Integrating with Backend

HTTP and JSON

Part 1 - Lecture

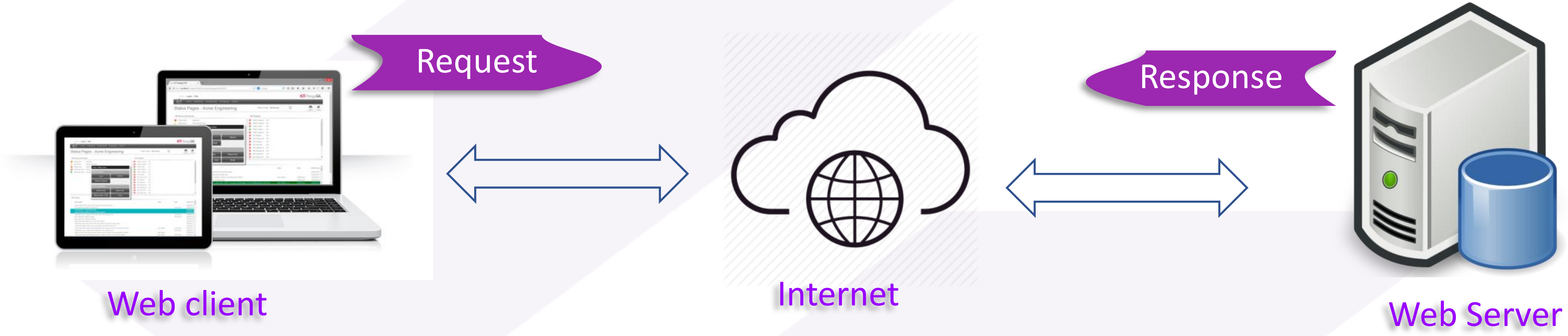
Jumail Bin Taliba
School of Computing, UTM

May 2020

Agenda

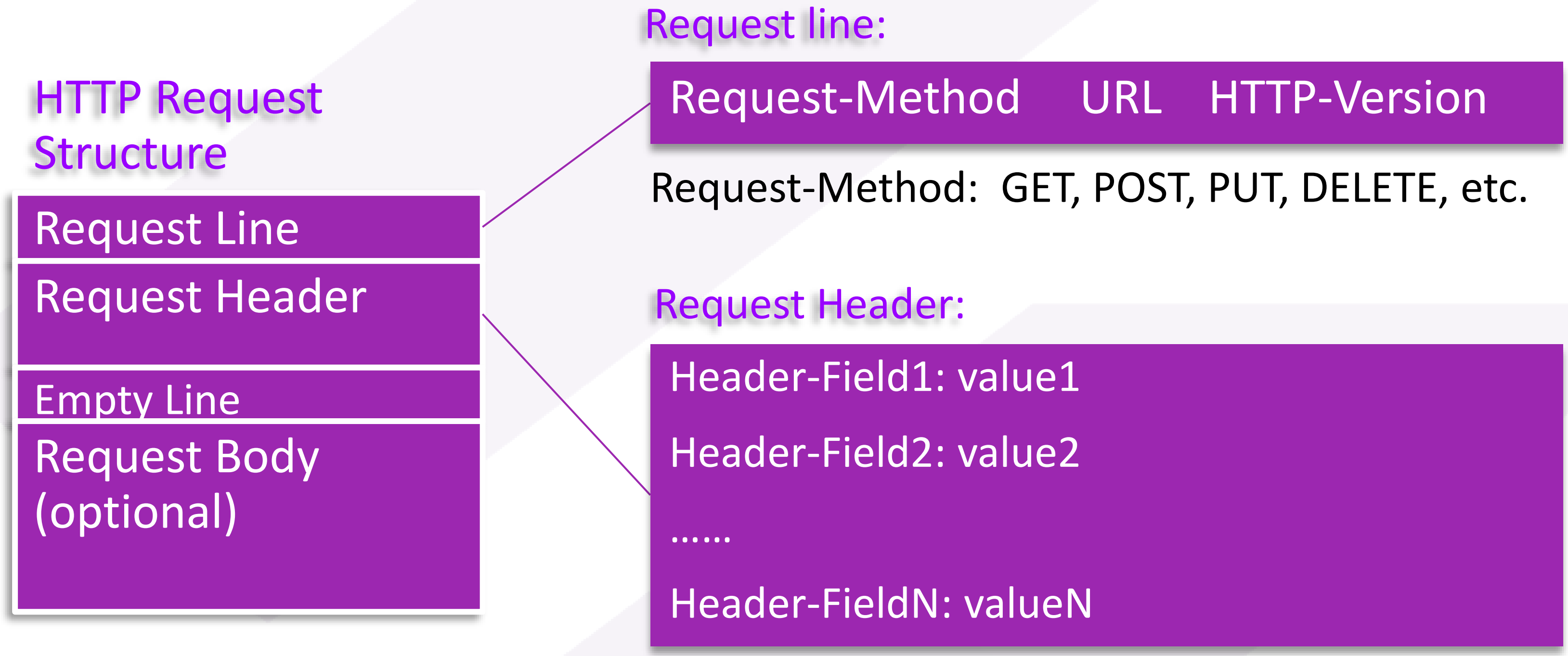
- Introduction to HTTP
- HTTP Request and Response
- Dart’s http package
- Introduction to JSON
- Data Conversion
- JSON Decoding and Encoding
- Conversion Examples
- Demo

Introduction to HTTP



- HTTP - HyperText Transfer Protocol
- Defines how data are transmitted between clients and servers over the world wide web.
- Client send HTTP Request to the server
- Server reply with HTTP Response to the client

HTTP Request



HTTP Request Structure

Request Line

Request Header

Empty Line

Request Body (optional)

Request line:

Request-Method URL HTTP-Version

Request-Method: GET, POST, PUT, DELETE, etc.

Request Header:

Header-Field1: value1

Header-Field2: value2

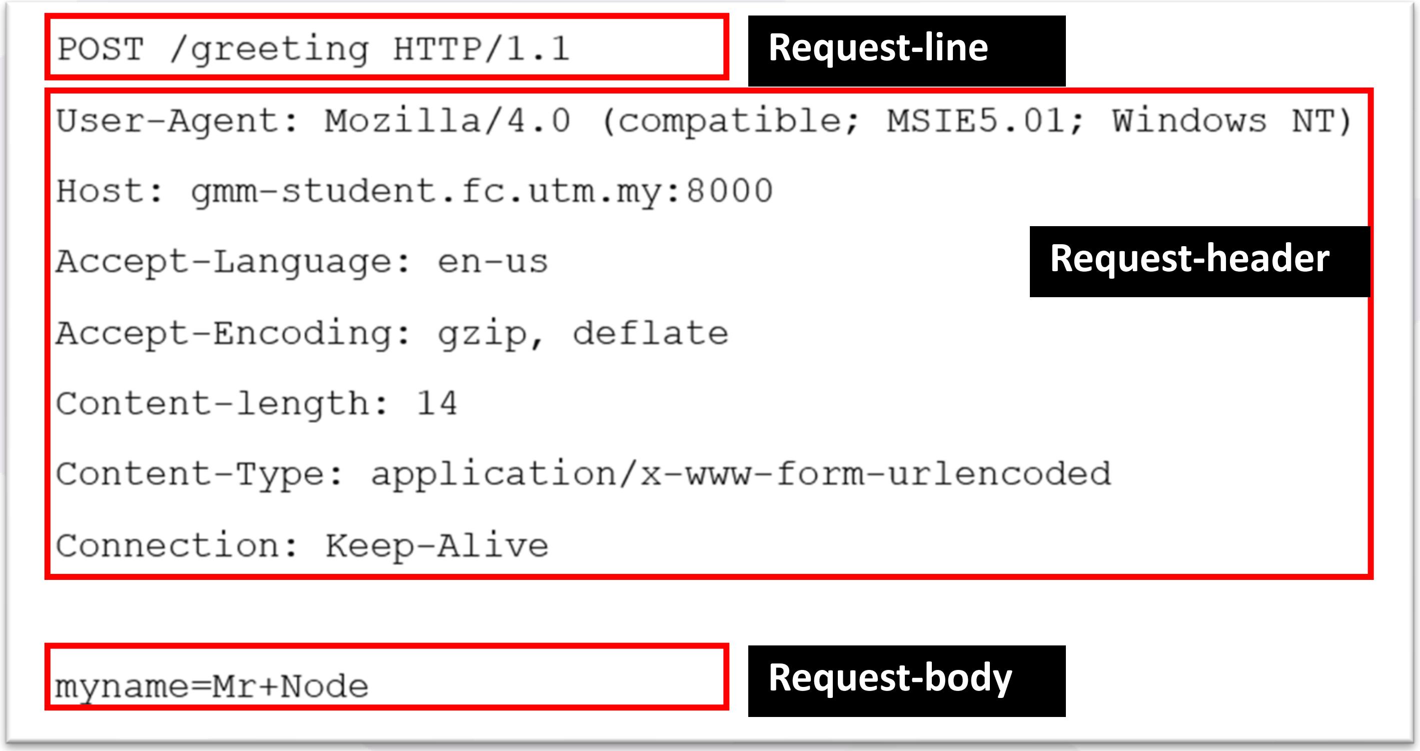
……

Header-FieldN: valueN

Request header fields allow the client to send additional information about the request and about the client itself.

HTTP Request (2)

HTTP Request Example



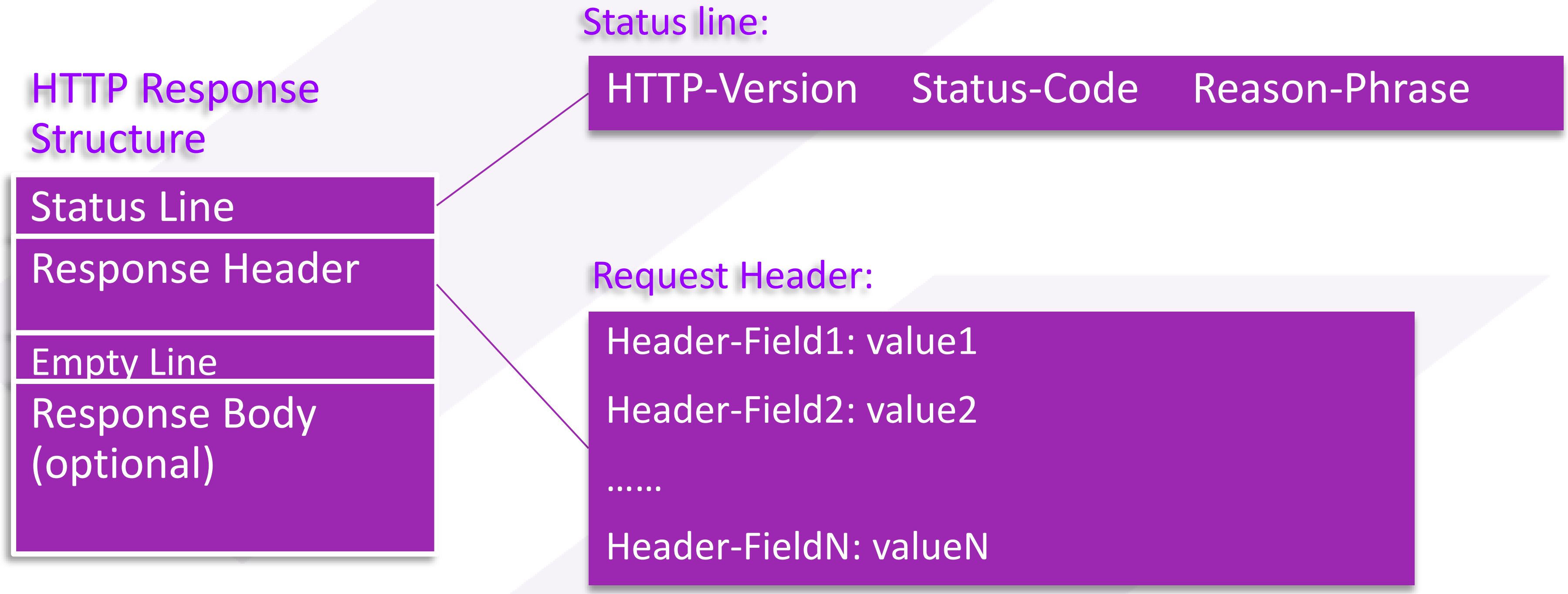
```
POST /greeting HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: gmm-student.fc.utm.my:8000
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Content-length: 14
Content-Type: application/x-www-form-urlencoded
Connection: Keep-Alive

myname=Mr+Node
```

HTTP Request (3)

HTTP Request Methods

- GET is used to request data from a specified resource.
- POST is used to request for the server to create a new resource.
- PUT is used to request for the server to replace / update a resource.
- HEAD is similar to GET, but returns only the response header.
- DELETE is used to request for the server to delete a resource.
- Other methods: PATCH, TRACE, OPTIONS, CONNECT

HTTP Response



HTTP Response Structure

Status Line

Response Header

Empty Line

Response Body (optional)

Status line:

HTTP-Version Status-Code Reason-Phrase

Request Header:

Header-Field1: value1

Header-Field2: value2

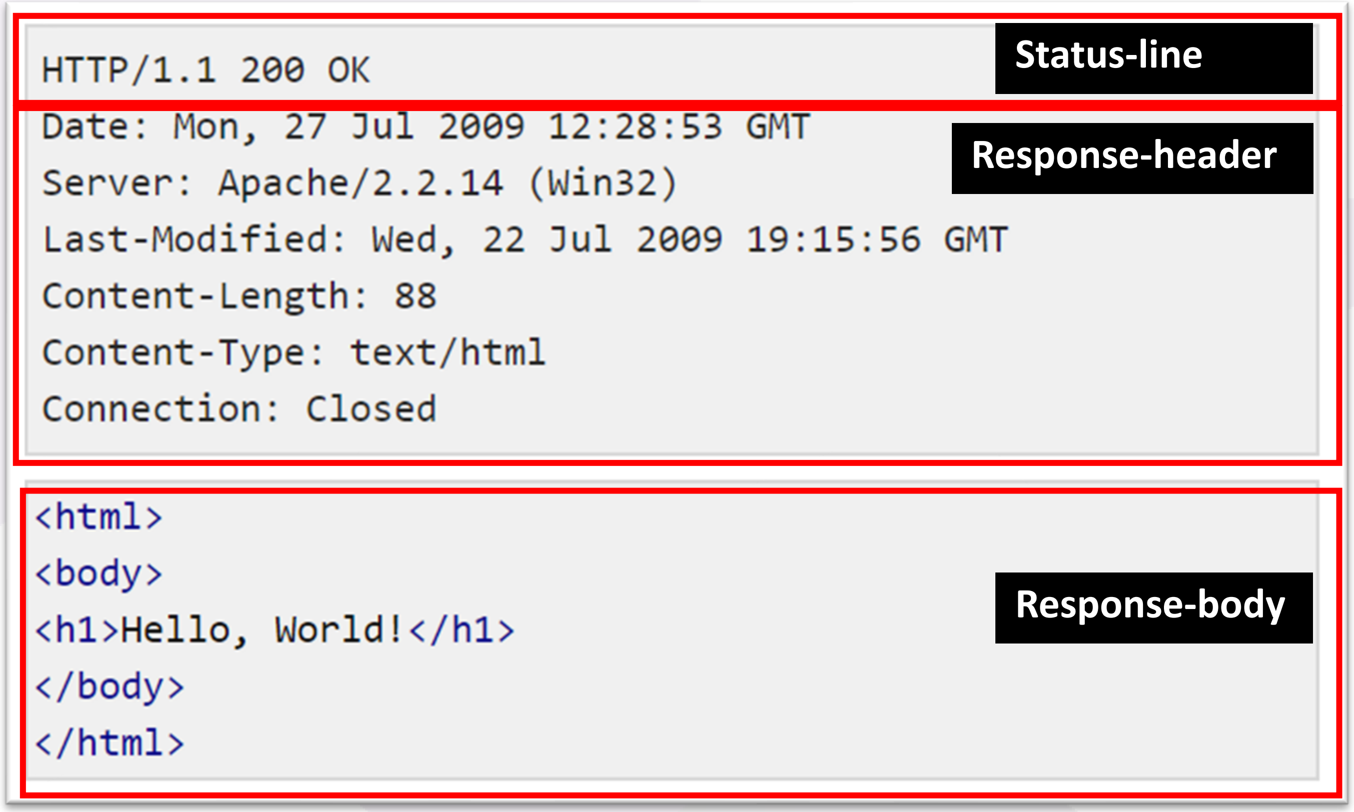
……

Header-FieldN: valueN

The response-header fields allow the server to send additional information about the response, such as information about the server itself

HTTP Response (2)

HTTP Response Example



```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

HTTP Response (3)

Response Status Code

- 1XX: Information
  - e.g. 100 Continue. The server has received the request headers, and the client should proceed to send the request body
- 2XX: Successful
  - 200 OK. The request is OK
  - 201 OK. The request has been fulfilled, and a new resource is created

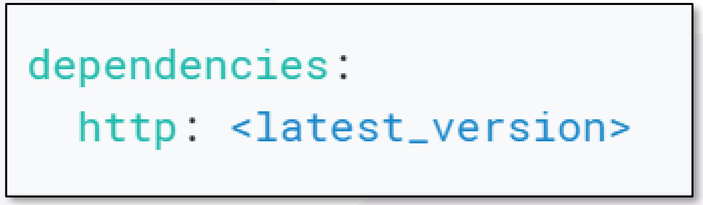
HTTP Response (4)

Response Status Code

- 3XX: Redirection
  - e.g. 301 Moved Permanently. The requested page has moved to a new URL
- 4XX: Client Error
  - e.g. 404 Not Found. The request was a legal request, but the server is refusing to respond to it
- 5XX: Server Error
  - e.g. 500 Internal Server Error. A generic error message, given when no more specific message is suitable

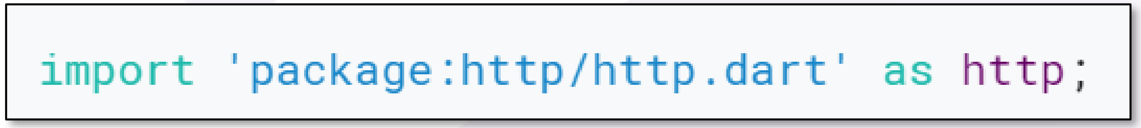
Dart’s http Package

- The http package provides services that allow a flutter app (as a client) to communicate to a web server.
- To add this package to a flutter project:
  - In pubspec.yaml file



```
dependencies:
  http: <latest_version>
```

  - Import it in dart file:



```
import 'package:http/http.dart' as http;
```

https://pub.dev/documentation/http/latest/http/http-library.html

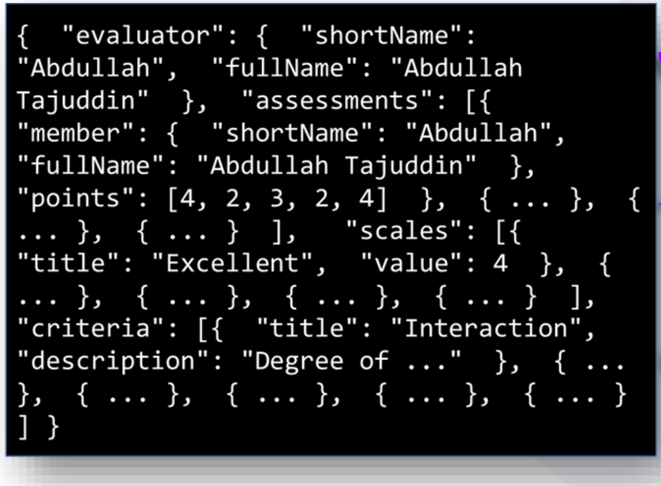
Introduction to JSON

- JSON stands for JavaScript Object Notation
- It is meant for exchanging data between systems, e.g. client and server
- It is text, written with JavaScript object syntax.
- It is language independent (although it is based on JavaScript)

Data Conversion

Client Application

String JSON

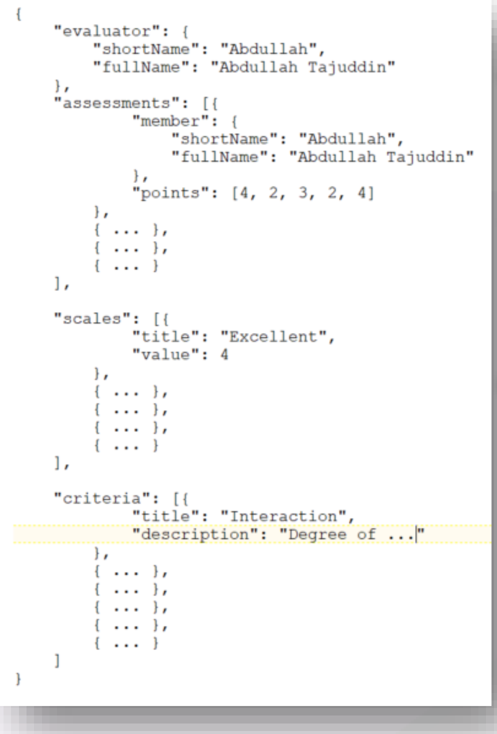


```
{  "evaluator": {  "shortName":
"Abdullah",  "fullName": "Abdullah
Tajuddin"  },  "assessments": [{
"member": {  "shortName": "Abdullah",
"fullName": "Abdullah Tajuddin"  },
"points": [4, 2, 3, 2, 4]  },  { ... },  {
... },  { ... }  ],   "scales": [{
"title": "Excellent",  "value": 4  },  {
... },  { ... },  { ... },  { ... }  ],
"criteria": [{  "title": "Interaction",
"description": "Degree of ..."  },  { ...
},  { ... },  { ... },  { ... },  { ... }
] }
```

deserialize / parse / decode

Serialize / encode

Parsed JSON (Map)

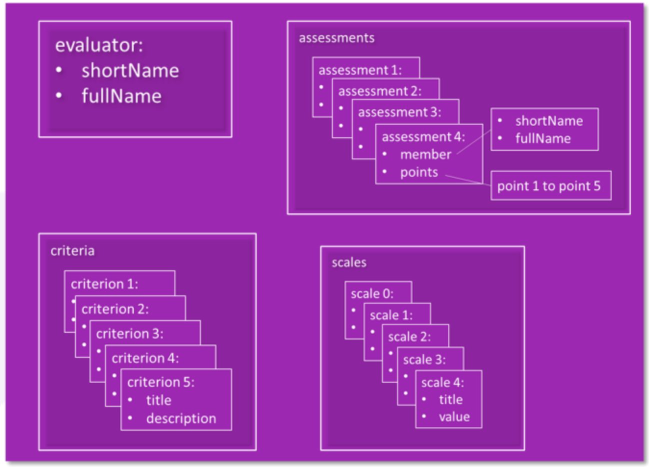


```
{
    "evaluator": {
        "shortName": "Abdullah",
        "fullName": "Abdullah Tajuddin"
    },
    "assessments": [{
            "member": {
                "shortName": "Abdullah",
                "fullName": "Abdullah Tajuddin"
            },
            "points": [4, 2, 3, 2, 4]
        },
        { ... },
        { ... },
        { ... }
    ],

    "scales": [{
            "title": "Excellent",
            "value": 4
        },
        { ... },
        { ... },
        { ... },
        { ... }
    ],

    "criteria": [{
            "title": "Interaction",
            "description": "Degree of ..."
        },
        { ... },
        { ... },
        { ... },
        { ... },
        { ... }
    ]
}
```

Further conversion to objects

Object to Map

Objects, Lists, Nested Objects



Where does it come from?

- Backend response
- App assets
- Device Local file
- Client request to backend

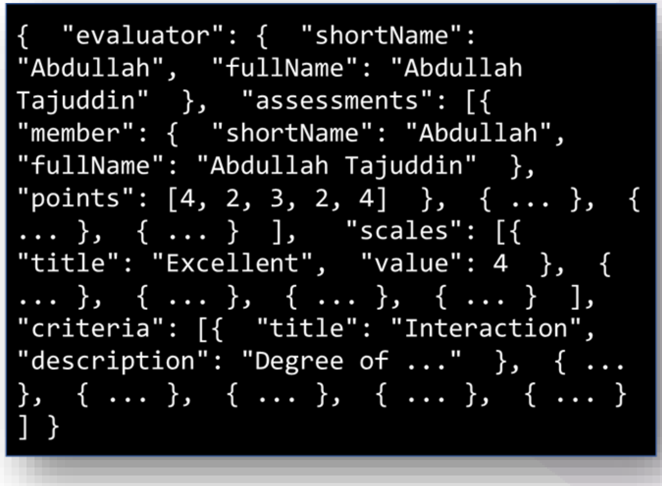
Data Conversion (2)

- Serialization – convert structured data to string (String is a series of characters).
- Parsing – split a string into its components.
- JSON data conversion in Dart can be done with the dart:convert package
- Related methods from the package:
  - jsonDecode() - for deserialization / decoding / parsing string json to map
  - jsonEncode() – for serialization / encoding object to string json

JSON Decoding

Client Application

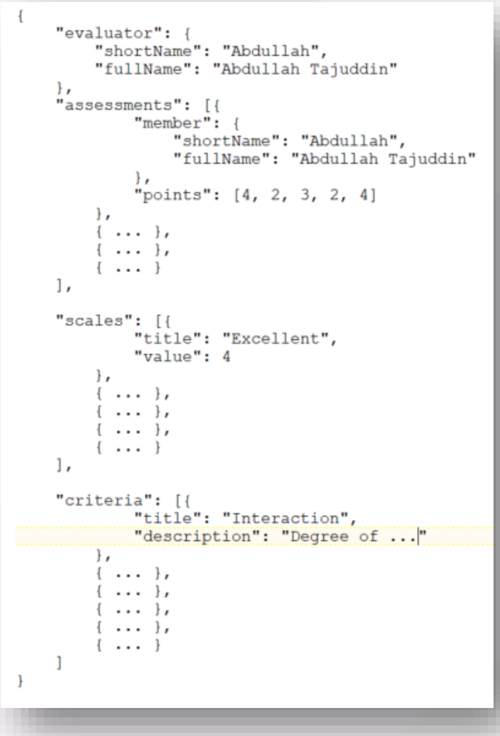
String JSON



```
{  "evaluator": {  "shortName":
"Abdullah",  "fullName": "Abdullah
Tajuddin"  },  "assessments": [{
"member": {  "shortName": "Abdullah",
"fullName": "Abdullah Tajuddin"  },
"points": [4, 2, 3, 2, 4]  },  { ... },  {
... },  { ... }  ],   "scales": [{
"title": "Excellent",  "value": 4  },  {
... },  { ... },  { ... },  { ... }  ],
"criteria": [{  "title": "Interaction",
"description": "Degree of ..."  },  { ...
},  { ... },  { ... },  { ... },  { ... }
] }
```

(1)
dart::convert
jsonDecode()

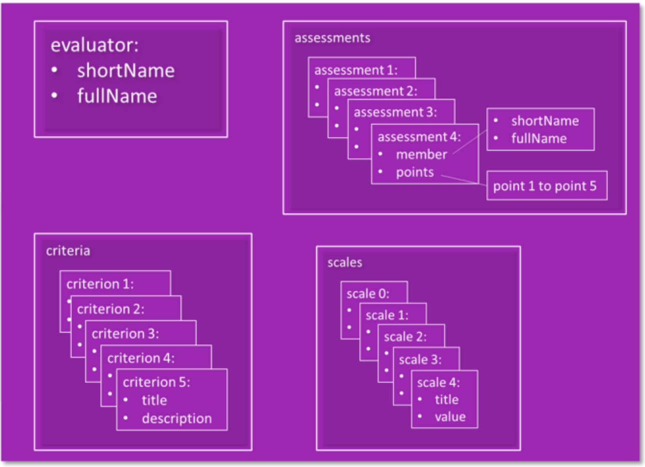
Parsed JSON (Map)



```
{
    "evaluator": {
        "shortName": "Abdullah",
        "fullName": "Abdullah Tajuddin"
    },
    "assessments": [{
            "member": {
                "shortName": "Abdullah",
                "fullName": "Abdullah Tajuddin"
            },
            "points": [4, 2, 3, 2, 4]
        },
        { ... },
        { ... },
        { ... }
    ],

    "scales": [{
            "title": "Excellent",
            "value": 4
        },
        { ... },
        { ... },
        { ... },
        { ... }
    ],

    "criteria": [{
            "title": "Interaction",
            "description": "Degree of ..."
        },
        { ... },
        { ... },
        { ... },
        { ... },
        { ... }
    ]
}
```

(2)
Model class's
fromJson()

Objects, Lists, Nested Objects



JSON Decoding (2)

First conversion:

- Is done with jsonDecode() .
- Convert string JSON to map data structure.

- So that we can interpret the content of the JSON data.
  - JSON string is just a series of characters. It has no meaning.
  - Thus, we need to parse or split to its component. This is done by jsonDecode() method.

JSON Decoding (3)

Second conversion:

- Is done with the fromJson() constructor from each model class.
- Convert the parsed JSON (i.e, a map) to strongly-typed data structure such as objects.
- So that we can still use statically typed language features, such as type safety and autocompletion.
  - For example, with the parsed JSON (i.e. a map), the code below has an error (i.e., there is no data of ‘longName’), however the error is only detected at runtime

```
print(parsedJson['longName']);
```

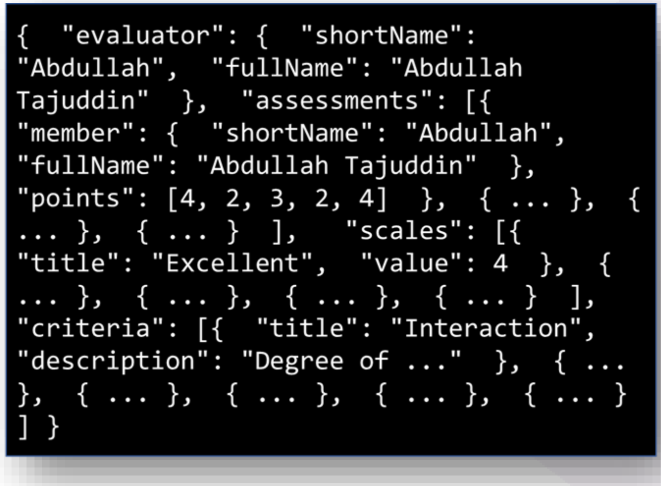
  - However, if using object the error can be detected at compile-time.

```
print(object.longName);
```

JSON Encoding

Client Application

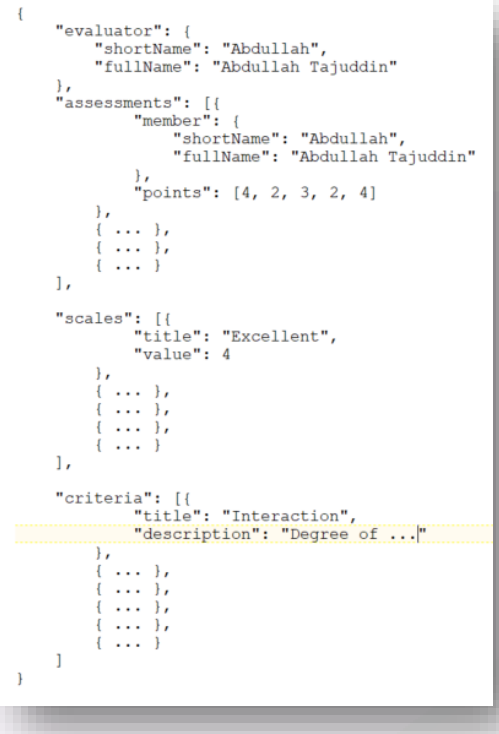
String JSON



```
{  "evaluator": {  "shortName":
"Abdullah",  "fullName": "Abdullah
Tajuddin"  },  "assessments": [{
"member": {  "shortName": "Abdullah",
"fullName": "Abdullah Tajuddin"  },
"points": [4, 2, 3, 2, 4]  },  { ... },  {
... },  { ... }  ],   "scales": [{
"title": "Excellent",  "value": 4  },  {
... },  { ... },  { ... },  { ... }  ],
"criteria": [{  "title": "Interaction",
"description": "Degree of ..."  },  { ...
},  { ... },  { ... },  { ... },  { ... }
] }
```

(2)
dart:convert
jsonEncode()

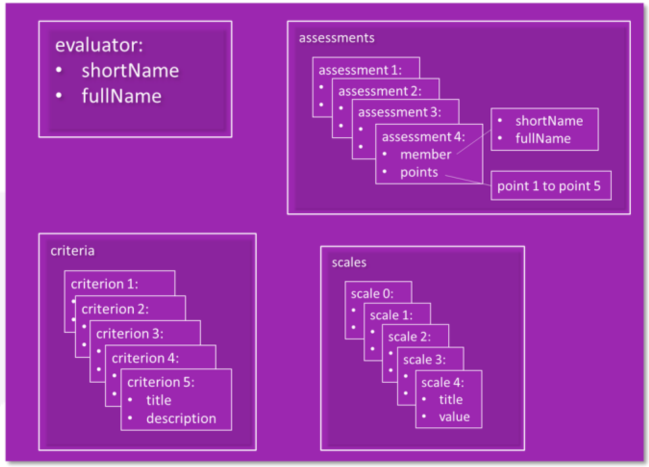
JSON Map



```
{
    "evaluator": {
        "shortName": "Abdullah",
        "fullName": "Abdullah Tajuddin"
    },
    "assessments": [{
            "member": {
                "shortName": "Abdullah",
                "fullName": "Abdullah Tajuddin"
            },
            "points": [4, 2, 3, 2, 4]
        },
        { ... },
        { ... },
        { ... }
    ],

    "scales": [{
            "title": "Excellent",
            "value": 4
        },
        { ... },
        { ... },
        { ... },
        { ... }
    ],

    "criteria": [{
            "title": "Interaction",
            "description": "Degree of ..."
        },
        { ... },
        { ... },
        { ... },
        { ... },
        { ... }
    ]
}
```

(1)
Model class's
toJson()

Objects, Lists, Nested Objects



evaluator:
- shortName
- fullName

assessments

assessment 1:

assessment 2:

assessment 3:

assessment 4:
- member
- points

- shortName
- fullName

point 1 to point 5

criteria

criterion 1:

criterion 2:

criterion 3:

criterion 4:

criterion 5:
- title
- description

scales

scale 0:

scale 1:

scale 2:

scale 3:

scale 4:
- title
- value

JSON Encoding (2)

First conversion:

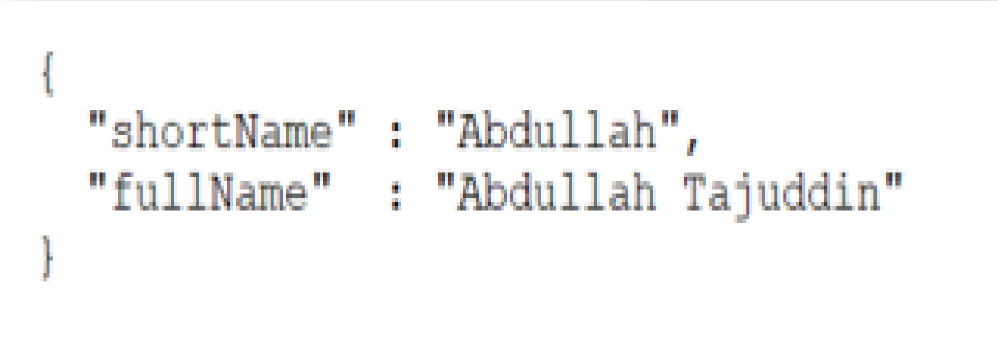
- Is done with the toJson() method from each model class.
- Convert structured data (such objects, list) to map.

Second conversion:

- Is done with jsonEncode() .
- Convert map to string JSON.

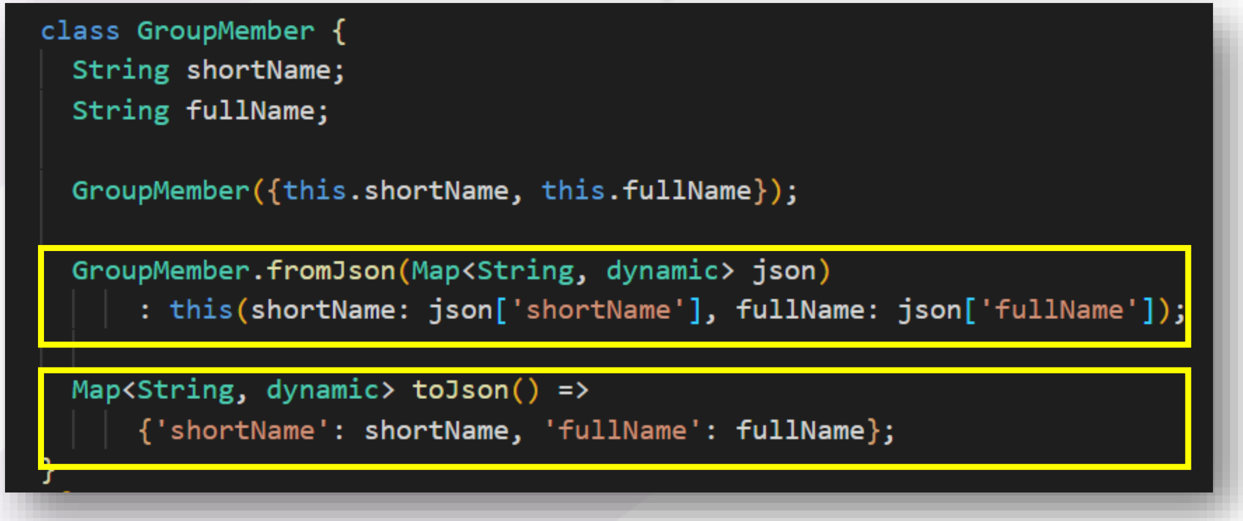
Conversion Example: Object

String JSON

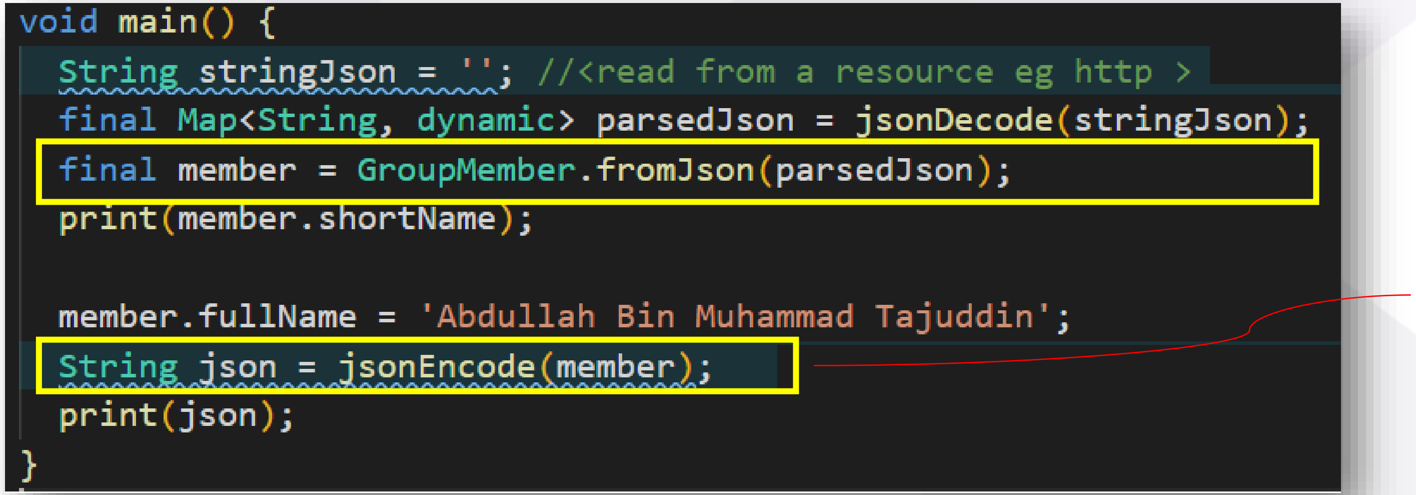


```
{
  "shortName" : "Abdullah",
  "fullName"  : "Abdullah Tajuddin"
}
```

Model class: GroupMember



```
class GroupMember {
  String shortName;
  String fullName;

  GroupMember({this.shortName, this.fullName});

  GroupMember.fromJson(Map<String, dynamic> json)
      : this(shortName: json['shortName'], fullName: json['fullName']);

  Map<String, dynamic> toJson() =>
      {'shortName': shortName, 'fullName': fullName};
}
```

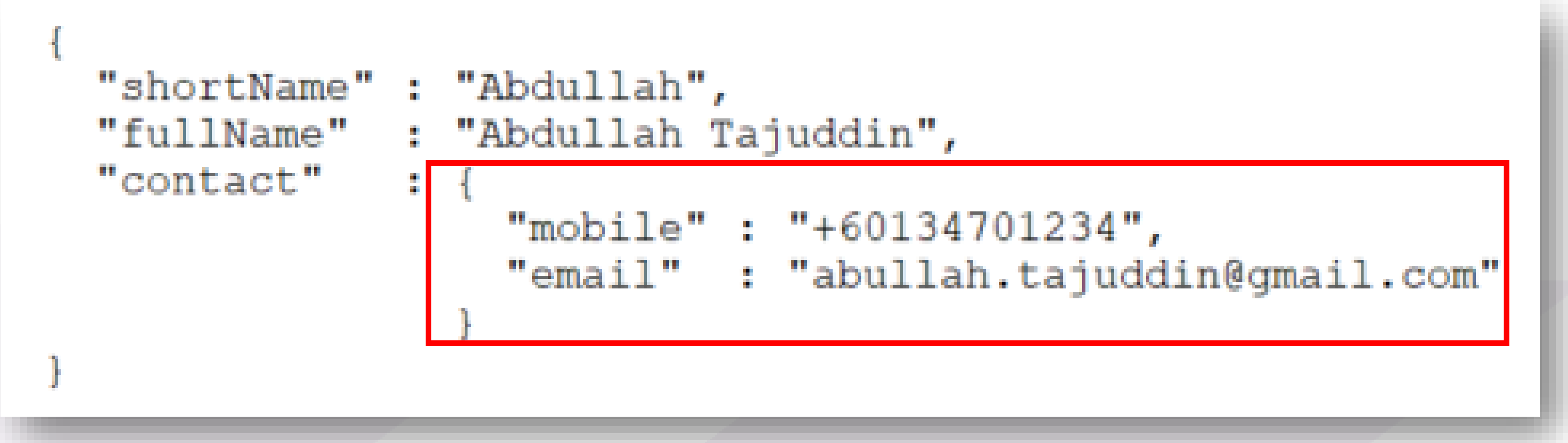


```
void main() {
  String stringJson = ''; //<read from a resource eg http >
  final Map<String, dynamic> parsedJson = jsonDecode(stringJson);
  final member = GroupMember.fromJson(parsedJson);
  print(member.shortName);

  member.fullName = 'Abdullah Bin Muhammad Tajuddin';
  String json = jsonEncode(member);
  print(json);
}
```

jsonEncode() automatically call to the toJson() method of the member object.

Conversion Example: Nested Object

String JSON

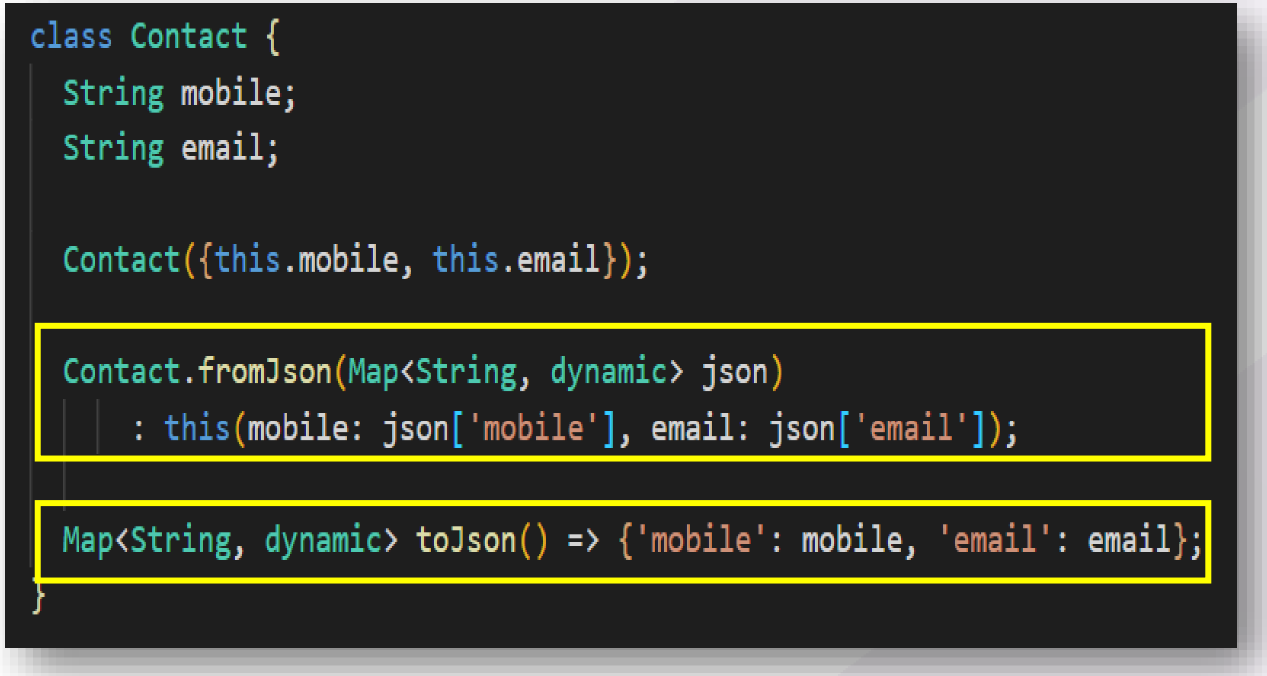


```
{
  "shortName" : "Abdullah",
  "fullName"  : "Abdullah Tajuddin",
  "contact"   : {
                  "mobile" : "+60134701234",
                  "email"  : "abullah.tajuddin@gmail.com"
                }
}
```

Define a dedicated class for the nested object, e.g. class Contact

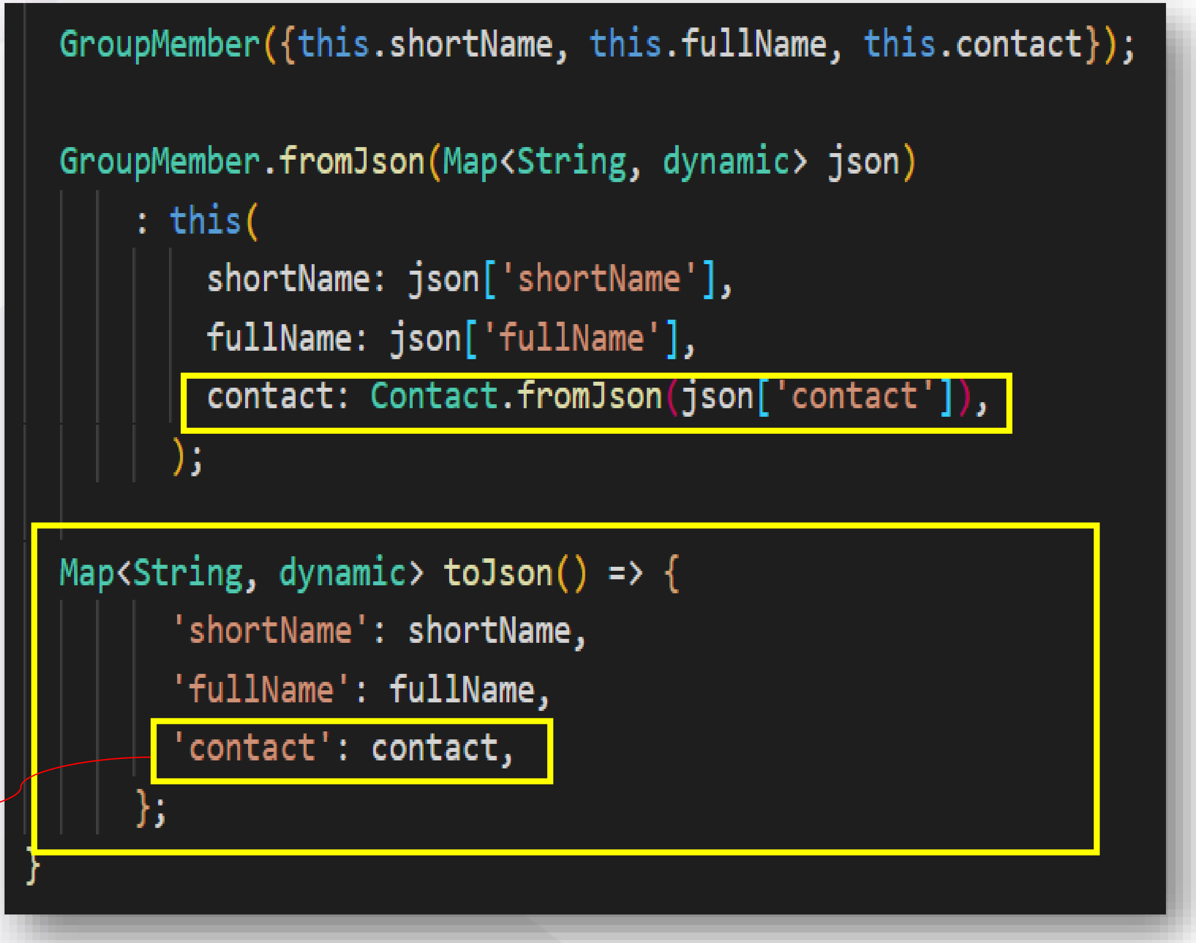
Conversion Example: Nested Object (2)

Model class: Contact



```
class Contact {
  String mobile;
  String email;

  Contact({this.mobile, this.email});

  Contact.fromJson(Map<String, dynamic> json)
      : this(mobile: json['mobile'], email: json['email']);

  Map<String, dynamic> toJson() => {'mobile': mobile, 'email': email};
}
```

Model class: GroupMember

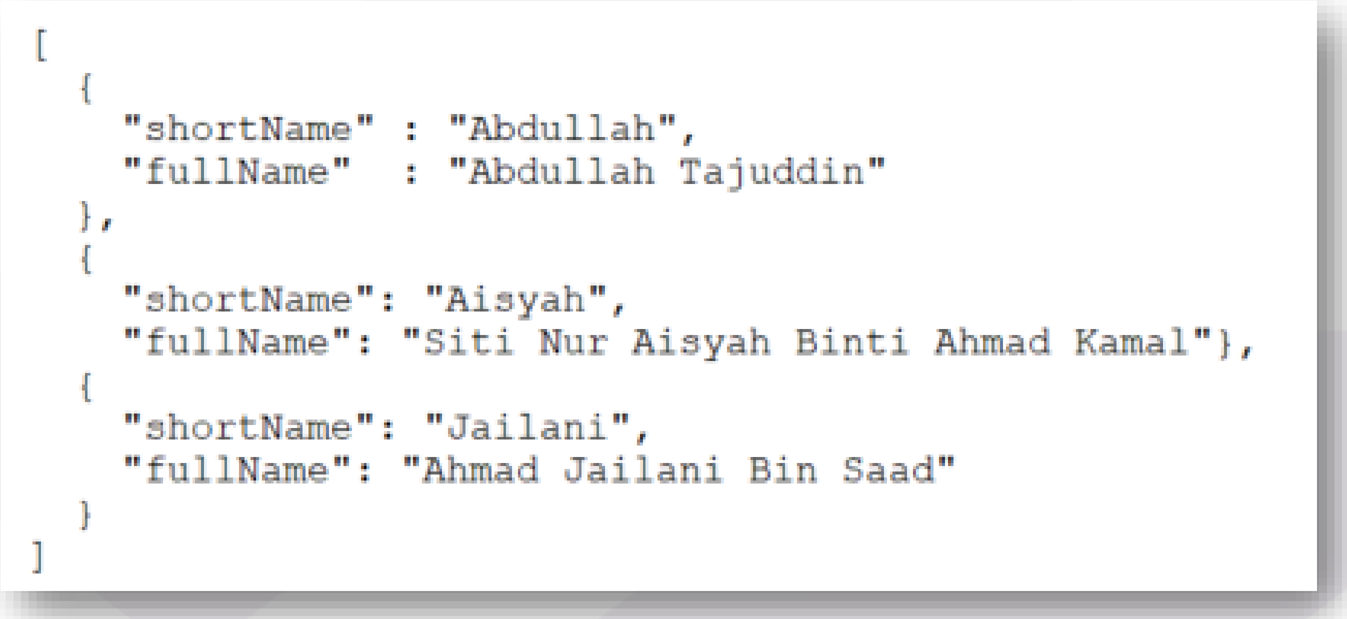


```
  GroupMember({this.shortName, this.fullName, this.contact});

  GroupMember.fromJson(Map<String, dynamic> json)
      : this(
          shortName: json['shortName'],
          fullName: json['fullName'],
          contact: Contact.fromJson(json['contact']),
        );

  Map<String, dynamic> toJson() => {
        'shortName': shortName,
        'fullName': fullName,
        'contact': contact,
      };
}
```

jsonEncode() will automatically call to the toJson() method of the contact object.

Conversion Example: List of Objects

String JSON

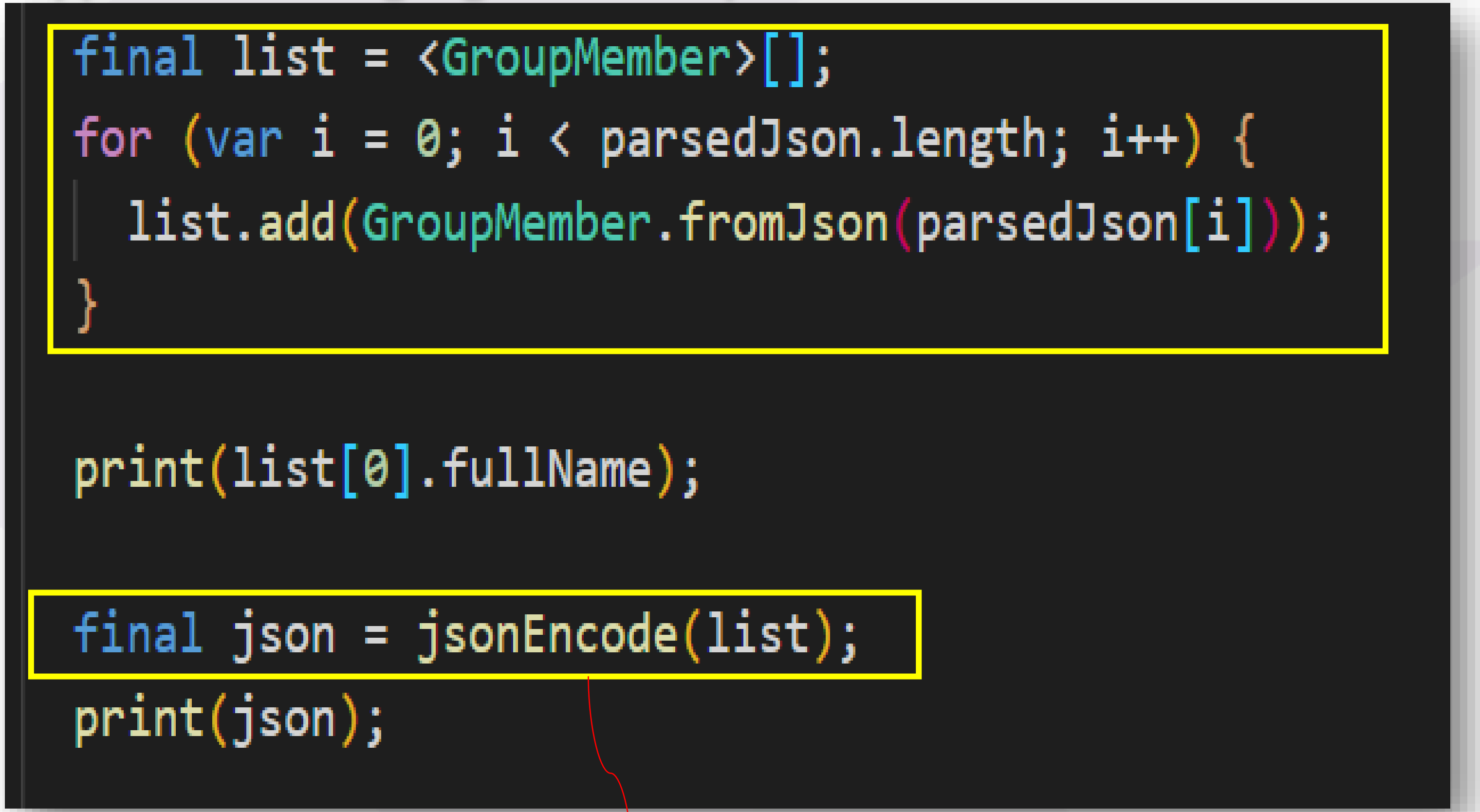


```
[
  {
    "shortName" : "Abdullah",
    "fullName"  : "Abdullah Tajuddin"
  },
  {
    "shortName": "Aisyah",
    "fullName": "Siti Nur Aisyah Binti Ahmad Kamal"},
  {
    "shortName": "Jailani",
    "fullName": "Ahmad Jailani Bin Saad"
  }
]
```

- Simply iterate each JSON data and put them in a list, `List<GroupMember>`
- Create an instance of `GroupMember` from the JSON data for each iteration

Conversion Example: List of Objects (2)

Approach 1: Using regular for-loop

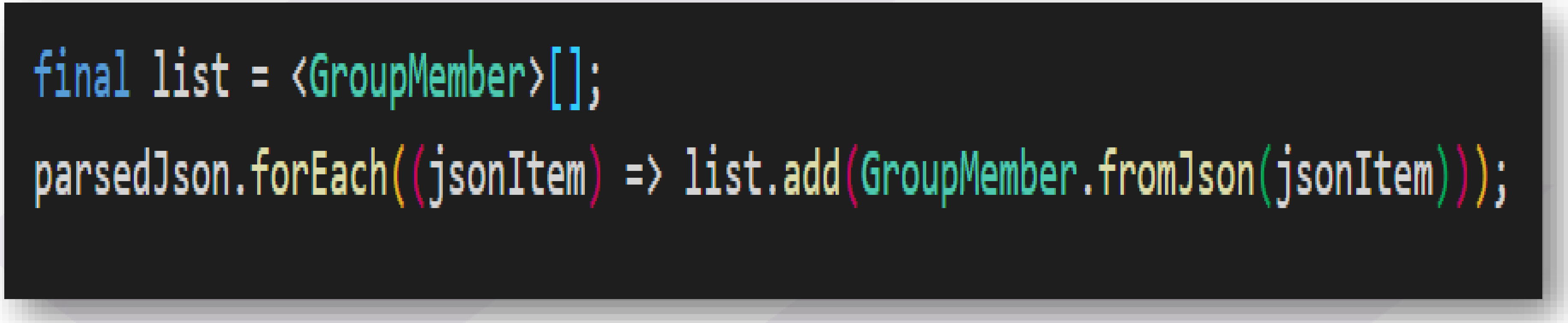


```
final list = <GroupMember>[];
for (var i = 0; i < parsedJson.length; i++) {
  list.add(GroupMember.fromJson(parsedJson[i]));
}

print(list[0].fullName);

final json = jsonEncode(list);
print(json);
```

jsonEncode() will automatically call to the toJson() method for each object.

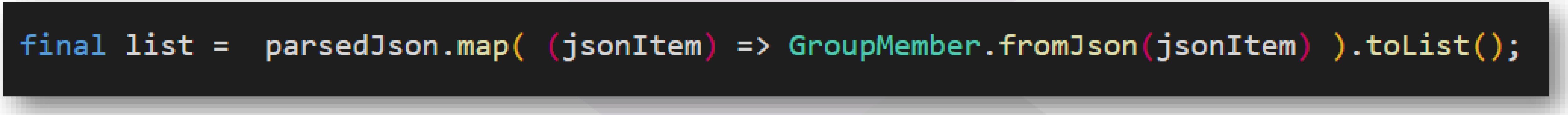
Conversion Example: List of Objects (3)

Approach 2: Using high-order method forEach()



```
final list = <GroupMember>[];
parsedJson.forEach((jsonItem) => list.add(GroupMember.fromJson(jsonItem)));
```

Approach 3: Using high-order method map() - Recommended



```
final list =  parsedJson.map( (jsonItem) => GroupMember.fromJson(jsonItem) ).toList();
```

Summary

- HTTP
- HTTP Request and Response
- JSON
- Decoding and Encoding
- Conversion – How to

Integrating with Backend

HTTP and JSON

Part 2 - Demo

Jumail Bin Taliba
School of Computing, UTM

May 2020

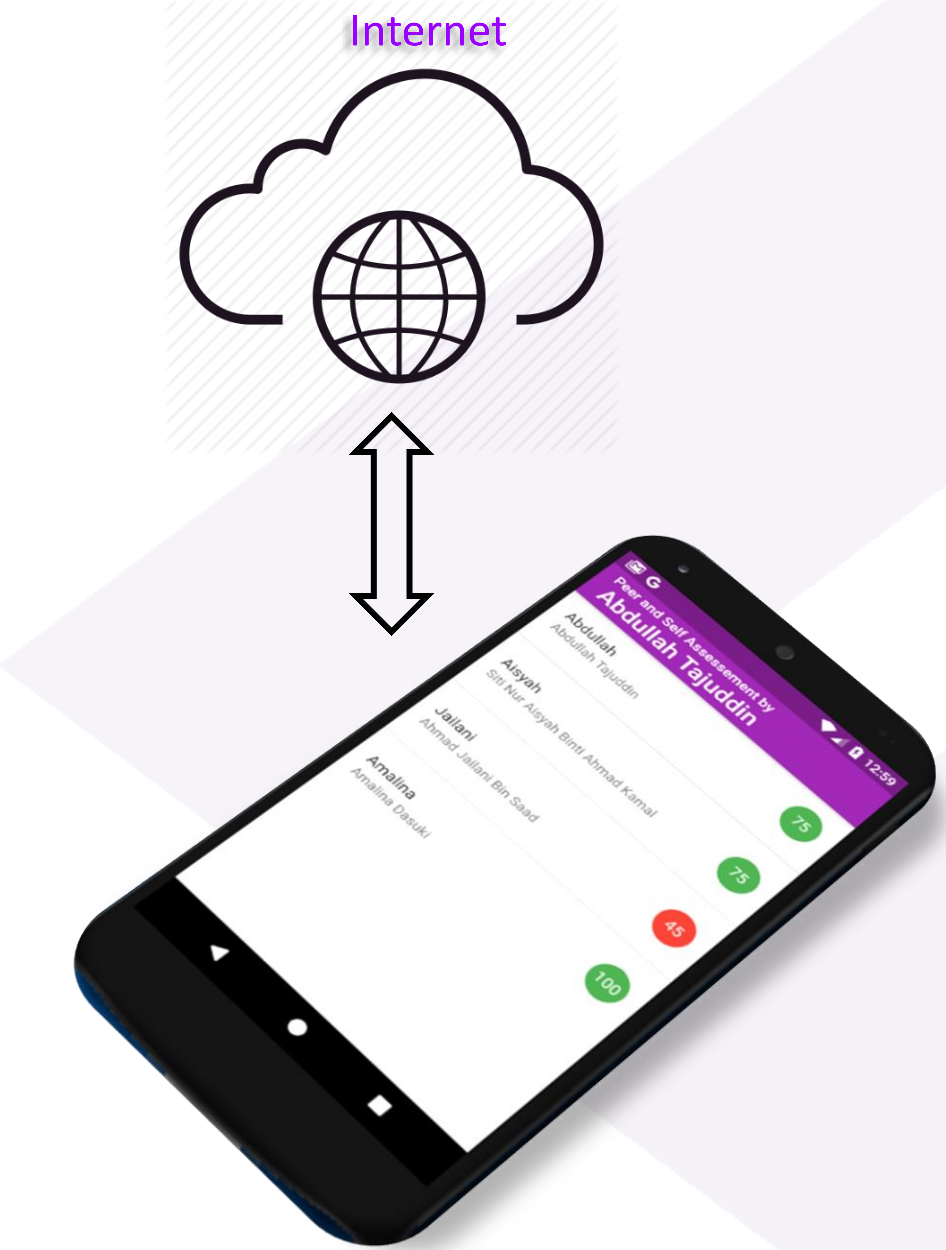
Watch on YouTube

Set the playback speed 1.5X

Use the timestamp in the description

Outline

- Adding Conversion Methods to Model Classes
- Fetching Data from Internet
- Using FutureBuilder Widget
- Introduction to REST
- Fetching Data from API Server



Peer and Self Assessement by
Abdullah Tajuddin

Abdullah Abdullah Tajuddin	75
Aisyah Siti Nur Aisyah Binti Ahmad Kamal	75
Jailani Ahmad Jailani Bin Saad	45
Amalina Amalina Dasuki	100

Demo App

Continue from the previous project

(navigation_named_routes)

Prepare the Codebase

Clone the source code

```
git clone https://github.com/jumail-utm/http_json
```

Start from any codebase branch

```
git checkout codebase-branch
```

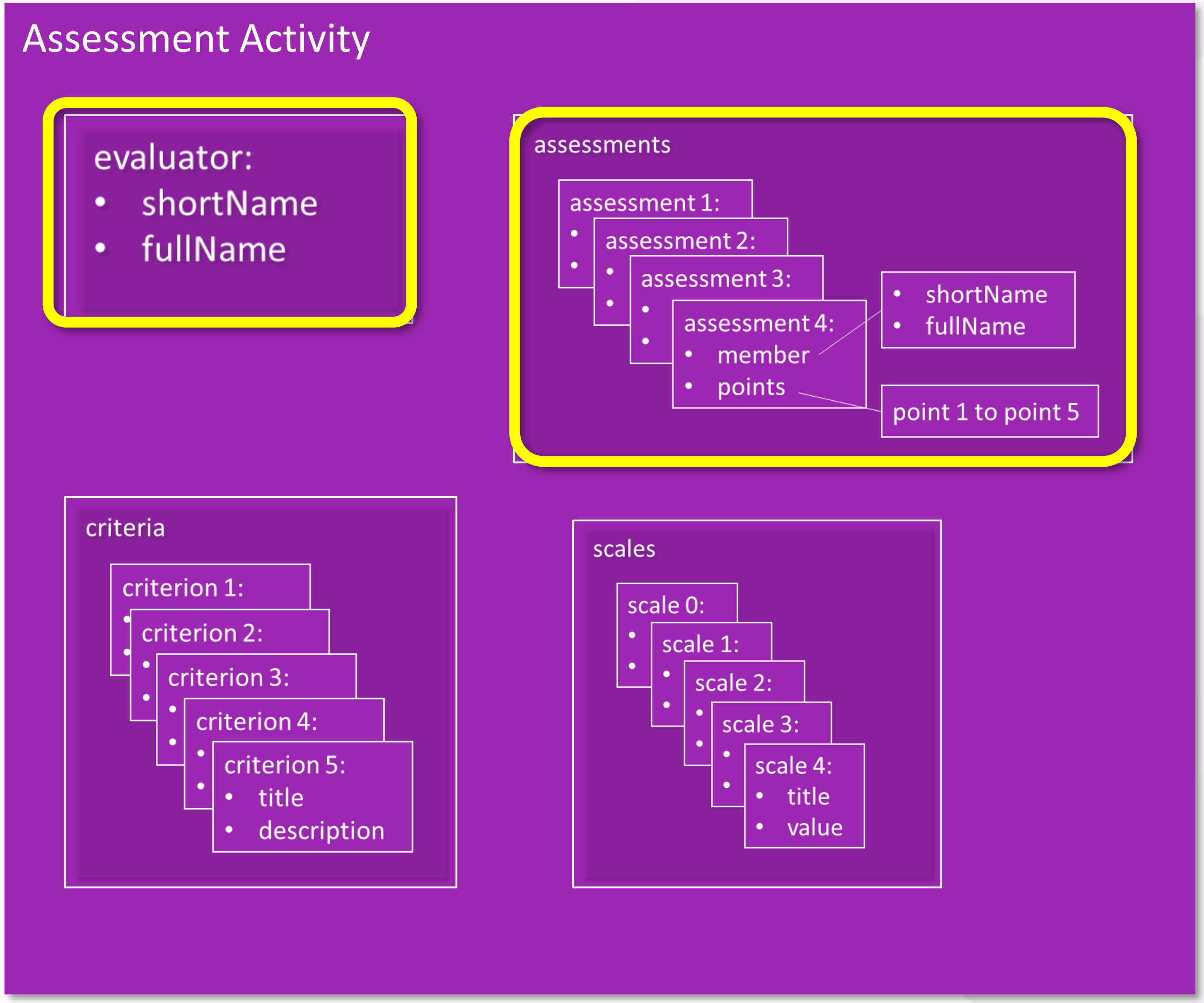
codebase-branch:

- `initial-codebase`
- `fetch-from-internet-codebase`
- `use-futurebuilder-codebase`
- `fetch-from-api-server-codebase`

Code snippet to code along

https://gist.github.com/jumail-utm/9bd2752eb8fed21878da18adb6848ad9

Application Data



Assessment Activity

evaluator:
- shortName
- fullName

assessments

assessment 1:

assessment 2:

assessment 3:

assessment 4:
- member
- points

- shortName
- fullName

point 1 to point 5

criteria

criterion 1:

criterion 2:

criterion 3:

criterion 4:

criterion 5:
- title
- description

scales

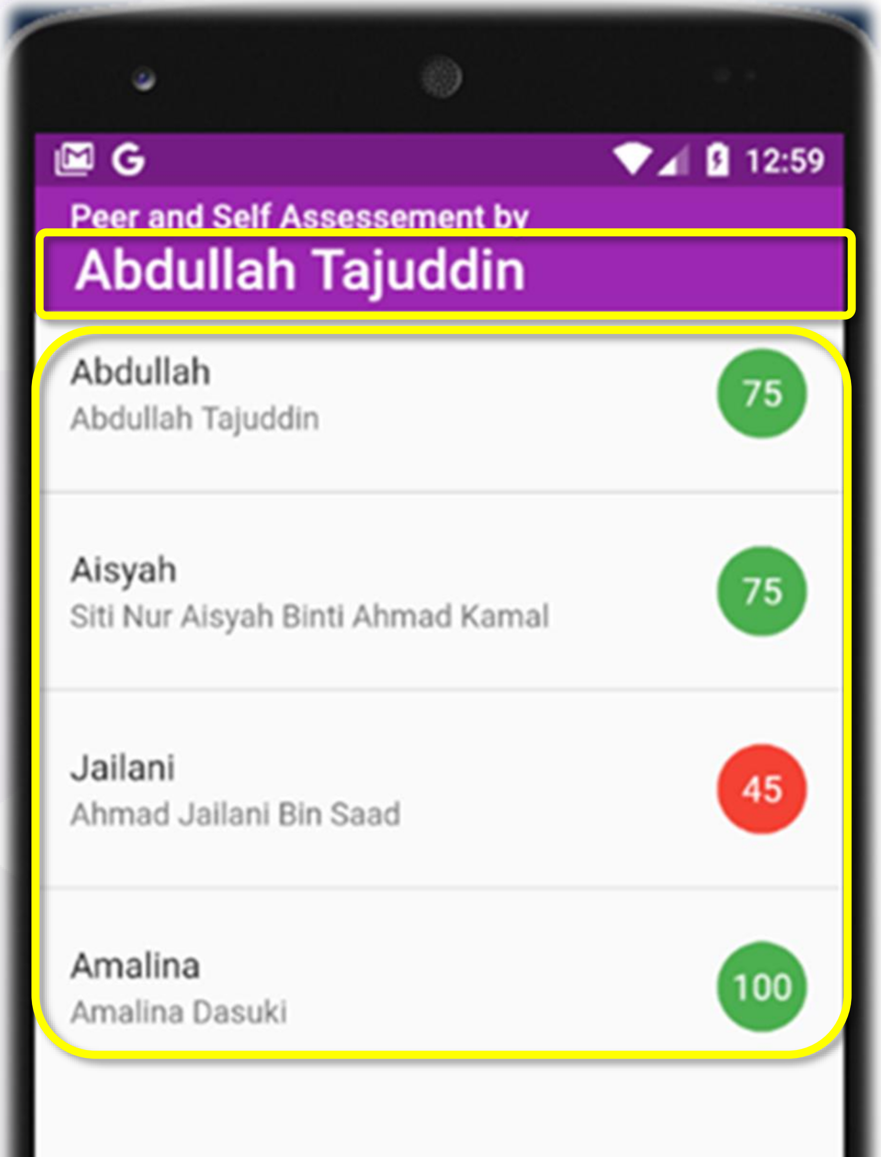
scale 0:

scale 1:

scale 2:

scale 3:

scale 4:
- title
- value

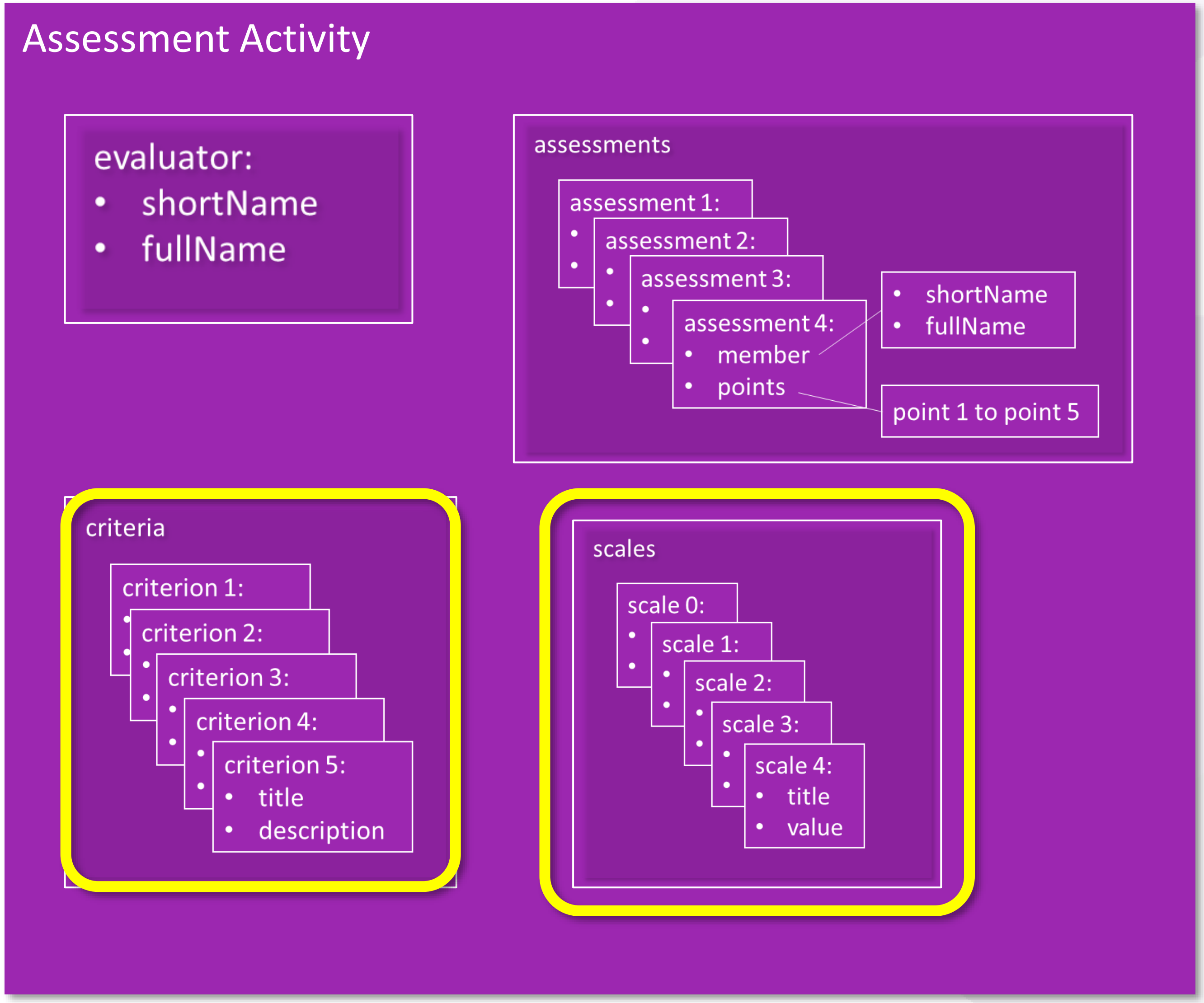


Peer and Self Assessment by

Abdullah Tajuddin

Abdullah Abdullah Tajuddin	75
Aisyah Siti Nur Aisyah Binti Ahmad Kamal	75
Jailani Ahmad Jailani Bin Saad	45
Amalina Amalina Dasuki	100

Application Data (2)



Assessment Activity

evaluator:
- shortName
- fullName

assessments

assessment 1:

assessment 2:

assessment 3:

assessment 4:
- member
- points

- shortName
- fullName

point 1 to point 5

criteria

criterion 1:

criterion 2:

criterion 3:

criterion 4:

criterion 5:
- title
- description

scales

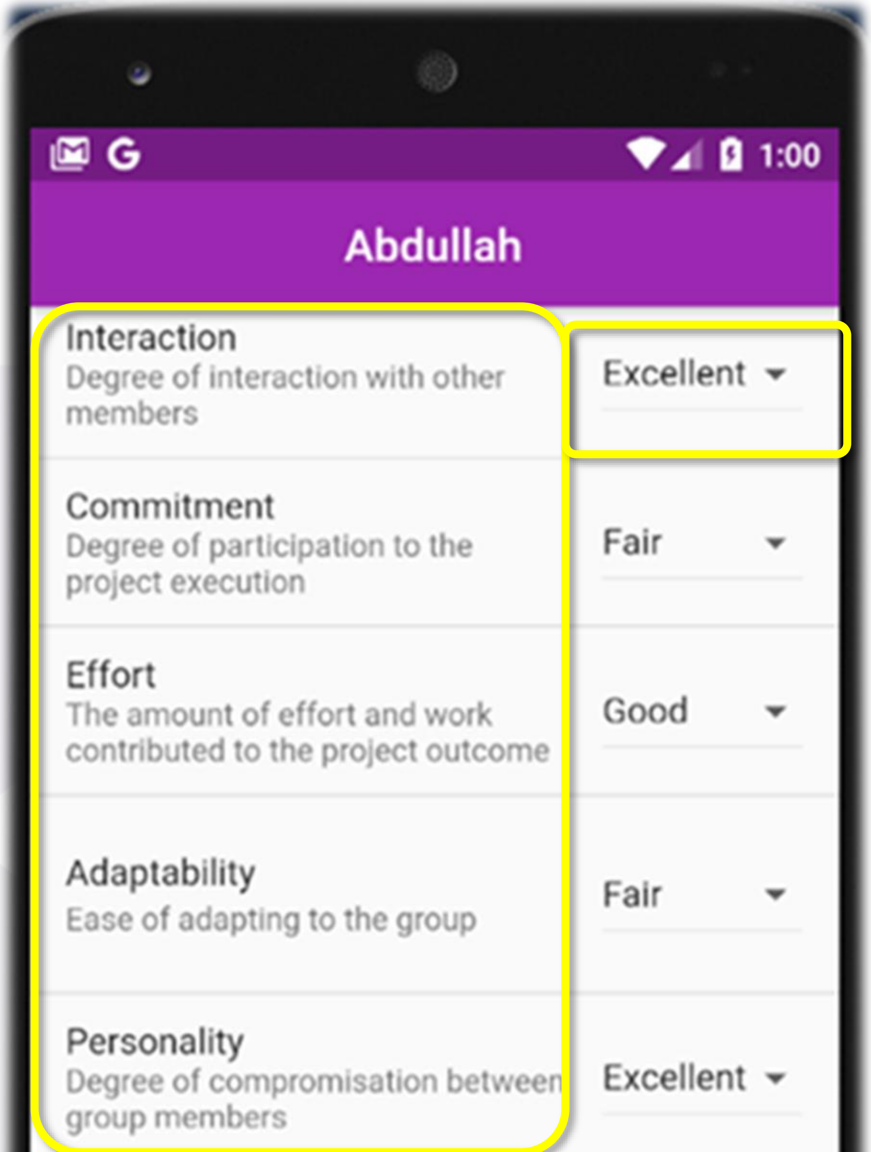
scale 0:

scale 1:

scale 2:

scale 3:

scale 4:
- title
- value



Abdullah

Interaction
Degree of interaction with other members
Excellent

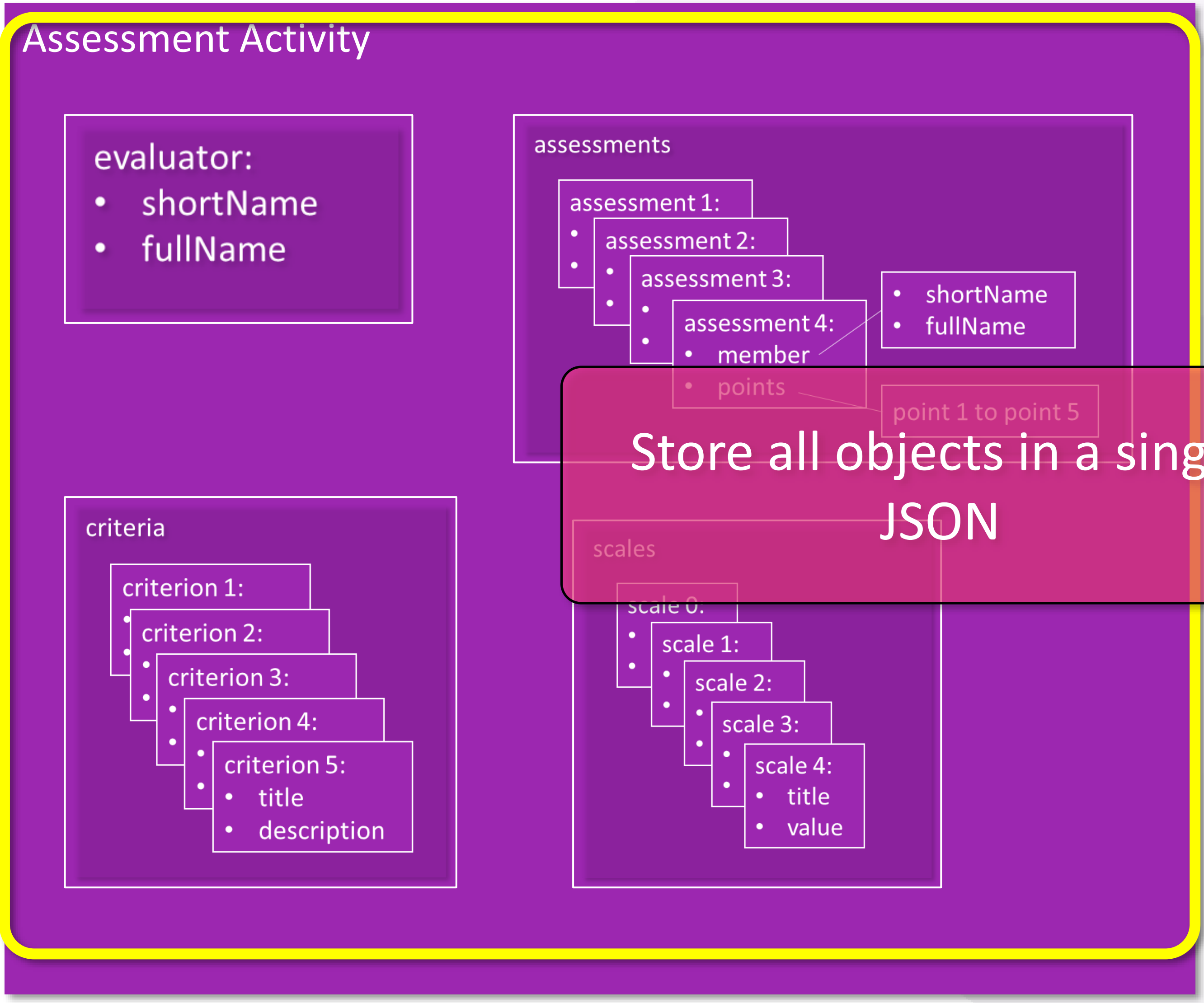
Commitment
Degree of participation to the project execution
Fair

Effort
The amount of effort and work contributed to the project outcome
Good

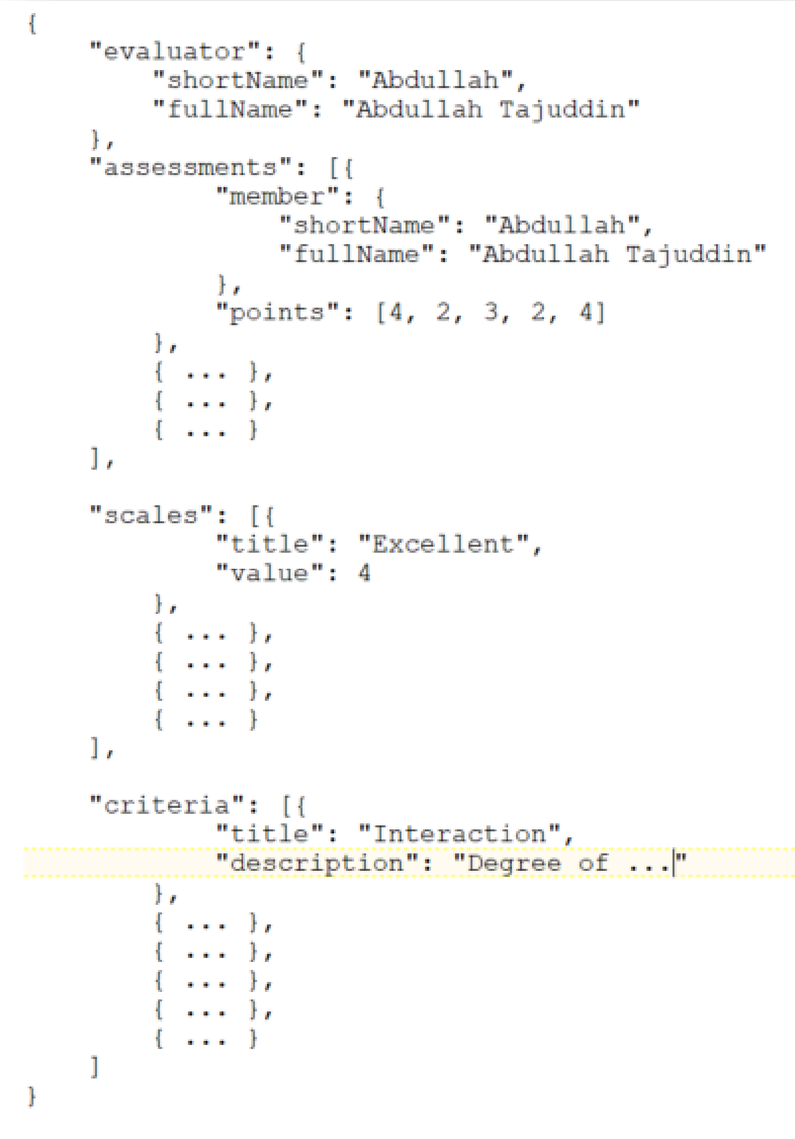
Adaptability
Ease of adapting to the group
Fair

Personality
Degree of compromisation between group members
Excellent

Application Data (3)



Store all objects in a single JSON



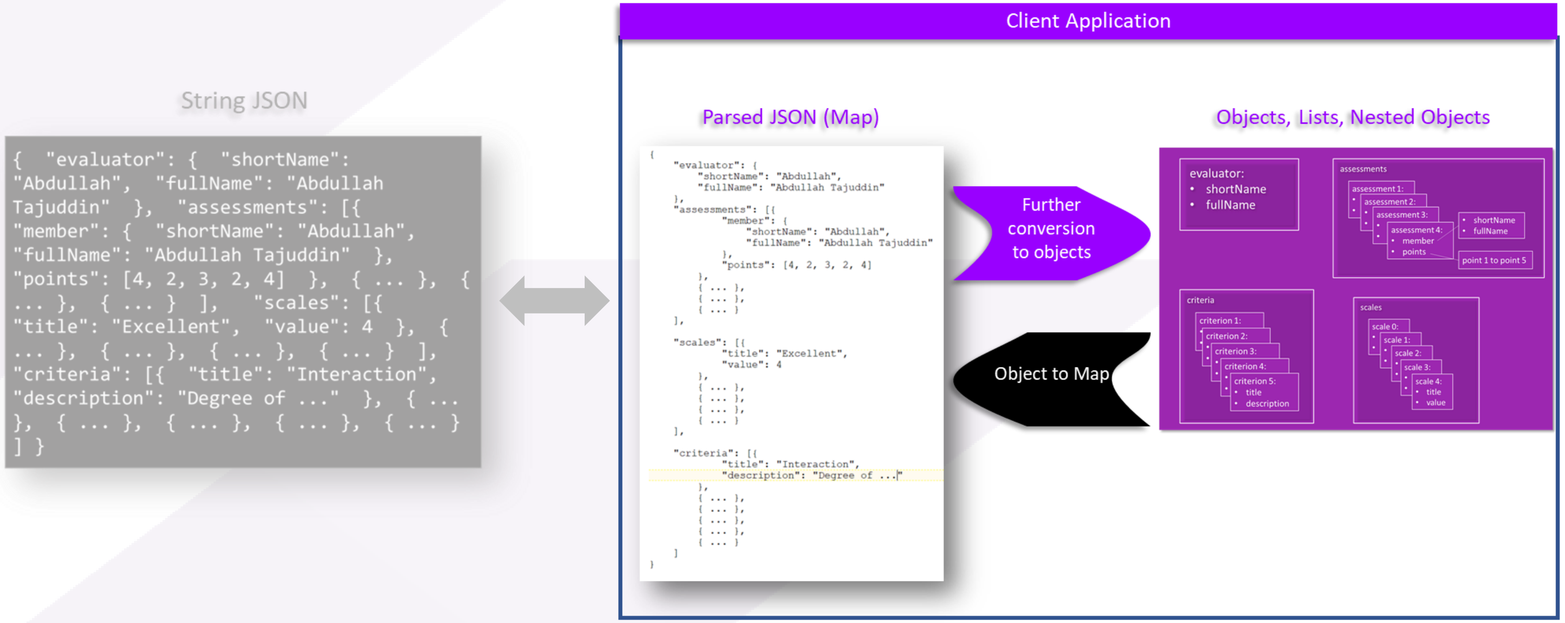
```
{
    "evaluator": {
        "shortName": "Abdullah",
        "fullName": "Abdullah Tajuddin"
    },
    "assessments": [{
            "member": {
                "shortName": "Abdullah",
                "fullName": "Abdullah Tajuddin"
            },
            "points": [4, 2, 3, 2, 4]
        },
        { ... },
        { ... },
        { ... }
    ],

    "scales": [{
            "title": "Excellent",
            "value": 4
        },
        { ... },
        { ... },
        { ... },
        { ... }
    ],

    "criteria": [{
            "title": "Interaction",
            "description": "Degree of ..."
        },
        { ... },
        { ... },
        { ... },
        { ... },
        { ... }
    ]
}
```

Task 1: Add Conversion Methods to the Model Classes

- Add `fromJson()` method
- Add `toJson()` method

- Test JSON decoding (deserialization) with hard-coded parsed JSON data

- Test JSON encoding on the debug console



String JSON

```
{  "evaluator": {  "shortName":
"Abdullah",  "fullName": "Abdullah
Tajuddin"  },  "assessments": [{
"member": {  "shortName": "Abdullah",
"fullName": "Abdullah Tajuddin"  },
"points": [4, 2, 3, 2, 4]  },  { ... },  {
... },  { ... } ],   "scales": [{
"title": "Excellent",  "value": 4  },  {
... },  { ... },  { ... },  { ... }  ],
"criteria": [{  "title": "Interaction",
"description": "Degree of ..."  },  { ...
},  { ... },  { ... },  { ... },  { ... }
] }
```

Client Application

Parsed JSON (Map)

```
{
    "evaluator": {
        "shortName": "Abdullah",
        "fullName": "Abdullah Tajuddin"
    },
    "assessments": [{
            "member": {
                "shortName": "Abdullah",
                "fullName": "Abdullah Tajuddin"
            },
            "points": [4, 2, 3, 2, 4]
        },
        { ... },
        { ... },
        { ... }
    ],

    "scales": [{
            "title": "Excellent",
            "value": 4
        },
        { ... },
        { ... },
        { ... },
        { ... }
    ],

    "criteria": [{
            "title": "Interaction",
            "description": "Degree of ..."
        },
        { ... },
        { ... },
        { ... },
        { ... },
        { ... }
    ]
}
```

Objects, Lists, Nested Objects

Task 2: Fetch Data from Internet

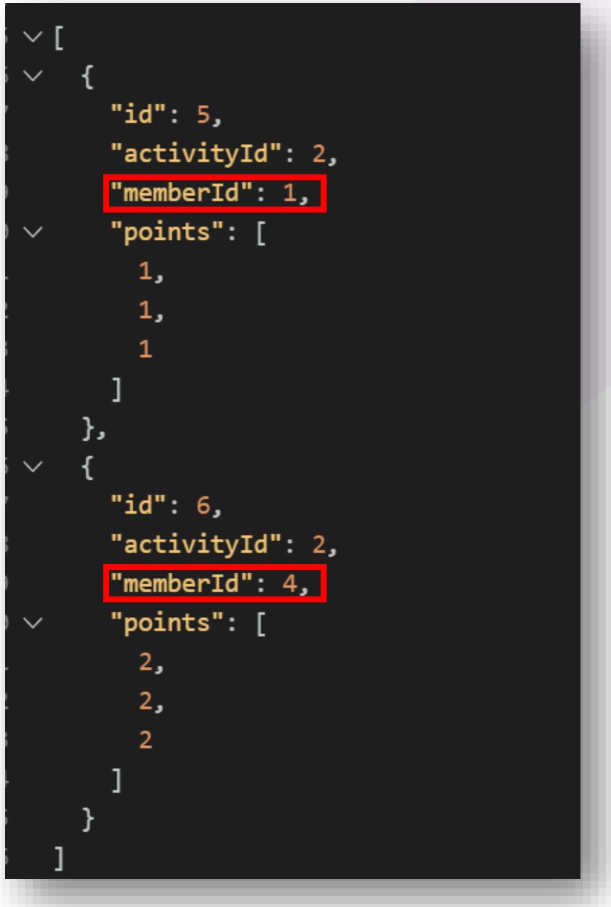
- In this section, we assume the string JSON received by the client, has already been pre-processed by the backend.
- For example all the lookup fields have been resolved to their details data.
- To mimic this, we simply use pre-created JSON file and host it on a web server.

Fetch Data from Internet (2)

An example of pre-processing, lookup field resolution

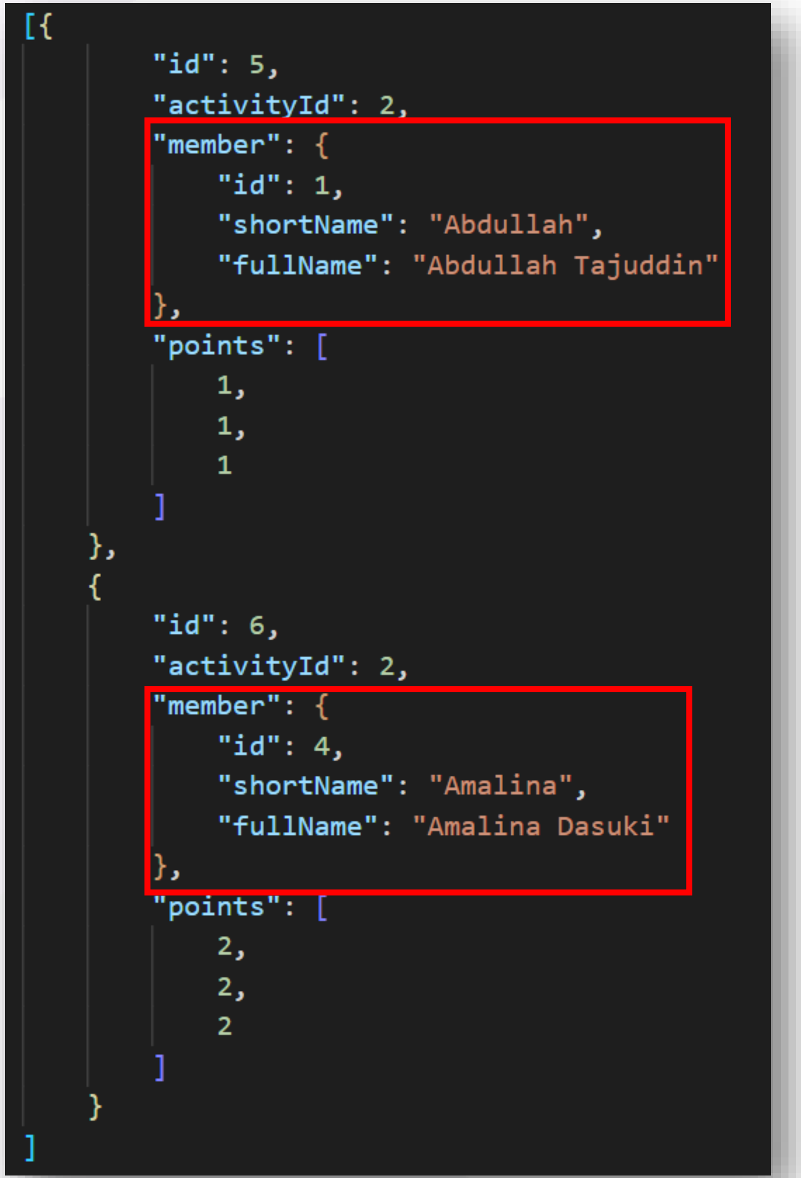
Client request:

http://myserver.com/assessments?activityid=2



```
[
  {
    "id": 5,
    "activityId": 2,
    "memberId": 1,
    "points": [
      1,
      1,
      1
    ]
  },
  {
    "id": 6,
    "activityId": 2,
    "memberId": 4,
    "points": [
      2,
      2,
      2
    ]
  }
]
```

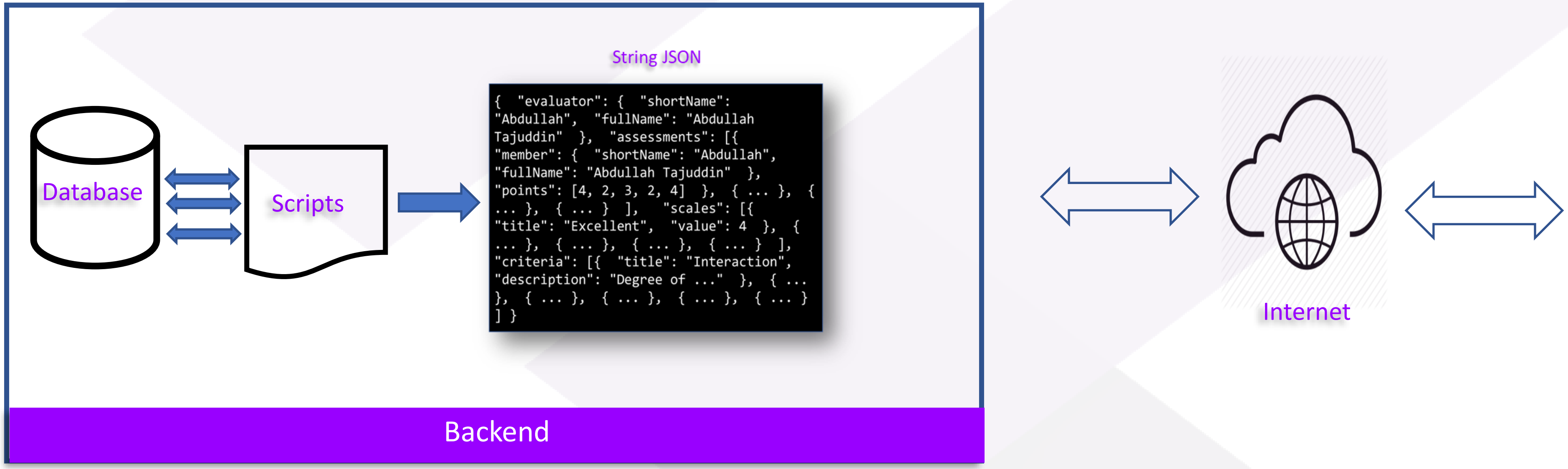
Resolve memberId



```
[{
        "id": 5,
        "activityId": 2,
        "member": {
            "id": 1,
            "shortName": "Abdullah",
            "fullName": "Abdullah Tajuddin"
        },
        "points": [
            1,
            1,
            1
        ]
    },
    {
        "id": 6,
        "activityId": 2,
        "member": {
            "id": 4,
            "shortName": "Amalina",
            "fullName": "Amalina Dasuki"
        },
        "points": [
            2,
            2,
            2
        ]
    }
]
```

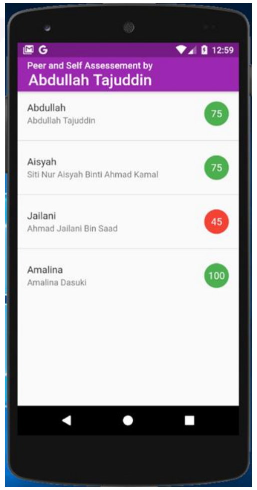
Fetch Data from Internet (3)

- In this section, we assume the lookup field resolution is done by the backend side, rather than the client



String JSON

```
{  "evaluator": {  "shortName":
"Abdullah",  "fullName": "Abdullah
Tajuddin"  },  "assessments": [{
"member": {  "shortName": "Abdullah",
"fullName": "Abdullah Tajuddin"  },
"points": [4, 2, 3, 2, 4]  },  { ... },  {
... },  { ... }  ],   "scales": [{
"title": "Excellent",  "value": 4  },  {
... },  { ... },  { ... },  { ... }  ],
"criteria": [{  "title": "Interaction",
"description": "Degree of ..."  },  { ...
},  { ... },  { ... },  { ... },  { ... }
] }
```


12:59

Peer and Self Assessement by
Abdullah Tajuddin

Abdullah
Abdullah Tajuddin
75

Aisyah
Siti Nur Aisyah Binti Ahmad Kamal
75

Jailani
Ahmad Jailani Bin Saad
45

Amalina
Amalina Dasuki
100

Flutter App

Fetch Data from Internet (4)

- Host the JSON data online

  http://www.mocky.io

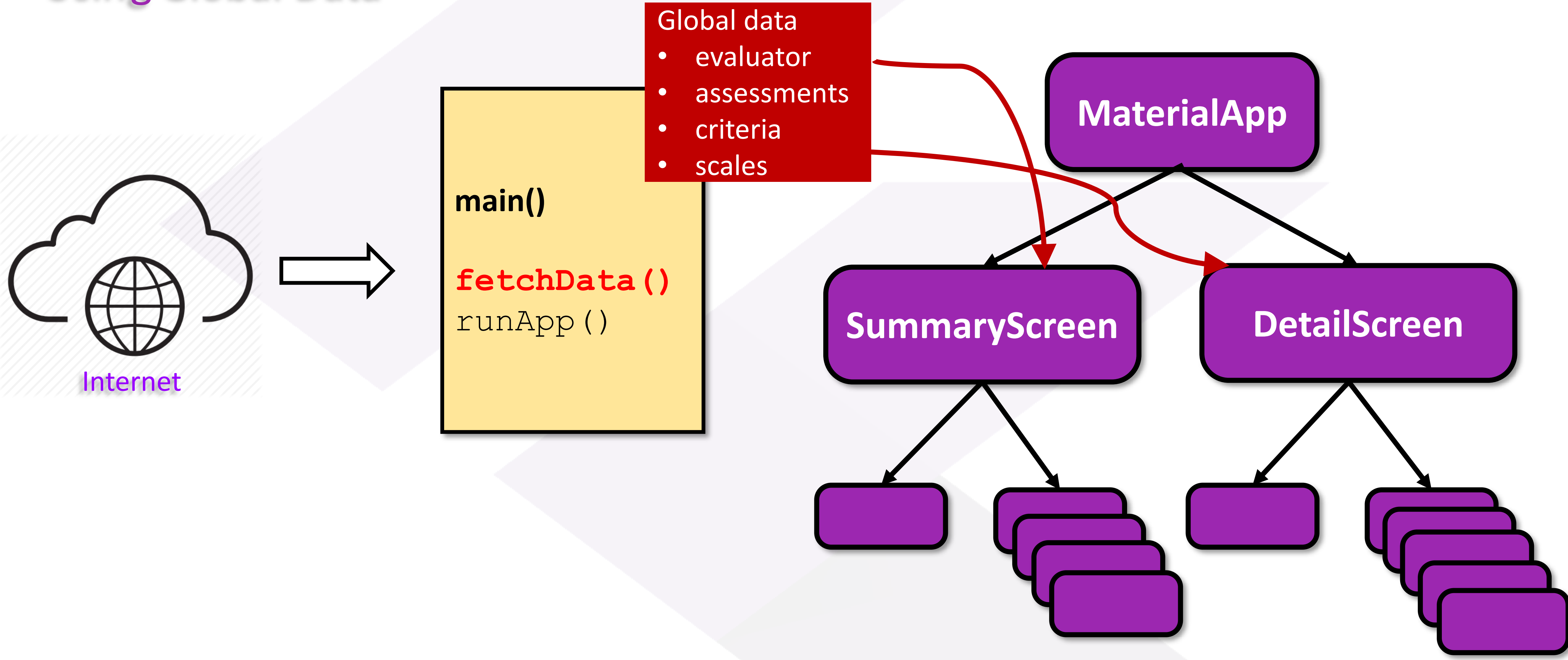
  Pre-created online data

  http://www.mocky.io/v2/5ea539bd3000005900ce2e8f

- Step-by-step
  - Using global data
  - Using passing data approach
  - Use the `FutureBuilder` widget to build the main screen

Fetch Data from Internet (5)

Using Global Data

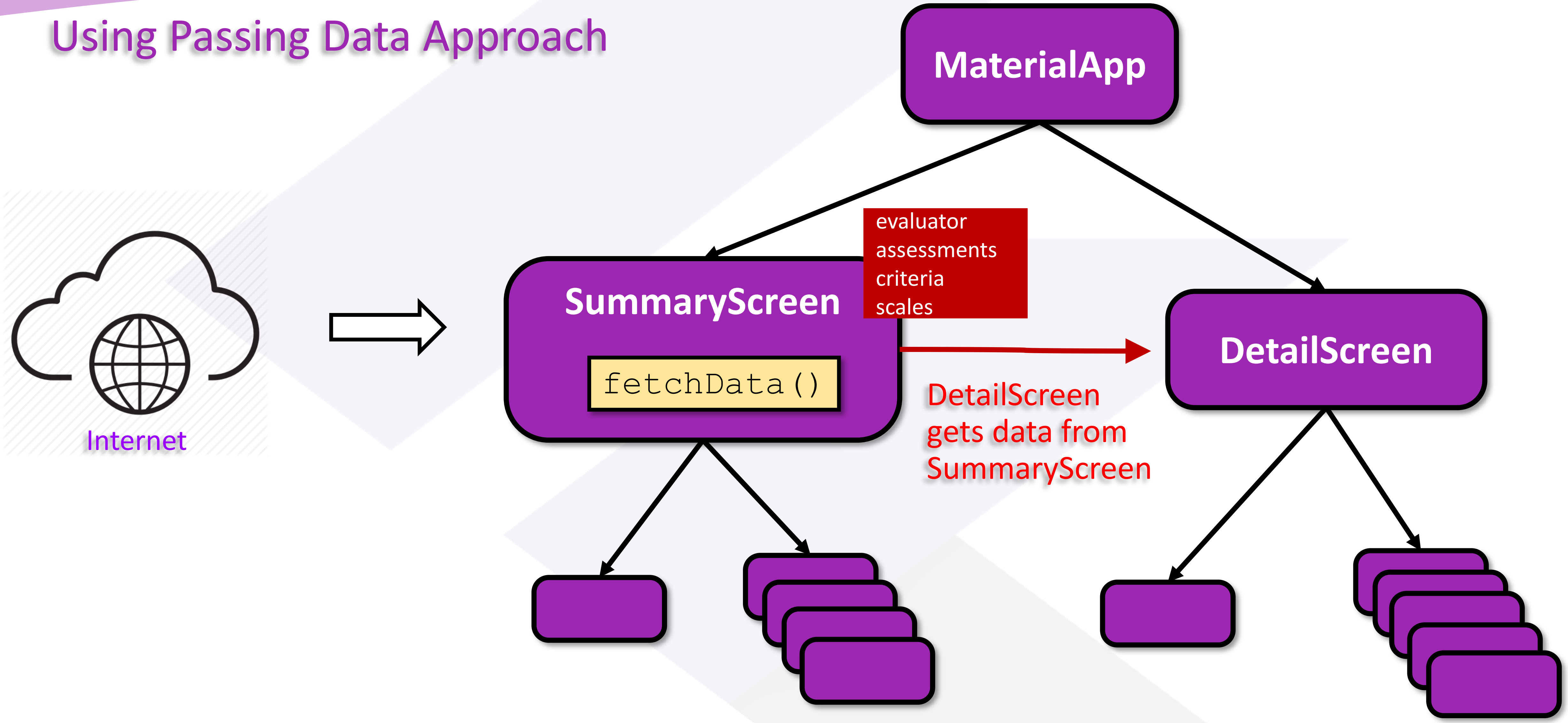


Global data
- evaluator
- assessments
- criteria
- scales

```
main()

fetchData()
runApp()
```

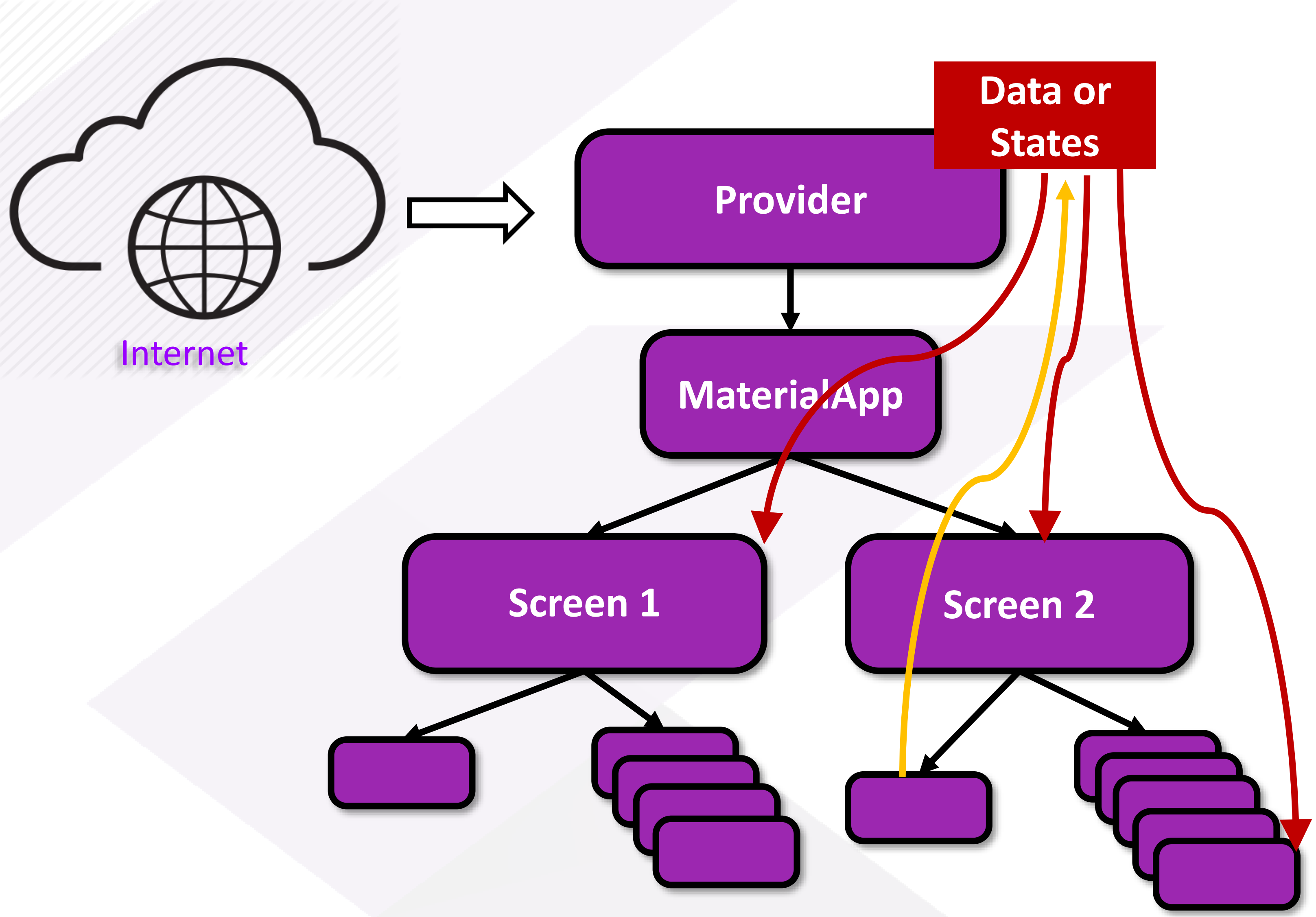
Fetch Data from Internet (6)

Using Passing Data Approach



Fetch Data from Internet (7)

A better approach with **Provider** (Not covered in this demo)



Task 3: Use FutureBuilder Widget

- Use the `FutureBuilder` widget to build the main screen (`SummaryScreen`)

- This widget will get triggered to perform its `build()` method when it receives a Future data.

- Two important properties to setup:
  - `future` : the Future data that this widget is depending on. In our case, it will be the result of the `http.get()` call.
  - `builder` : what this widget need to build when a future data arrives.

Task 4: Fetch Data from API Server

API Server

JSON Server

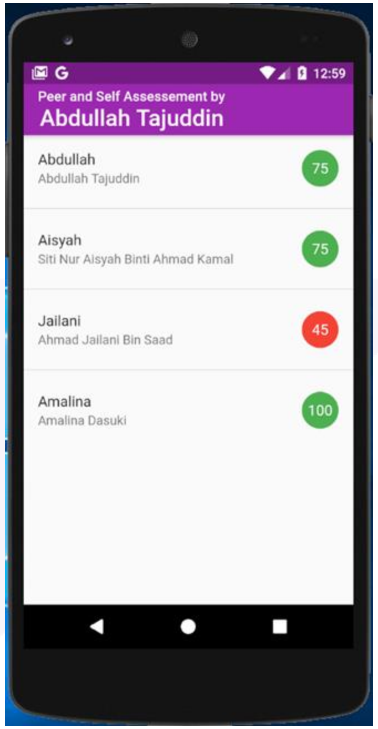
NoSQL-like Database

Client-server communication:

HTTP

JSON

REST



Peer and Self Assessement by

Abdullah Tajuddin

Abdullah
Abdullah Tajuddin
75

Aisyah
Siti Nur Aisyah Binti Ahmad Kamal
75

Jailani
Ahmad Jailani Bin Saad
45

Amalina
Amalina Dasuki
100

Introduction to

REST

- What is REST?
- Why REST?
- REST Guidelines
- Learning REST by Examples

What is REST?

- REST stands for Representational State Transfer
- Architecture style for the communication between client and server applications
- Works on top of a stateless, client-server protocol mainly HTTP
- Language agnostic
- Facilitate the CRUD operations - the communication part, between the client and server

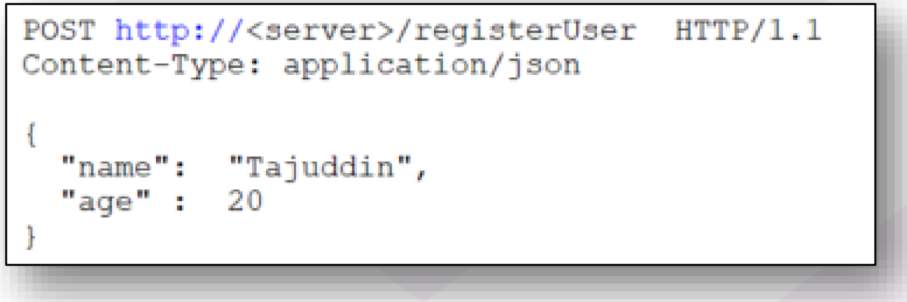
Why REST?

- Let’s revisit HTTP.

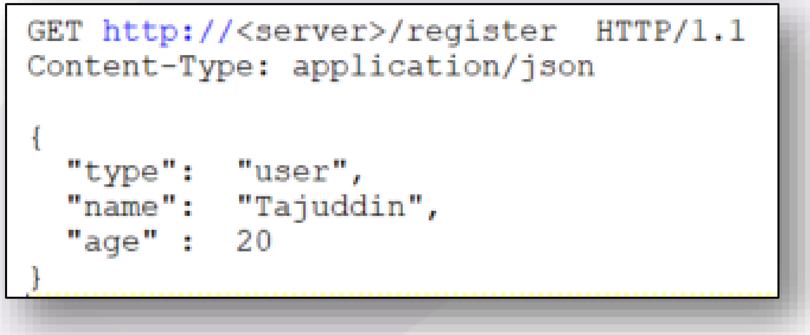
  Example: To register a new user, you have many ways with HTTP

  http://<server>/registerUser.php?name=Tajuddin&age=20

  http://<server>/register.php?type=user&name=Tajuddin&age=20



```
POST http://<server>/registerUser  HTTP/1.1
Content-Type: application/json

{
  "name":  "Tajuddin",
  "age" :  20
}
```



```
GET http://<server>/register  HTTP/1.1
Content-Type: application/json

{
  "type":  "user",
  "name":  "Tajuddin",
  "age" :  20
}
```

- You can simply choose only one HTTP Request (e.g. POST) for all CRUD operations – NOT a good practice, violates the HTTP guidelines

- REST lets us use HTTP in a more consistent way

REST Guidelines

- Accept and respond with JSON
- Use nouns (or objects) instead of verbs (or actions) in endpoint path
  - The action has been indicated by the HTTP Request method

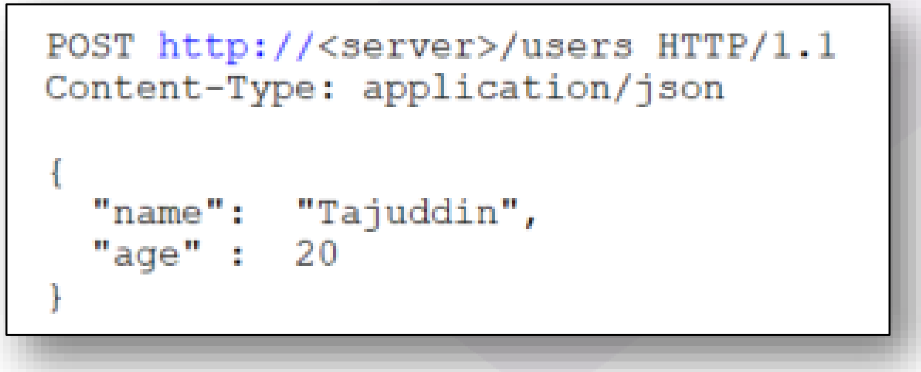
HTTP Request Methods

- GET is used to request data from a specified resource.
- POST is used to request for the server to create a new resource.
- PUT is used to request for the server to replace / update a resource.
- PATCH is similar to PUT, but only update specified attributes of a resource.
- HEAD is similar to GET, but returns only the response header.
- DELETE is used to request for the server to delete a resource.
- Other methods: TRACE, OPTIONS, CONNECT

REST Guidelines (2)

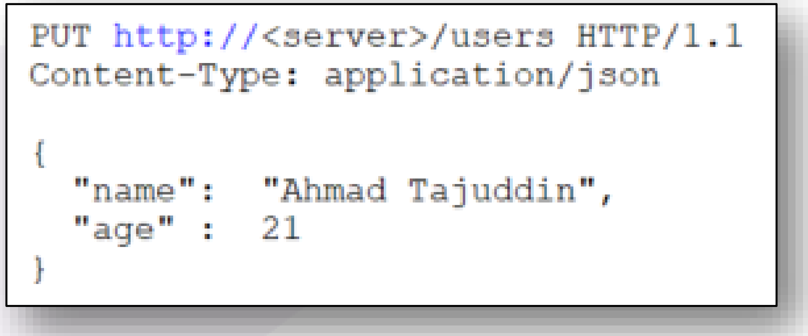
Thus, to register a new user:

- name the path as `/users` instead of `/registerUser`
- always use a POST method



```
POST http://<server>/users HTTP/1.1
Content-Type: application/json

{
  "name":  "Tajuddin",
  "age" :  20
}
```

To update an existing user:



```
PUT http://<server>/users HTTP/1.1
Content-Type: application/json

{
  "name":  "Ahmad Tajuddin",
  "age" :  21
}
```

To retrieve a user (for a given id):

```
GET http://<server>/users/5 HTTP/1.1
```

To delete a user (for a given id):

```
DELETE http://<server>/users/5 HTTP/1.1
```

REST Guidelines (3)

Name collections with plural nouns

- To reflect with tables in the database. A table consists a list of entries

Examples:

Get all users:

```
GET http://<server>/users HTTP/1.1
```

Get the user whose id 5:

```
GET http://<server>/users/5 HTTP/1.1
```

Get the user whose name Tajuddin:

```
GET http://<server>/users?name=Tajuddin HTTP/1.1
```

REST Guidelines (4)

Append a nested resource as the name of the path that comes after the parent resource.

Examples:

Get all the contact information for a given user:
e.g. contacts: phone, mobile, main email, second email, etc

```
GET http://<server>/users/5/contacts HTTP/1.1
```

REST Guidelines (5)

Allow filtering, sorting and pagination

Examples:

Filtering: Get the user whose a given name and age:

```
GET http://<server>/users?name=Tajuddin&age=20 HTTP/1.1
```

Sorting: Get all users sorted by age (youngest first) followed by names (in alphabetical order)

+ means ascending and – means descending order

```
GET http://<server>/users?sort=-age,+name HTTP/1.1
```

Pagination: Get users from page 2

```
GET http://<server>/users?page=2 HTTP/1.1
```

REST Guidelines (6)

Error handling – Return standard error codes

HTTP Response Status code

- 1XX: Information
- 2XX: Successful
- 3XX: Redirection
- 4XX: Client Error
- 5XX: Server Error

Example:

- 400 Bad Request – The client-side input fails validation.
- 401 Unauthorized – The user isn't not authorized to access a resource.
- 403 Forbidden – The user is authenticated, but it's not allowed to access a resource.
- 404 Not Found – A resource is not found.
- 500 Internal server error – A generic server error.
- 502 Bad Gateway – An invalid response from an upstream server.
- 503 Service Unavailable –Something unexpected happened on server side

REST Guidelines (7)

- Maintain good security practices
  - Use SSL/TLS
- Cache data to improve performance
  - Caching allows retrieving data faster
- Versioning the APIs
  - To prevent breaking the clients should a new version is implemented
  - To phase out old endpoints gradually instead of forcing everyone to the new API
  - Common strategy, add version number as part of the endpoint path, e.g. `/v1/`, `/v2/`

REST Examples

- Prepare the codebase

```
git clone https://github.com/jumail-utm/http_json
git checkout fetch-from-api-server-codebase
```

- Setup fake API server
  - Install node.js: https://nodejs.org/en/download
  - Install JSON Server: https://github.com/typicode/json-server
  - Create JSON database
  - Run the server

```
json-server --host your-pc-IP-address db.json
```

Note: Run `ipconfig` on Command Prompt to check your PC's IP address. Do not use localhost.

REST Examples (2)

- Alternatively, you can use this online JSON server. However, it will not reflect data update permanently.

  https://my-json-server.typicode.com/jumail-utm/http_json

- Install VSCode extension REST Client to test the API server.
  - Alternative to using REST client, you can use Postman.

    https://www.postman.com/downloads/

- Open the file rest_client/learning_rest_examples.http into VS Code.

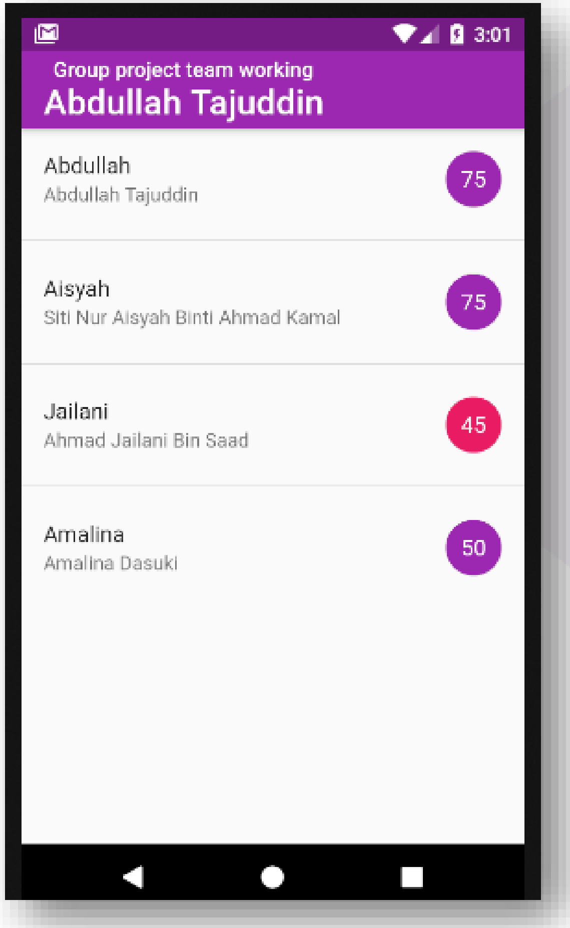
Application Data (Upgraded Version)

- Supports multiple assessment activities, such as
  - Activity 1: Group Project Team Working Assessment
  - Activity 2: Pair Programming Exercise Assessment
- Each assessment activity has its own assessment criteria and scales

- Use a NoSQL database.
  - A database consists of a list of collections (like tables in SQL). Our collections are activities, assessments, forms, and users
  - Each collection contains a list of documents (like rows or records in SQL)

Application Data (2)

Example 1



Group project team working

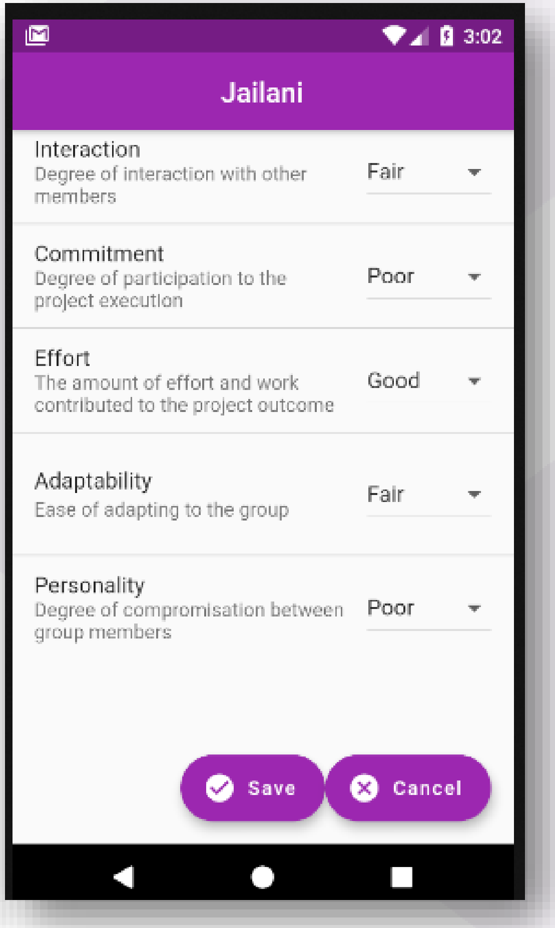
Abdullah Tajuddin

Abdullah
Abdullah Tajuddin
75

Aisyah
Siti Nur Aisyah Binti Ahmad Kamal
75

Jailani
Ahmad Jailani Bin Saad
45

Amalina
Amalina Dasuki
50



Jailani

Interaction
Degree of interaction with other members
Fair

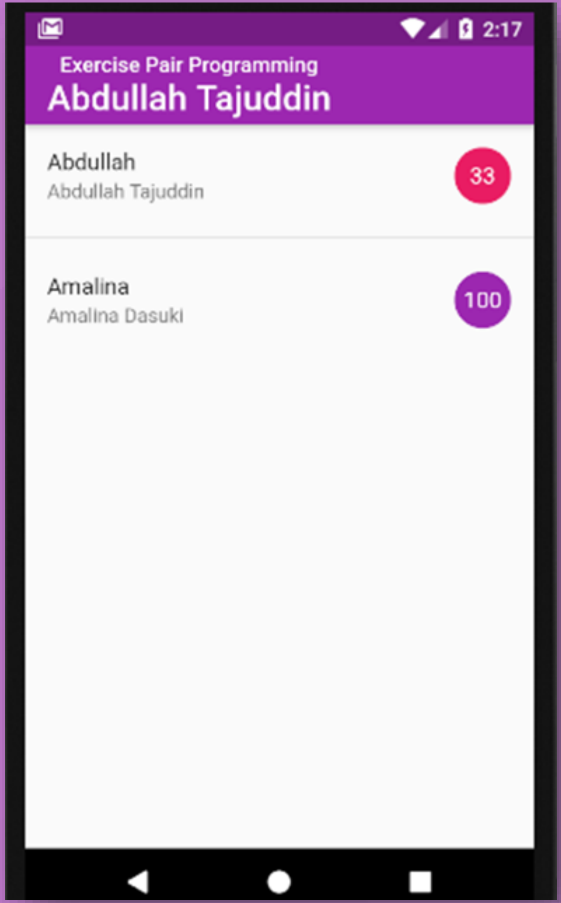
Commitment
Degree of participation to the project execution
Poor

Effort
The amount of effort and work contributed to the project outcome
Good

Adaptability
Ease of adapting to the group
Fair

Personality
Degree of compromisation between group members
Poor

Example 2

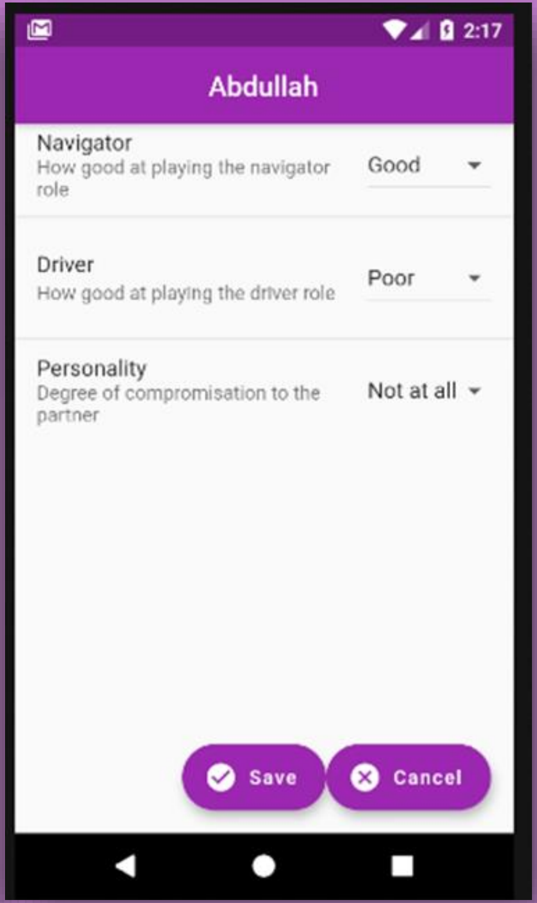


Exercise Pair Programming

Abdullah Tajuddin

Abdullah
Abdullah Tajuddin
33

Amalina
Amalina Dasuki
100



Abdullah

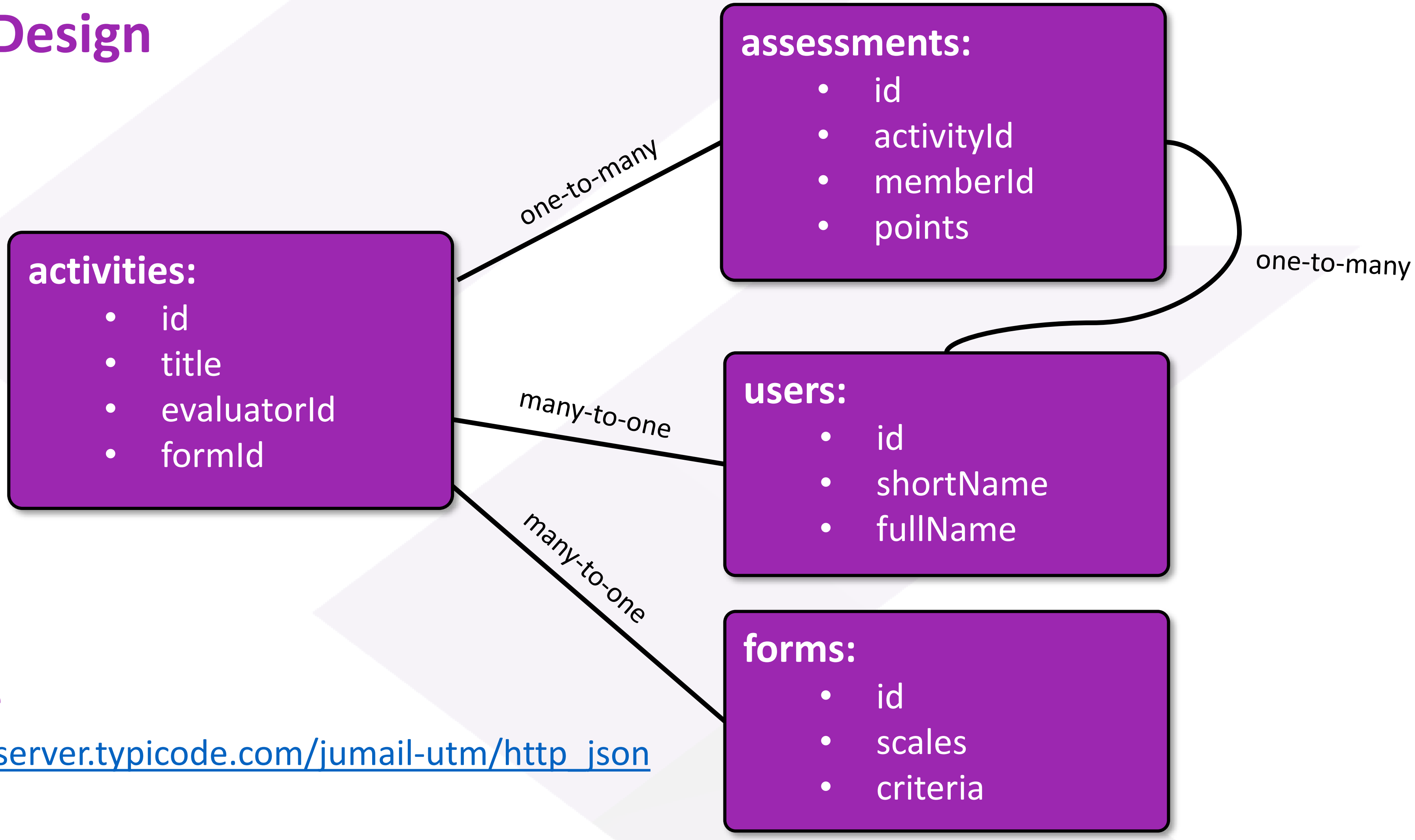
Navigator
How good at playing the navigator role
Good

Driver
How good at playing the driver role
Poor

Personality
Degree of compromisation to the partner
Not at all

Application Data (3)

Database Design



activities:

- id
- title
- evaluatorId
- formId

assessments:

- id
- activityId
- memberId
- points

users:

- id
- shortName
- fullName

forms:

- id
- scales
- criteria

JSON Database

https://my-json-server.typicode.com/jumail-utm/http_json

Application Data (4)

Testing the Database on Local Server

- Prepare the codebase

```
git clone https://github.com/jumail-utm/http_json
git checkout fetch-from-api-server-codebase
```

- Setup fake API server
  - Install node.js: https://nodejs.org/en/download
  - Install JSON Server: https://github.com/typicode/json-server
  - Create JSON database
  - Run the server

```
json-server --host your-pc-IP-address db.json
```

Note: Run `ipconfig` on Command Prompt to check your PC's IP address. Do not use localhost.

Application Data (5)

Testing the Database on Local Server

- Alternatively, you can use this online JSON server. However, it will not reflect data update permanently.

  https://my-json-server.typicode.com/jumail-utm/http_json

- Install VSCode extension REST Client to test the API server.
  - Alternative to using REST client, you can use Postman.

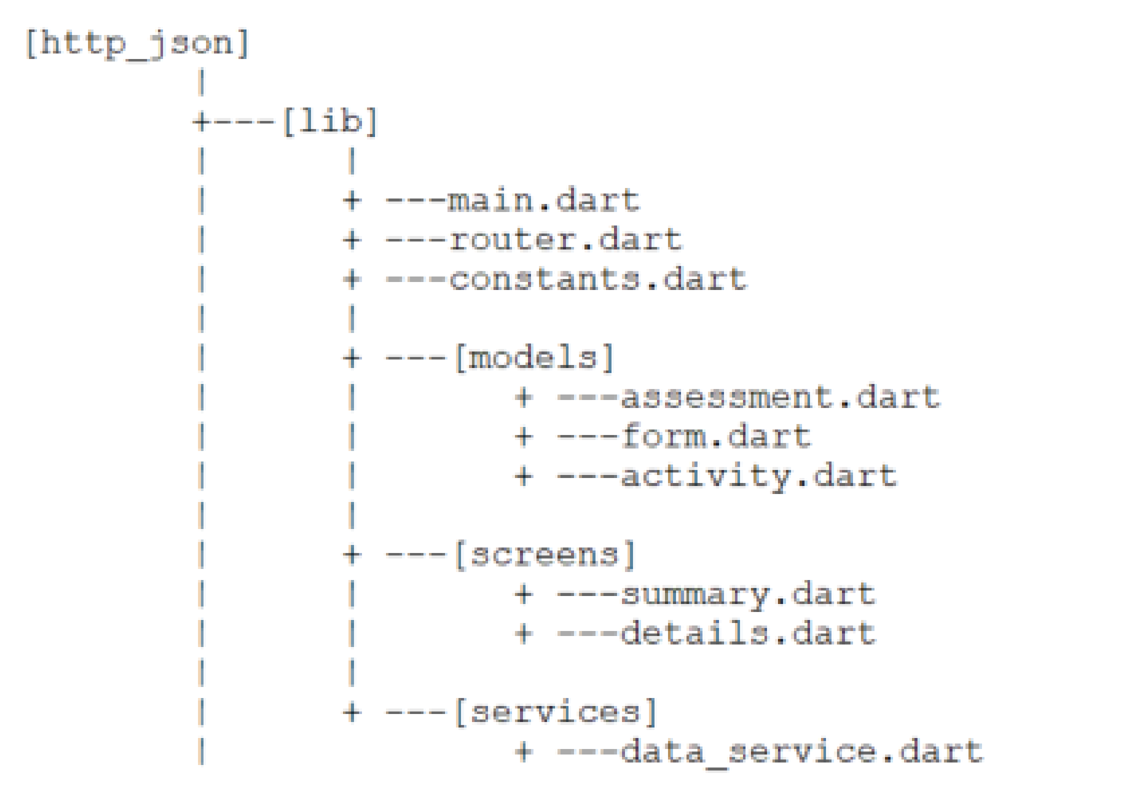
    https://www.postman.com/downloads/

- Open the file rest_client/test_database.http into VS Code.

Task 4: Fetch Data from API Server

- Update model classes to reflect to the changes of upgraded database
- Add DataService class to handle REST requests for database manipulation
- Update only SummaryScreen to use the DataService

Project File Structure



```
[http_json]
        |
        +---[lib]
        |       |
        |       + ---main.dart
        |       + ---router.dart
        |       + ---constants.dart
        |       |
        |       + ---[models]
        |       |       + ---assessment.dart
        |       |       + ---form.dart
        |       |       + ---activity.dart
        |       |
        |       + ---[screens]
        |       |       + ---summary.dart
        |       |       + ---details.dart
        |       |
        |       + ---[services]
        |               + ---data_service.dart
```

Summary

- Fetch data from internet
- Use FutureBuilder widget
- Data access with REST
- Fetch Data from API Server
- Structure the project code: screens, models, and services