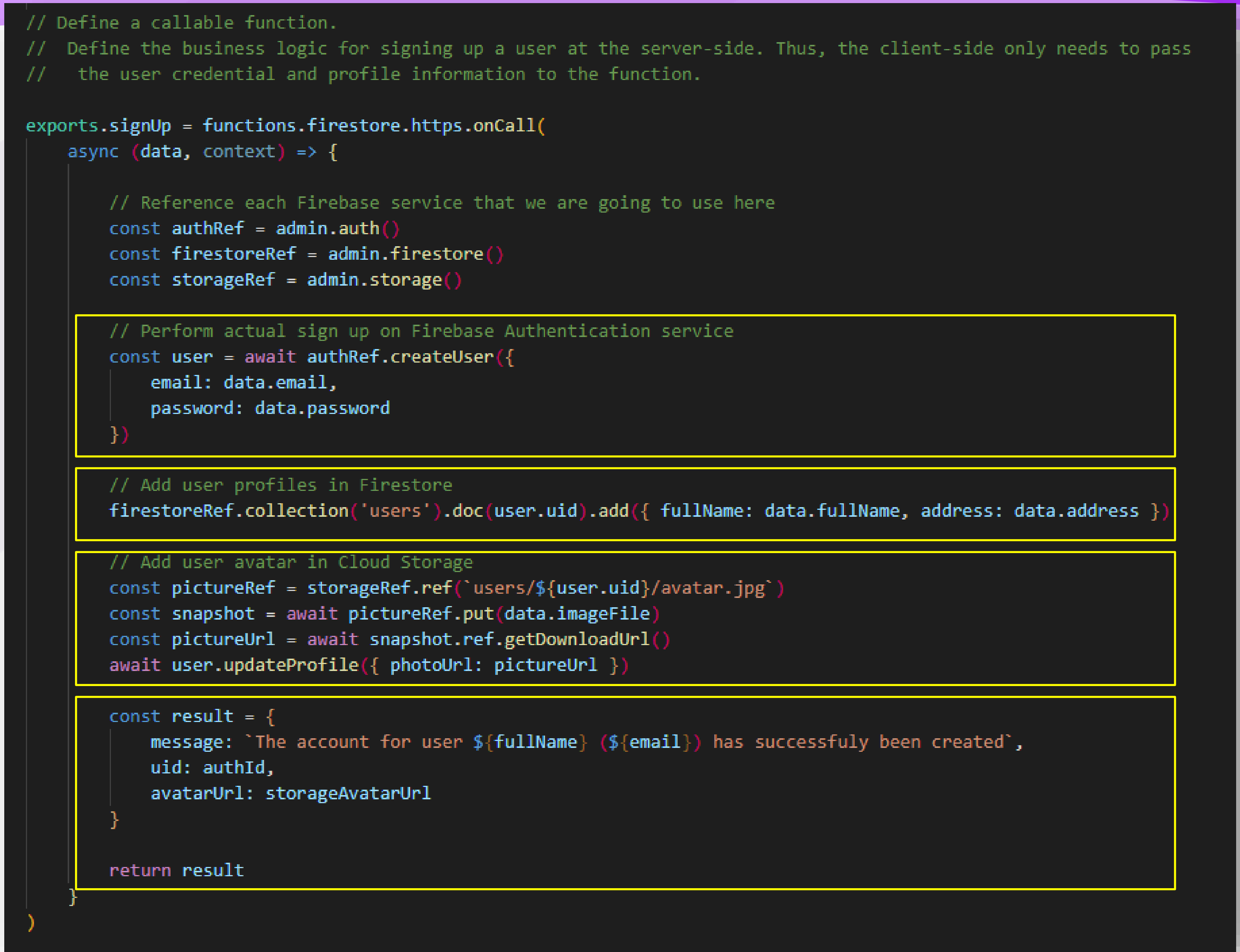
How to Write Cloud Functions? (4)

Example 2:

Callable Functions

A callable function can be called directly from the client app.

An example use case, to refactor some business logics code at client-side and move it to server-side

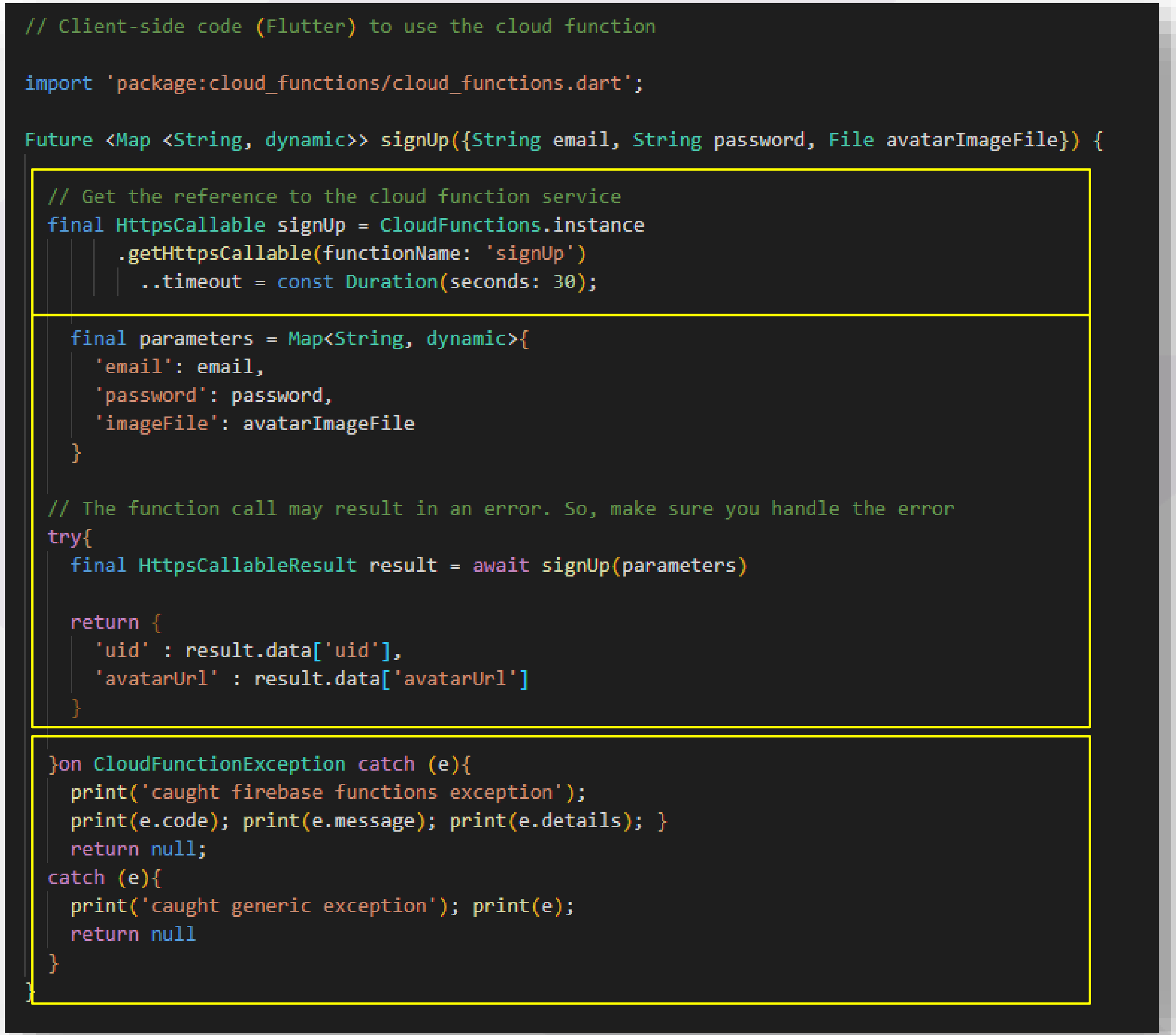


```
// Define a callable function.
//  Define the business logic for signing up a user at the server-side. Thus, the client-side only needs to pass
//   the user credential and profile information to the function.

exports.signUp = functions.firestore.https.onCall(
    async (data, context) => {

        // Reference each Firebase service that we are going to use here
        const authRef = admin.auth()
        const firestoreRef = admin.firestore()
        const storageRef = admin.storage()

        // Perform actual sign up on Firebase Authentication service
        const user = await authRef.createUser({
            email: data.email,
            password: data.password
        })

        // Add user profiles in Firestore
        firestoreRef.collection('users').doc(user.uid).add({ fullName: data.fullName, address: data.address })

        // Add user avatar in Cloud Storage
        const pictureRef = storageRef.ref(`users/${user.uid}/avatar.jpg`)
        const snapshot = await pictureRef.put(data.imageFile)
        const pictureUrl = await snapshot.ref.getDownloadUrl()
        await user.updateProfile({ photoUrl: pictureUrl })

        const result = {
            message: `The account for user ${fullName} (${email}) has successfuly been created`,
            uid: authId,
            avatarUrl: storageAvatarUrl
        }

        return result
    }
)
```

How to Write Cloud Functions? (5)

To call to cloud functions from Flutter, use the CloudFunction package

Each cloud function call is asynchronous



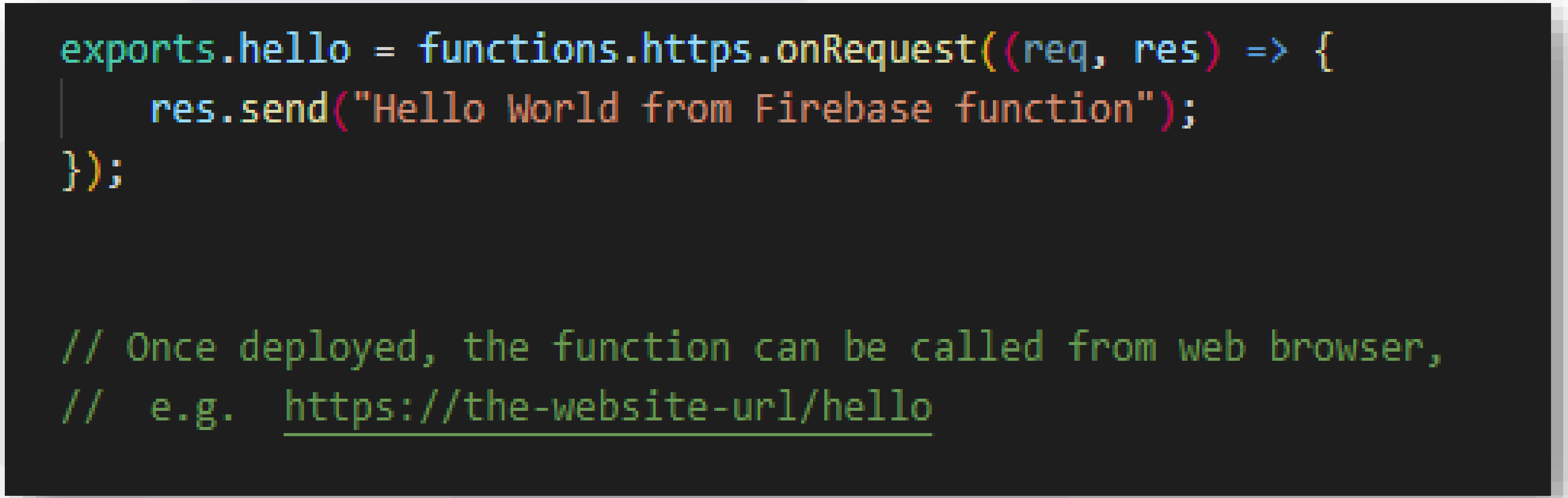
```
// Client-side code (Flutter) to use the cloud function

import 'package:cloud_functions/cloud_functions.dart';

Future <Map <String, dynamic>> signUp({String email, String password, File avatarImageFile}) {

  // Get the reference to the cloud function service
  final HttpsCallable signUp = CloudFunctions.instance
        .getHttpsCallable(functionName: 'signUp')
          ..timeout = const Duration(seconds: 30);

    final parameters = Map<String, dynamic>{
      'email': email,
      'password': password,
      'imageFile': avatarImageFile
    }

  // The function call may result in an error. So, make sure you handle the error
  try{
    final HttpsCallableResult result = await signUp(parameters)

    return {
      'uid' : result.data['uid'],
      'avatarUrl' : result.data['avatarUrl']
    }

  }on CloudFunctionException catch (e){
    print('caught firebase functions exception');
    print(e.code); print(e.message); print(e.details); }
    return null;
  catch (e){
    print('caught generic exception'); print(e);
    return null
  }
}
```

How to Write Cloud Functions? (6)

Example 3:

HTTP Requests

Functions are called from HTTP-based clients, e.g. web browser, REST client



```
exports.hello = functions.https.onRequest((req, res) => {
    res.send("Hello World from Firebase function");
});


// Once deployed, the function can be called from web browser,
//  e.g.  https://the-website-url/hello
```

Demo

Developing REST Service on Firebase

Demo - Project Source Code

backend

https://github.com/jumail-utm/backend_firebase_rest

Frontend (Flutter App – Todo List)

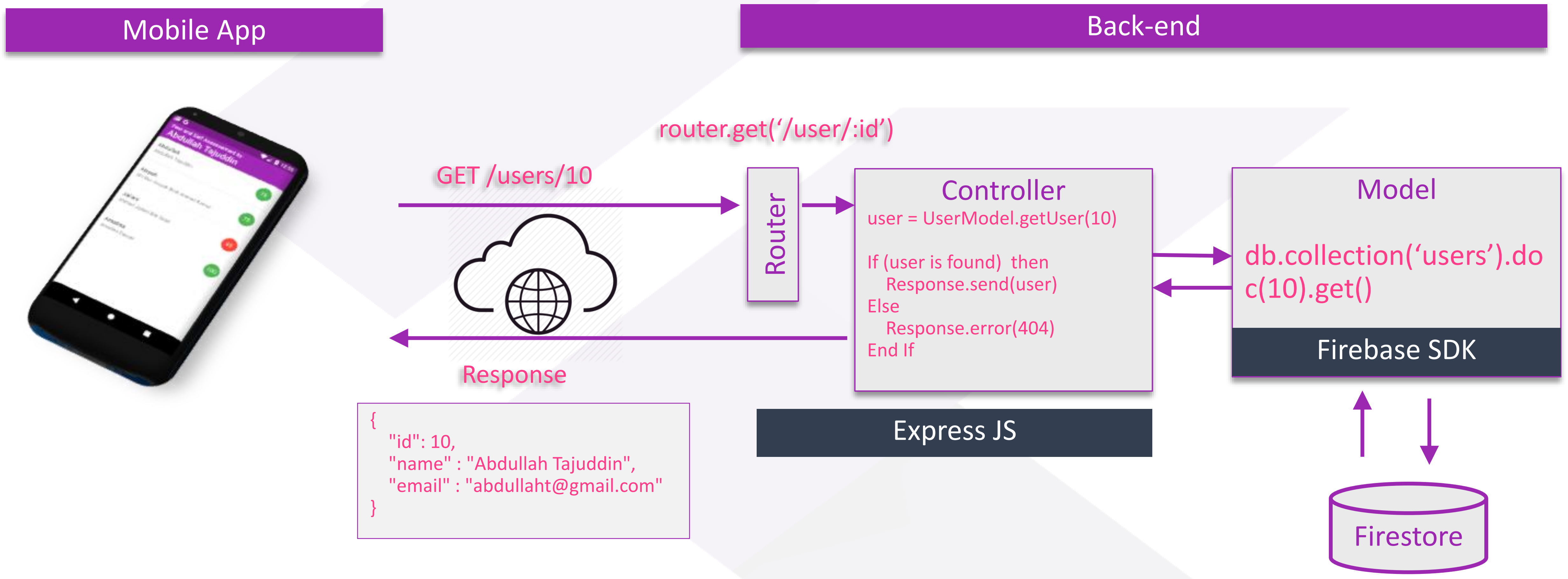
https://github.com/jumail-utm/flutter_todo_rest

Command line and code snippet to code along:

https://jumail-utm.github.io/backend_firebase_rest

Back-end Architecture

Adopt MVC architecture

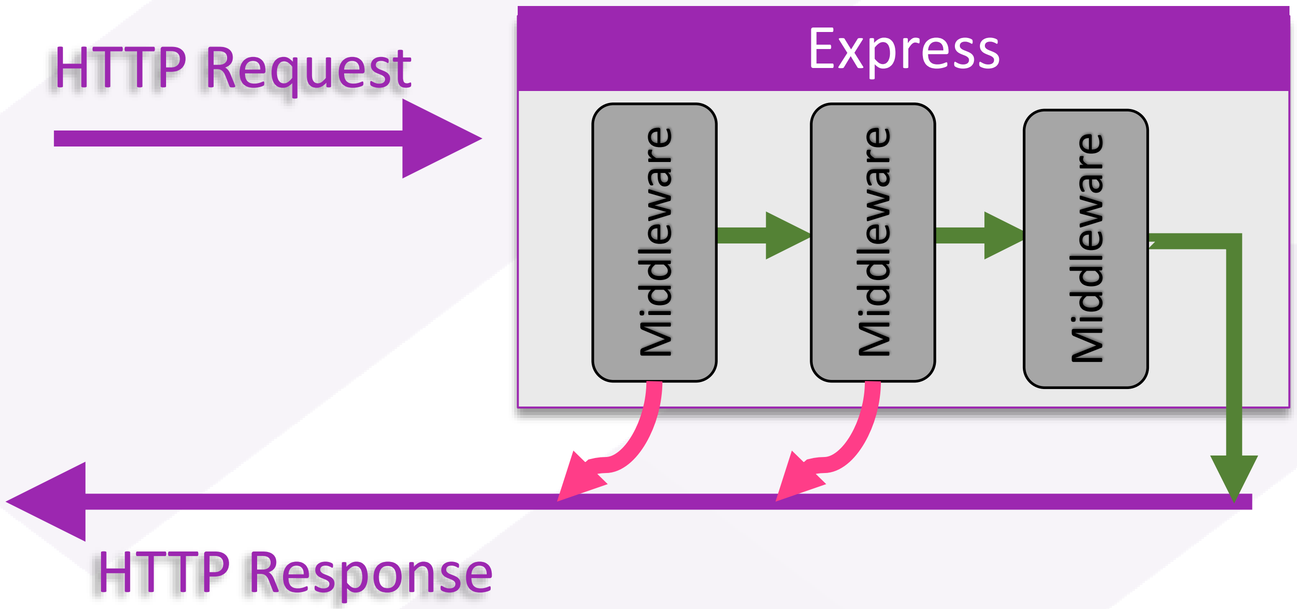


```
user = UserModel.getUser(10)

If (user is found)  then
    Response.send(user)
Else
    Response.error(404)
End If
```

```
db.collection('users').doc(10).get()
```

```
{
   "id": 10,
   "name" : "Abdullah Tajuddin",
   "email" : "abdullaht@gmail.com"
}
```

Express JS Middleware

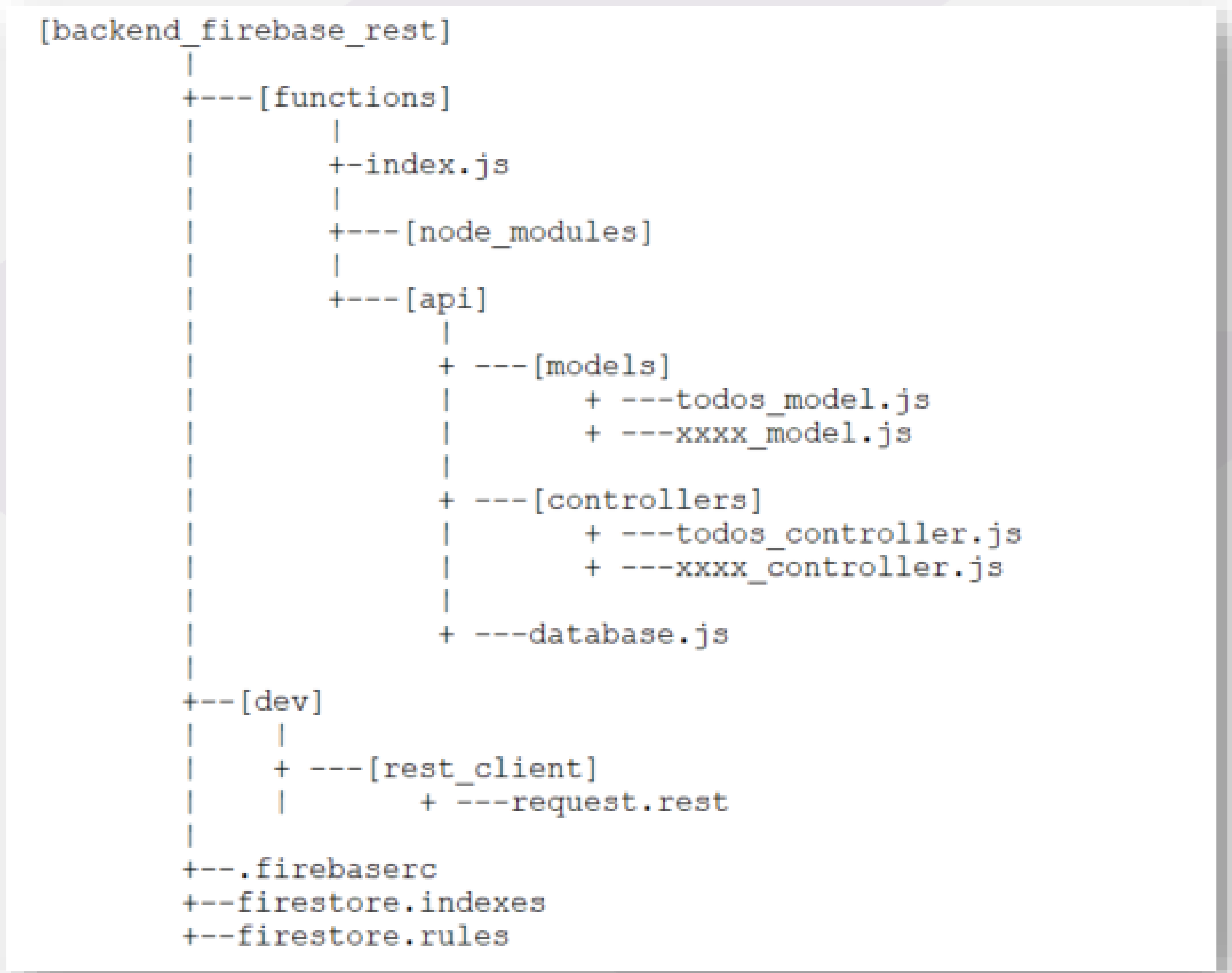


Watch the previous lecture video to learn more about Express JS and Router

Setting Up Local Firebase

Developing REST API

Project Structure



```
[backend_firebase_rest]
        |
        +---[functions]
        |       |
        |       +-index.js
        |       |
        |       +---[node_modules]
        |       |
        |       +---[api]
        |             |
        |             + ---[models]
        |             |       + ---todos_model.js
        |             |       + ---xxxx_model.js
        |             |
        |             + ---[controllers]
        |             |       + ---todos_controller.js
        |             |       + ---xxxx_controller.js
        |             |
        |             + ---database.js
        |
        +--[dev]
        |    |
        |    + ---[rest_client]
        |    |        + ---request.rest
        |
        +--.firebaserc
        +--firestore.indexes
        +--firestore.rules
```

Deploying the Project

Summary

- Introduction to Firebase
- Set up Local Firebase
- Develop REST API on Firebase
- Deploy REST API to Firebase