EEE 4482 Server Installation and Programming

Worksheet 7d – Interaction between UI and Database

Objective: To create HTTP requests for communication between a web application and a database via API services

Tools: Windows PC

Software: Oracle VM Virtual Box version 6.1.12
CentOS 7
Filezilla
Visual Studio Code
Chrome

Topics covered:

- Sending HTTP requests to API server
- Extracting information from API responses for GUI display
- Validation of the input data

Component list:

mCentos7min-trial01_AfterWS07b.zip - A virtual machine zip file with Apache, MariaDB and PHP7.4 installed

HTTP Request

An HTTP request is a communication message transmitted from a client to a server over the Internet using the Hypertext Transfer Protocol (HTTP). It serves the purpose of requesting a specific action or resource from the server.

The HTTP request comprises several essential components. Firstly, the request method indicates the intended operation the client wants the server to perform. Commonly used methods include *GET*, *POST*, *PUT*, and *DELETE*. For instance, a *GET* request is employed to retrieve a resource, while a *POST* request is utilized to submit data for processing.

The request URL specifies the location of the desired resource on the server. It typically starts with the protocol (e.g., "http://" or "https://") followed by the domain name or IP address and the path to the resource.

Request headers contain supplementary information pertaining to the request. They convey details such as the client's preferred content type, language, and authentication credentials.

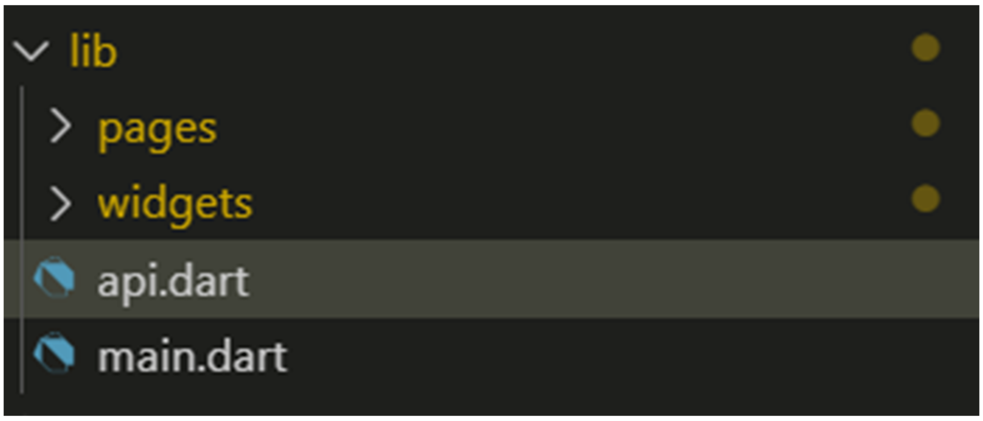
In certain request methods like *POST* and *PUT*, the request may include a request body. This optional component carries data that necessitates transmission to the server, such as form input or a JSON payload.

Upon receiving an HTTP request, the server processes it and generates an appropriate response. The response includes a status code that indicates the outcome of the request, such as a **200 status** code denoting a successful request or a 404 code indicating a not found resource. Additionally, the response headers provide further information, while the response body contains the requested resource or any relevant error messages.

In summary, HTTP requests serve as the foundation for communication between clients and servers on the World Wide Web(WWW). They facilitate the retrieval and manipulation of web resources.

Getting book records via API

1. Create a file named as "***api.dart***" in the "***eee4482_elibrary/lib/***" directory.



2. Add the template codes to the "*api.dart*" file as below.

```dart
import 'package:http/http.dart' as http; //flutter pub add http
import 'package:http/browser_client.dart'; //flutter pub add http
import 'dart:convert';

final String API_ENDPOINT = "http://192.168.XX.XX"; // your API server address

http.Client getCredentialsClient() {
  //create a client object for HTTP request
  http.Client client = http.Client();
  if (client is BrowserClient) {
    client.withCredentials = true;
  }
  return client;
}
```

3. Install "*http*" package by input the following command in terminal.

```
flutter pub add http
```

4. Write a function “*apiGetAllBooks*” below the template codes. It is used to get all book records from the server by a HTTP Get request.

```
Future<List<dynamic>> apiGetAllBooks() async {
  var client = getCredentialsClient();
  final response = await client.get(Uri.parse(API_ENDPOINT + '/api/books/all'));
  client.close();

  List<dynamic> bookList = [];

  if (response.statusCode == 200) {
    try {
      print(response.body); //for debug
      var jsonResponse = jsonDecode(response.body);
      bookList = jsonResponse;
    } catch (e) {
      bookList = [];
    }
  } else {
    throw Exception('Failed to load data');
  }

  return bookList;
}
```

5. Import “api.dart” to the “booklist_page.dart” file.

```
import 'package:flutter/material.dart';
import '../widgets/book_form.dart';
import '../widgets/navigation_frame.dart';
import '../api.dart';
```

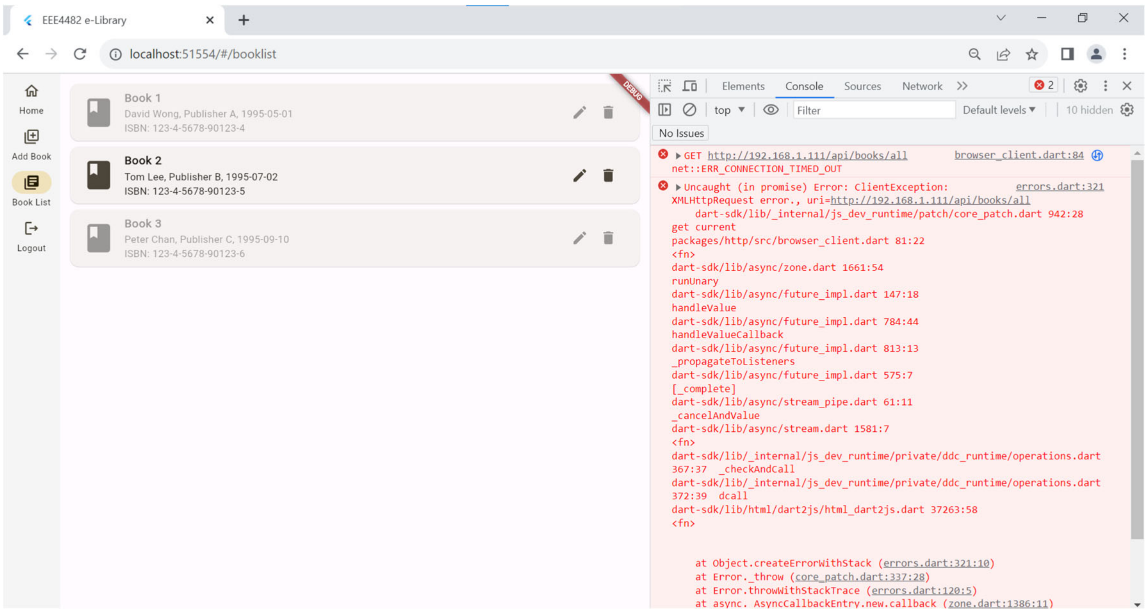
6. Call the function “*apiGetAllBooks*” in the build function of BookListPage as below.

```
class _BooklistPageState extends State<BooklistPage> {
  @override
  Widget build(BuildContext context) {
    apiGetAllBooks(); //<<<< try your api function
    return NavigationFrame(
      selectedIndex: 2,
      child: Padding(
          padding: EdgeInsets.all(10), child: createBookLists(widget.booklist)),
    );
  }
```

7. Run your application in debug mode by input the command below in terminal.

```
Flutter run -d chrome
```

8. Go to BookList page and open the console of dev tool in chrome. You will see the following error.



```
▶GET http://192.168.1.111/api/books/all          browser_client.dart:84
net::ERR_CONNECTION_TIMED_OUT
▶Uncaught (in promise) Error: ClientException:          errors.dart:321
XMLHttpRequest error., uri=http://192.168.1.111/api/books/all
    dart-sdk/lib/_internal/js_dev_runtime/patch/core_patch.dart 942:28
get current
packages/http/src/browser_client.dart 81:22
<fn>
dart-sdk/lib/async/zone.dart 1661:54
runUnary
dart-sdk/lib/async/future_impl.dart 147:18
handleValue
dart-sdk/lib/async/future_impl.dart 784:44
handleValueCallback
dart-sdk/lib/async/future_impl.dart 813:13
_propagateToListeners
dart-sdk/lib/async/future_impl.dart 575:7
[_complete]
dart-sdk/lib/async/stream_pipe.dart 61:11
_cancelAndValue
dart-sdk/lib/async/stream.dart 1581:7
<fn>
dart-sdk/lib/_internal/js_dev_runtime/private/ddc_runtime/operations.dart
367:37  _checkAndCall
dart-sdk/lib/_internal/js_dev_runtime/private/ddc_runtime/operations.dart
372:39  dcall
dart-sdk/lib/html/dart2js/html_dart2js.dart 37263:58
<fn>


    at Object.createErrorWithStack (errors.dart:321:10)
    at Error._throw (core_patch.dart:337:28)
    at Error.throwWithStackTrace (errors.dart:120:5)
    at async._AsyncCallbackEntry.new.callback (zone.dart:1386:11)
```

The expected json results are not received, but some error messages are shown in the terminal.

What is CORS issue?

During web debugging on localhost, CORS (Cross-Origin Resource Sharing) issues may arise. CORS is a security mechanism implemented by web browsers to restrict cross-origin requests made by web pages. It ensures that resources requested from a different domain, port, or protocol are only accessible if explicitly allowed by the server.

There are some methods to address CORS issues during local development:

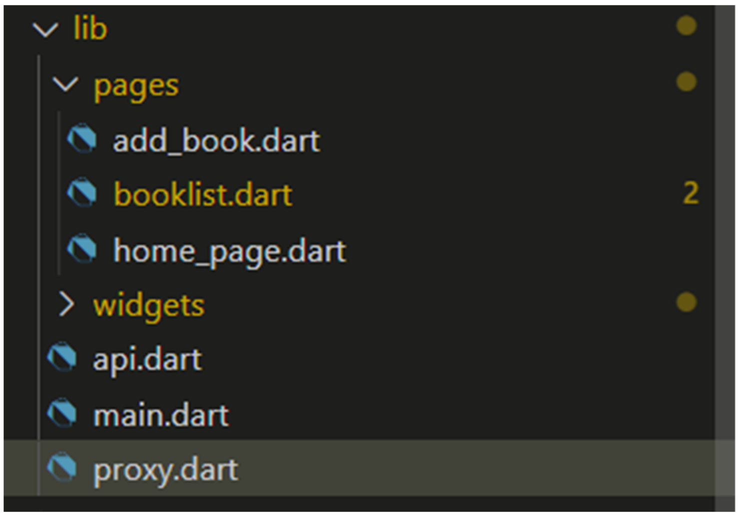
- Enable CORS on the server: Configure the server to include the necessary CORS headers in its responses. These headers typically include Access-Control-Allow-Origin, Access-Control-Allow-Methods, Access-Control-Allow-Headers, and others. By specifying the appropriate values in these headers, the server instructs the browser to allow cross-origin requests.
- Proxy server: Set up a proxy server that acts as an intermediary between your web application and the server you're making requests to. The proxy server can handle the cross-origin requests and forward them to the intended server. This bypasses CORS restrictions as the requests are made from the same origin.
- Browser extensions: Install browser extensions designed to disable CORS restrictions during development. These extensions temporarily override the browser's default behavior and allow cross-origin requests. However, exercise caution and limit their use to development environments only, as they can introduce security vulnerabilities if used in a production setting.

Once your web application is deployed to a production environment with proper server configuration, CORS issues should no longer persist. CORS restrictions primarily exist to safeguard users and prevent unauthorized access to resources across different domains.

Building a Proxy for debugging a local web application

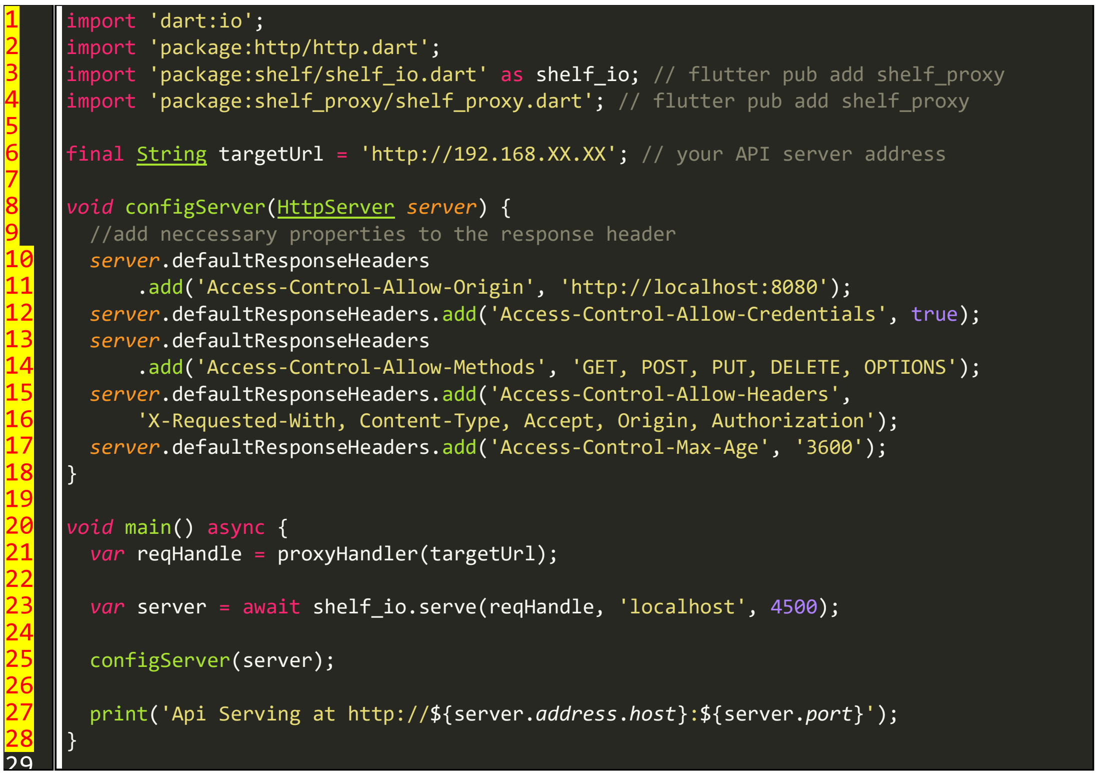
Before developing our UI application, we need to setup a proxy for debugging a web app at localhost domain.

1. Create a "***proxy.dart***" file in the "***eee4482_elibrary/lib/***" directory.as below.



```
lib
  pages
    add_book.dart
    booklist.dart    2
    home_page.dart
  widgets
  api.dart
  main.dart
  proxy.dart
```

2. Add the following codes to the "*proxy.dart*" file.



```
1  import 'dart:io';
2  import 'package:http/http.dart';
3  import 'package:shelf/shelf_io.dart' as shelf_io; // flutter pub add shelf_proxy
4  import 'package:shelf_proxy/shelf_proxy.dart'; // flutter pub add shelf_proxy
5
6  final String targetUrl = 'http://192.168.XX.XX'; // your API server address
7
8  void configServer(HttpServer server) {
9    //add neccessary properties to the response header
10   server.defaultResponseHeaders
11       .add('Access-Control-Allow-Origin', 'http://localhost:8080');
12   server.defaultResponseHeaders.add('Access-Control-Allow-Credentials', true);
13   server.defaultResponseHeaders
14       .add('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, OPTIONS');
15   server.defaultResponseHeaders.add('Access-Control-Allow-Headers',
16       'X-Requested-With, Content-Type, Accept, Origin, Authorization');
17   server.defaultResponseHeaders.add('Access-Control-Max-Age', '3600');
18 }
19
20 void main() async {
21   var reqHandle = proxyHandler(targetUrl);
22
23   var server = await shelf_io.serve(reqHandle, 'localhost', 4500);
24
25   configServer(server);
26
27   print('Api Serving at http://${server.address.host}:${server.port}');
28 }
29
```

In line 6, you need to modify the IP address to become the address of your API server.

3. Install package "*shelf_proxy*" in by input the following command in terminal.

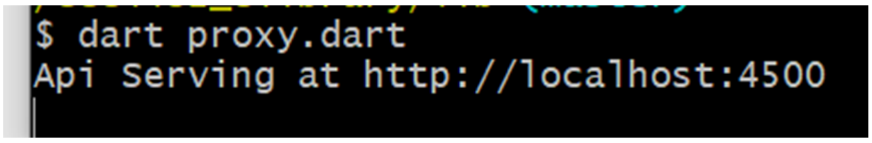
```
flutter pub add shelf_proxy
```

4. Open another terminal (e.g. Windows power shell or GitBash) and cd to the directory of the file "*proxy.dart*" (e.g. "*C:\XXX\YYY\ZZZ\eee4482_elibrary\lib*").

```
cd "C:\XXX\YYY\ZZZ\eee4482_elibrary\lib"
```

The proxy program can be executed with following command.

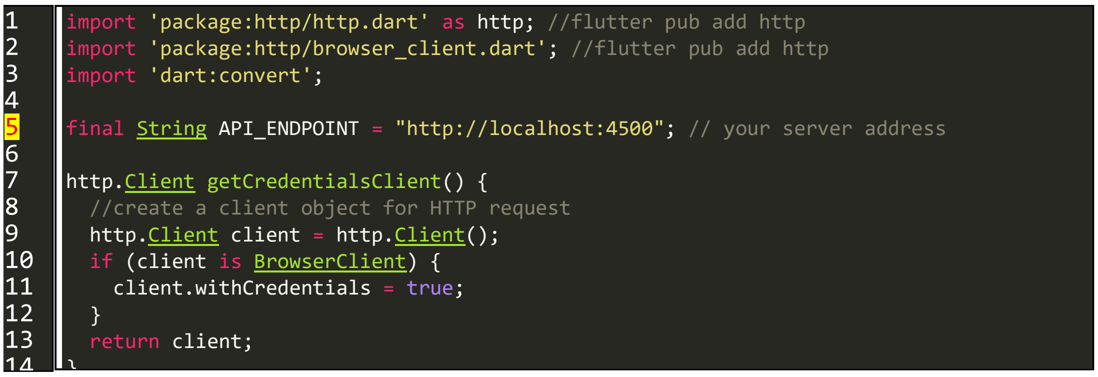
```
dart proxy.dart
```



```
$ dart proxy.dart
Api Serving at http://localhost:4500
```

All of the response from your API server (e.g. 192.168.XX.XX) will be received by the proxy program, and then be transferred to your client application from "*localhost:4500*".

5. Go back to the "***api.dart***" file and modify your api endpoint to "*http://localhost:4500*" as below.



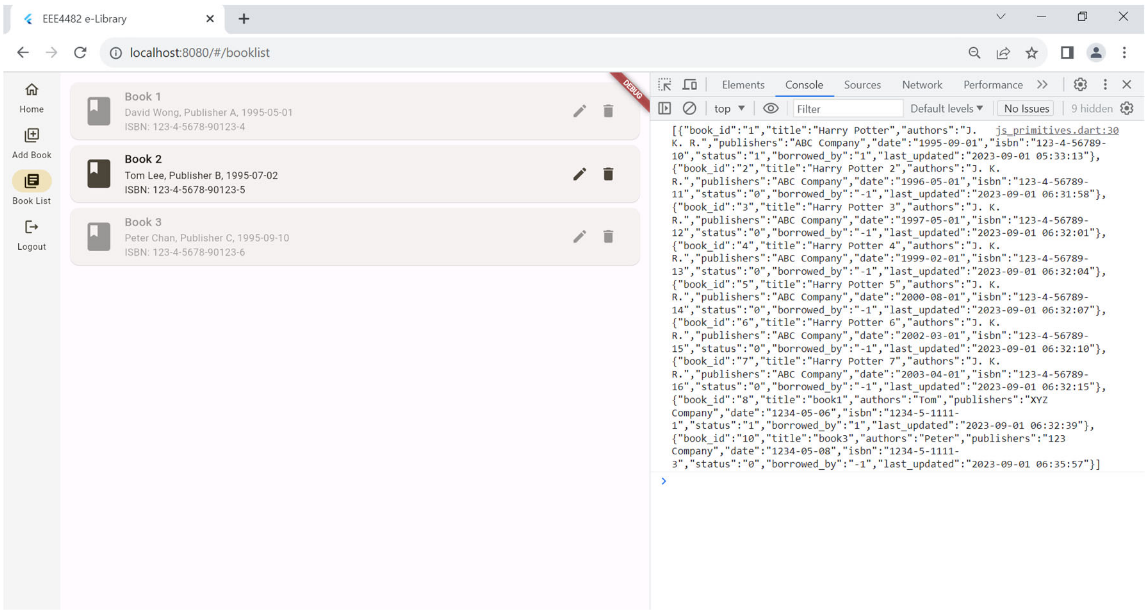
```
import 'package:http/http.dart' as http; //flutter pub add http
import 'package:http/browser_client.dart'; //flutter pub add http
import 'dart:convert';

final String API_ENDPOINT = "http://localhost:4500"; // your server address

http.Client getCredentialsClient() {
  //create a client object for HTTP request
  http.Client client = http.Client();
  if (client is BrowserClient) {
    client.withCredentials = true;
  }
  return client;
}
```

You application will send HTTP request to "localhost:4500" instead of other domain.

6. Run your application with the following command and additional parameters as below.

```
flutter run -d chrome --web-port 8080 --web-hostname localhost
```

These parameters allow you to run you application with a domain of "*localhost:8080*" which satisfy the CORS policy in Chrome.



Book 1
David Wong, Publisher A, 1995-05-01
ISBN: 123-4-5678-90123-4

Book 2
Tom Lee, Publisher B, 1995-07-02
ISBN: 123-4-5678-90123-5

Book 3
Peter Chan, Publisher C, 1995-09-10
ISBN: 123-4-5678-90123-6

```
[{"book_id":"1","title":"Harry Potter","authors":"J.
K. R.","publishers":"ABC Company","date":"1995-09-01","isbn":"123-4-56789-
10","status":"1","borrowed_by":"1","last_updated":"2023-09-01 05:33:13"},
{"book_id":"2","title":"Harry Potter 2","authors":"J. K.
R.","publishers":"ABC Company","date":"1996-05-01","isbn":"123-4-56789-
11","status":"0","borrowed_by":"-1","last_updated":"2023-09-01 06:31:58"},
{"book_id":"3","title":"Harry Potter 3","authors":"J. K.
R.","publishers":"ABC Company","date":"1997-05-01","isbn":"123-4-56789-
12","status":"0","borrowed_by":"-1","last_updated":"2023-09-01 06:32:01"},
{"book_id":"4","title":"Harry Potter 4","authors":"J. K.
R.","publishers":"ABC Company","date":"1999-02-01","isbn":"123-4-56789-
13","status":"0","borrowed_by":"-1","last_updated":"2023-09-01 06:32:04"},
{"book_id":"5","title":"Harry Potter 5","authors":"J. K.
R.","publishers":"ABC Company","date":"2000-08-01","isbn":"123-4-56789-
14","status":"0","borrowed_by":"-1","last_updated":"2023-09-01 06:32:07"},
{"book_id":"6","title":"Harry Potter 6","authors":"J. K.
R.","publishers":"ABC Company","date":"2002-03-01","isbn":"123-4-56789-
15","status":"0","borrowed_by":"-1","last_updated":"2023-09-01 06:32:10"},
{"book_id":"7","title":"Harry Potter 7","authors":"J. K.
R.","publishers":"ABC Company","date":"2003-04-01","isbn":"123-4-56789-
16","status":"0","borrowed_by":"-1","last_updated":"2023-09-01 06:32:15"},
{"book_id":"8","title":"book1","authors":"Tom","publishers":"XYZ
Company","date":"1234-05-06","isbn":"1234-5-1111-
1","status":"1","borrowed_by":"1","last_updated":"2023-09-01 06:32:39"},
{"book_id":"10","title":"book3","authors":"Peter","publishers":"123
Company","date":"1234-05-08","isbn":"1234-5-1111-
3","status":"0","borrowed_by":"-1","last_updated":"2023-09-01 06:35:57"}]
```

Your application successfully retrieves the data from your API server, and it is printed on the console.

What is Async in UI application?

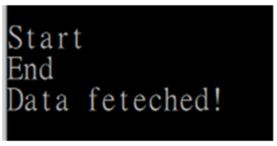
In Dart, the async keyword is used to mark a function as asynchronous. It allows the function to perform tasks concurrently without blocking the program's execution. By using async, a function can utilize the await keyword to pause its execution temporarily until an asynchronous operation completes.

Asynchronous functions in Dart enable non-blocking behavior, allowing other code to run concurrently while waiting for operations such as network requests or file I/O to finish. This approach promotes efficient resource utilization and responsiveness in applications.

Consider the following code example:

```dart
Future<void> fetchData() async {
  // Simulating an asynchronous operation
  await Future.delayed(const Duration(seconds: 2));

  // Perform actions after the delay
  print('Data fetched!');
}

void main() {
  print('Start');
  fetchData();
  print('End');
}
```

In this code snippet, the *fetchData* function is declared as async. It utilizes await *Future.delayed*(...) to pause execution for 2 seconds, simulating an asynchronous task. During this delay, the program doesn't block, and the message 'End' is printed before the asynchronous operation concludes. Once the delay is over, the message 'Data fetched!' is printed.



```
Start
End
Data feteched!
```

By incorporating async and await, Dart provides a more readable and manageable approach to asynchronous programming, particularly when dealing with operations that involve waiting for I/O or time-consuming tasks.

Showing API results in GUI

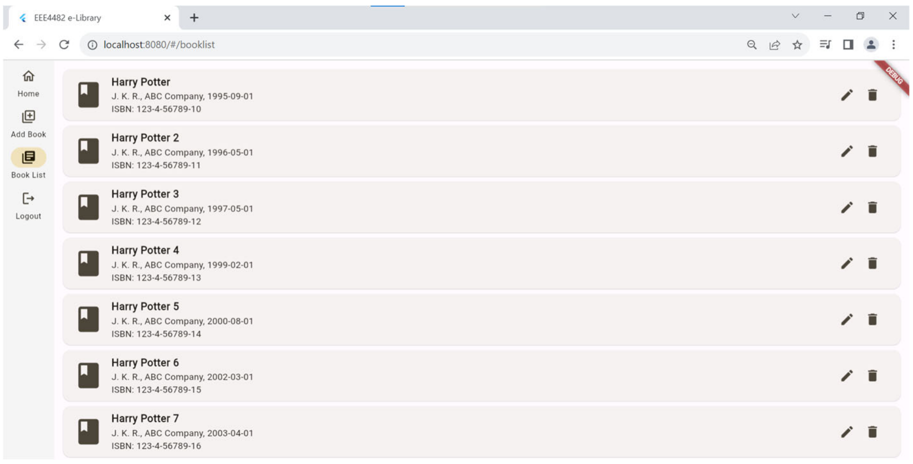
As the API results need time to be received from server, we can't directly process the results in *build* function of a **Widget**. In Flutter, a **FutureBuilder** is required to wait and process the asynchronized data in a widget.

1. Add **FutureBuilder** inside the *build* function of the **State** of **BookListPage**. It has to enclose the widget (e.g. *createBookLists*) where you want to display the API result.

```
40 class _BooklistPageState extends State<BooklistPage> {
41   @override
42   Widget build(BuildContext context) {
43     return NavigationFrame(
44       selectedIndex: 2,
45       child: Padding(
46           padding: EdgeInsets.all(10),
47           child: FutureBuilder<List<dynamic>>(
48             //List<dynamic> is json data
49             future: apiGetAllBooks(), //<<<< try your api function
50             builder: (context, snapshot) {
51               if (snapshot.connectionState == ConnectionState.done) {
52                 if (snapshot.hasError) {
53                   //error
54                   return Text("Error: ${snapshot.error}");
55                 } else {
56                   //success
57                   return createBookLists(snapshot.data!);
58                 }
59               } else {
60                 //loading
61                 return CircularProgressIndicator();
62               }
63             },
64           )),
65     );
66   }
67
```

In line 47 – 64: The **FutureBuilder** is defined with the expected data type of results (e.g. **List<dynamic>),** the asynchronous function (e.g. *apiGetAllbooks*) for getting the data, and the follow up actions when the loading is completed.

Result:



Harry Potter
J. K. R., ABC Company, 1995-09-01
ISBN: 123-4-56789-10

Harry Potter 2
J. K. R., ABC Company, 1996-05-01
ISBN: 123-4-56789-11

Harry Potter 3
J. K. R., ABC Company, 1997-05-01
ISBN: 123-4-56789-12

Harry Potter 4
J. K. R., ABC Company, 1999-02-01
ISBN: 123-4-56789-13

Harry Potter 5
J. K. R., ABC Company, 2000-08-01
ISBN: 123-4-56789-14

Harry Potter 6
J. K. R., ABC Company, 2002-03-01
ISBN: 123-4-56789-15

Harry Potter 7
J. K. R., ABC Company, 2003-04-01
ISBN: 123-4-56789-16

Add records to the database via API function

1. Create the following function at the end of "***api.dart***" file. It is used for sending a new book's record to API server via HTTP POST method.

```dart

Future<bool> apiAddBook(String title, String authors, String publishers,
    String date, String isbn) async {
  var client = getCredentialsClient();
  final response = await client.post(
    Uri.parse(API_ENDPOINT + '/api/books/add'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, String>{
      'title': title,
      'authors': authors,
      'publishers': publishers,
      'date': date,
      'isbn': isbn
    }),
  );
  client.close();

  if (response.statusCode == 200) {
    var jsonResponse = jsonDecode(response.body);
    if (jsonResponse != false) {
      return true;
    } else {
      return false;
    }
  } else {
    return false;
  }
}

```

2. Import "*api.dart*" to the "*book_form.dart*" file as below.

```dart
import 'package:flutter/material.dart';
import '../api.dart';
import '../widgets/input_box.dart';
```

3. Create an Async function named *addBook* inside **_BookFormState** class "*book_form.dart*" file as below.

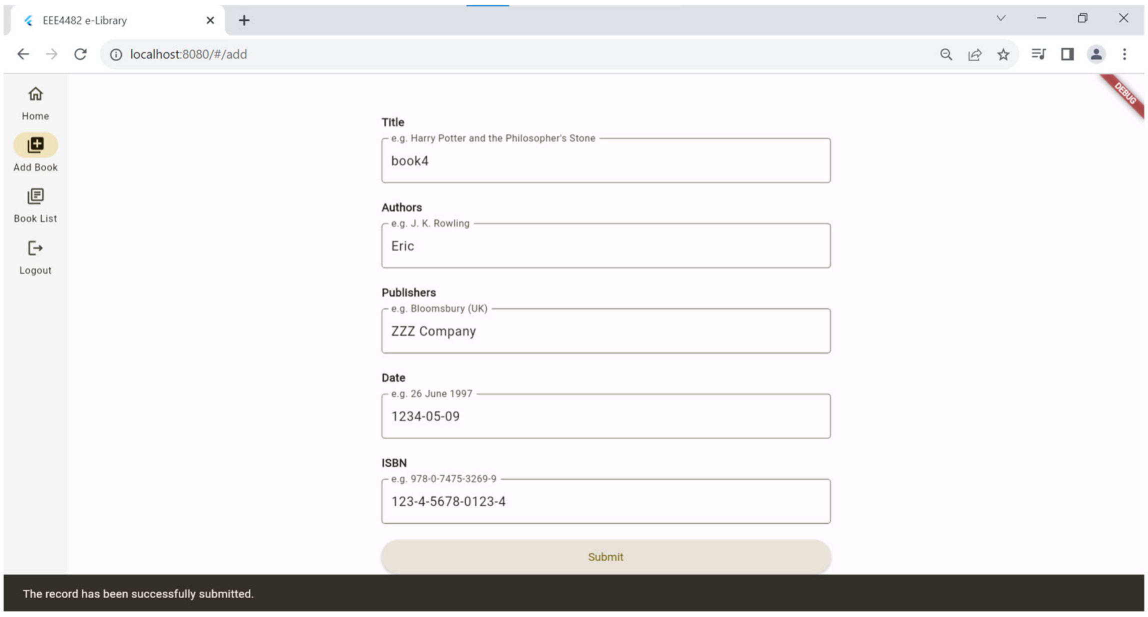
```dart

  Future<void> addBook() async {
    bool result = await apiAddBook(
        _titleController.text,
        _authorsController.text,
        _publishersController.text,
        _dateController.text,
        _isbnController.text);
    String message = 'Failed to submit the book record.';
    if (result == true) {
      message = 'The record has been successfully submitted.';
    }
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(message)),
    );
  }
}
```

4. Update the *_buildWidgets* function by calling *addBook* in the callback function of the form's **ElevatedButton** as below.

```dart
55          Container(
56              margin: EdgeInsets.only(left: 20, top: 10, bottom: 10, right: 20),
57              alignment: Alignment.centerLeft,
58              child: ElevatedButton(
59                style: ElevatedButton.styleFrom(
60                    minimumSize: const Size.fromHeight(50),
61                    textStyle: const TextStyle(fontSize: 14)),
62                onPressed: () {
63                  if (widget.mode == 0) {
64                    addBook();
65                  } else {
66                    ScaffoldMessenger.of(context).showSnackBar(
67                      SnackBar(
68                          content:
69                              Text("Updated " + _titleController.text + ".")),
70                    );
71                  }
72                },
73                child: const Text('Submit'),
74              )),
```

In line 66: the original *showSnackBar* function is replaced by the new *addbook* function.

5. Try the *addbook* function after reload your application.



Title
e.g. Harry Potter and the Philosopher's Stone
book4

Authors
e.g. J. K. Rowling
Eric

Publishers
e.g. Bloomsbury (UK)
ZZZ Company

Date
e.g. 26 June 1997
1234-05-09

ISBN
e.g. 978-0-7475-3269-9
123-4-5678-0123-4

The record has been successfully submitted.

Update and Delete records in database via API functions

1. Create the following functions at the end of "***api.dart***" file. The functions are used for updating or deleting a book's record in database via HTTP PUT method and HTTP DELETE method respectively.

```

Future<bool> apiUpdateBook(int bookId, String title, String authors,
    String publishers, String date, String isbn) async {
  var client = getCredentialsClient();
  final response = await client.put(
    Uri.parse(API_ENDPOINT + '/api/books/update/' + bookId.toString()),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, String>{
      'title': title,
      'authors': authors,
      'publishers': publishers,
      'date': date,
      'isbn': isbn
    }),
  );
  client.close();
  if (response.statusCode == 200) {
    var jsonResponse = jsonDecode(response.body);
    if (jsonResponse != false) {
      return true;
    } else {
      return false;
    }
  } else {
    return false;
  }
}

Future<bool> apiDeleteBook(int bookId) async {
  var client = getCredentialsClient();
  final response = await client.delete(
      Uri.parse(API_ENDPOINT + '/api/books/delete/' + bookId.toString()));
  client.close();
  if (response.statusCode == 200) {
    var jsonResponse = jsonDecode(response.body);
    if (jsonResponse != false) {
      return true;
    } else {
      return false;
    }
  } else {
    return false;
  }
}

```

2. Create an Async function named *updateBook* inside **_BookFormState** class "*book_form.dart*" file as below.

```
  Future<void> updateBook() async {
    bool result = await apiUpdateBook(
        widget.bookId,
        _titleController.text,
        _authorsController.text,
        _publishersController.text,
        _dateController.text,
        _isbnController.text);
    String message = 'Failed to update the book record.';
    if (result == true) {
      message = 'The record has been successfully updated.';
    }
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(message)),
    );
  }
}
```

3. Update the *_buildWidgets* function to call *updateBook* in the *onPressed* callback function of the form's **ElevatedButton** as below.

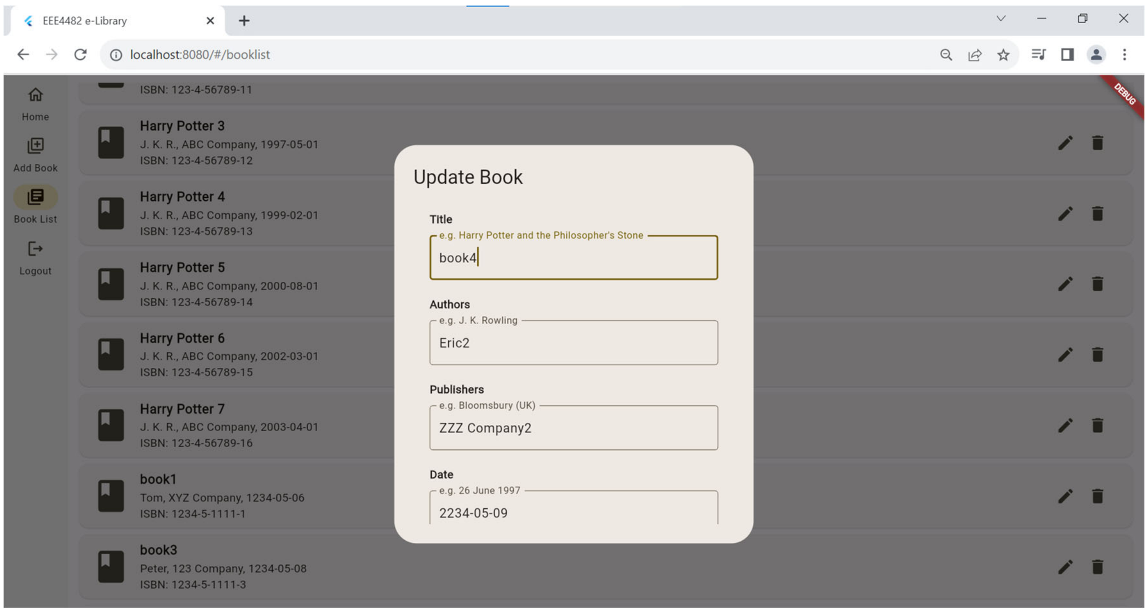
```
55          Container(
56              margin: EdgeInsets.only(left: 20, top: 10, bottom: 10, right: 20),
57              alignment: Alignment.centerLeft,
58              child: ElevatedButton(
59                style: ElevatedButton.styleFrom(
60                    minimumSize: const Size.fromHeight(50),
61                    textStyle: const TextStyle(fontSize: 14)),
62                onPressed: () {
63                  if (widget.mode == 0) {
64                    addBook();
65                  } else {
66                    updateBook();
67                  }
68                },
69                child: const Text('Submit'),
70              )),
```

In line 66: the original *showSnackBar* function is replaced by the new *updatebook* function.

4. Update *popupUpdateDialog* function in "*booklist_page.dart*" to pass the *book_id* from **BooklistPage** to the **BookForm** for the use of *apiUpdateBook* function.

```
  }

  void popupUpdateDialog(Map<String, dynamic> book) {
    showDialog<String>(
        context: context,
        builder: (BuildContext context) => AlertDialog(
              title: const Text('Update Book'),
              content: Container(
                  width: 400,
                  height: 400,
                  child: SingleChildScrollView(
                      child: BookForm(
                          mode: 1, bookId: int.parse(book['book_id'])))),
            ));
  }

  void popupDeletewDialog(Map<String, dynamic> book) {
```

Try the update function yourself:



ISBN: 123-4-56789-11

Harry Potter 3
J. K. R., ABC Company, 1997-05-01
ISBN: 123-4-56789-12

Harry Potter 4
J. K. R., ABC Company, 1999-02-01
ISBN: 123-4-56789-13

Harry Potter 5
J. K. R., ABC Company, 2000-08-01
ISBN: 123-4-56789-14

Harry Potter 6
J. K. R., ABC Company, 2002-03-01
ISBN: 123-4-56789-15

Harry Potter 7
J. K. R., ABC Company, 2003-04-01
ISBN: 123-4-56789-16

book1
Tom, XYZ Company, 1234-05-06
ISBN: 1234-5-1111-1

book3
Peter, 123 Company, 1234-05-08
ISBN: 1234-5-1111-3

Update Book

Title
e.g. Harry Potter and the Philosopher's Stone
book4

Authors
e.g. J. K. Rowling
Eric2

Publishers
e.g. Bloomsbury (UK)
ZZZ Company2

Date
e.g. 26 June 1997
2234-05-09

5. Create an Async function named *deleteBook* inside **_BooklistPageState** class "*booklist_page.dart*" file as below.

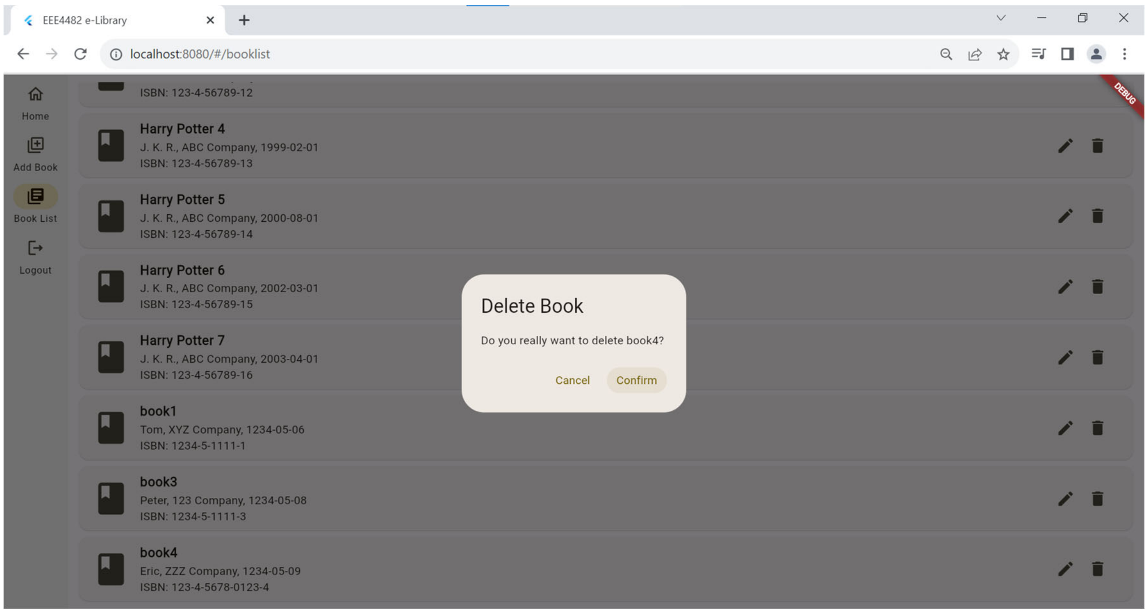
```

  Future<void> deleteBook(int bookId) async {
    String message = "Failed to delete the book.";
    if (await apiDeleteBook(bookId) == true) {
      message = "The book has been deleted successfully.";
      Navigator.of(context).pushNamed("/booklist");
    }
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(message)),
    );
  }
}
```

6. Update *popupDeletewDialog* function in "*booklist_page.dart*" to call *deleteBook* in the *onPressed* callback function of the confirm button.

```
166
167   void popupDeletewDialog(Map<String, dynamic> book) {
168     showDialog<String>(
169         context: context,
170         builder: (BuildContext context) => AlertDialog(
171               title: const Text('Delete Book'),
172               content:
173                   Text('Do you really want to delete ' + book["title"] + '?'),
174               actions: <Widget>[
175                 TextButton(
176                   onPressed: () => Navigator.pop(context, 'Cancel'),
177                   child: const Text('Cancel'),
178                 ),
179                 TextButton(
180                   onPressed: () {
181                     deleteBook(int.parse(book["book_id"]));
182                     Navigator.pop(context, 'Confirm');
183                   },
184                   child: const Text('Confirm'),
185                 ),
186               ],
187             ));
188   }
189
```

In line 184: The json data of *book_id* is parsed as integer and is passed to the *deletebook* function for the use of *apiDeleteBook* function.

Try the delete function yourself:



ISBN: 123-4-56789-12

Harry Potter 4
J. K. R., ABC Company, 1999-02-01
ISBN: 123-4-56789-13

Harry Potter 5
J. K. R., ABC Company, 2000-08-01
ISBN: 123-4-56789-14

Harry Potter 6
J. K. R., ABC Company, 2002-03-01
ISBN: 123-4-56789-15

Harry Potter 7
J. K. R., ABC Company, 2003-04-01
ISBN: 123-4-56789-16

book1
Tom, XYZ Company, 1234-05-06
ISBN: 1234-5-1111-1

book3
Peter, 123 Company, 1234-05-08
ISBN: 1234-5-1111-3

book4
Eric, ZZZ Company, 1234-05-09
ISBN: 123-4-5678-0123-4

Delete Book
Do you really want to delete book4?
Cancel Confirm

Validation of user inputs in Flutter

Input validation is the process of evaluating and ensuring that data or information entered by a user or received from an external source meets certain criteria or constraints. It is an essential step in developing secure and reliable software applications.

The main purpose of input validation is to prevent malicious or erroneous data from compromising the integrity, availability, and security of the system. By validating input, developers can verify that the data conforms to the expected format, type, length, and range before processing or storing it.

Input validation can involve several techniques and methods, including:

- Data type validation: Checking if the input matches the expected data type, such as strings, numbers, dates, or email addresses.
- Length validation: Verifying that the input falls within the allowed length limits to prevent buffer overflows or truncation issues.
- Format validation: Ensuring that the input adheres to a specified format, such as phone numbers, social security numbers, or credit card numbers.
- Range validation: Validating that numeric inputs are within the acceptable range or limits defined by the application.
- Whitelist/Blacklist validation: Comparing input against a predefined list of allowed (whitelist) or disallowed (blacklist) values to filter out potentially harmful or unauthorized data.
- Cross-site scripting (XSS) prevention: Escaping or sanitizing user input to prevent script injection attacks.
- SQL injection prevention: Sanitizing or parameterizing user input to prevent malicious SQL queries from being executed.
- Regular expression matching: Using regular expressions to validate input against complex patterns or rules.
- Business rule validation: Checking if the input satisfies specific business logic or rules relevant to the application's domain.

It's important to note that input validation should be performed on both the client-side (e.g., in web forms) and the server-side (e.g., in backend processing) to ensure comprehensive protection. By implementing robust input validation mechanisms, developers can enhance the security and reliability of their applications.

1. Create a simple validation function below the *build* function in "*input_box.dart*".

```
1  import 'package:flutter/material.dart';
2  
3  class InputBox extends StatefulWidget {
4    final String name;
5    final String hint;
6    final TextEditingController controller;
7    final bool obscureText;
8    InputBox(
9        {super.key,
10       required this.name,
11       required this.hint,
12       required this.controller,
13       this.obscureText = false});
14   @override
15   State<InputBox> createState() => _InputBoxState();
16 }
17 
18 class _InputBoxState extends State<InputBox> {
19   @override
20   Widget build(BuildContext context) {
21     return Column(
22       crossAxisAlignment: CrossAxisAlignment.start,
23       children: [
24         Container(
25           margin: EdgeInsets.only(left: 20, top: 10, bottom: 10, right: 20),
26           child: Text(
27             widget.name,
28             style: TextStyle(fontSize: 14, fontWeight: FontWeight.bold),
29           ),
30         ),
31         Container(
32             margin: EdgeInsets.only(left: 20, top: 0, bottom: 10, right: 20),
33             alignment: Alignment.centerLeft,
34             child: TextFormField(
35               obscureText: widget.obscureText,
36               controller: widget.controller,
37               decoration: InputDecoration(
38                 border: OutlineInputBorder(),
39                 labelText: widget.hint,
40               ),
41               validator: _validateInput,
42             )),
43       ],
44     );
45   }
46 
47   String? _validateInput(String? value) {
48     if (value == null || value.isEmpty) return 'Input cannot be empty.';
49     return null;
50   }
51 }
52 
53 
```

In line 47 – 51: A simple validation function is created to check whether the **TextFormField** is empty.

In line 41: the validation function is added into the **TextFormField**.

2. Update the **BookForm** widget in "*book_form.dart*" to perform the validation if the submission button is pressed.

```
1  import 'package:flutter/material.dart';
2  import '../api.dart';
3  import '../widgets/input_box.dart';
4
5  class BookForm extends StatefulWidget {
6    final int mode;
7    final int bookId;
8    const BookForm({super.key, required this.mode, this.bookId = 0});
9    @override
10   State<BookForm> createState() => _BookFormState();
11 }
12
13 class _BookFormState extends State<BookForm> {
14   final _formKey = GlobalKey<FormState>();
15   final TextEditingController _titleController = TextEditingController();
16   final TextEditingController _authorsController = TextEditingController();
17   final TextEditingController _publishersController = TextEditingController();
18   final TextEditingController _dateController = TextEditingController();
19   final TextEditingController _isbnController = TextEditingController();
20   @override
21   Widget build(BuildContext context) {
22     return Form(key: _formKey, child: _buildWidgets(context));
23   }
```

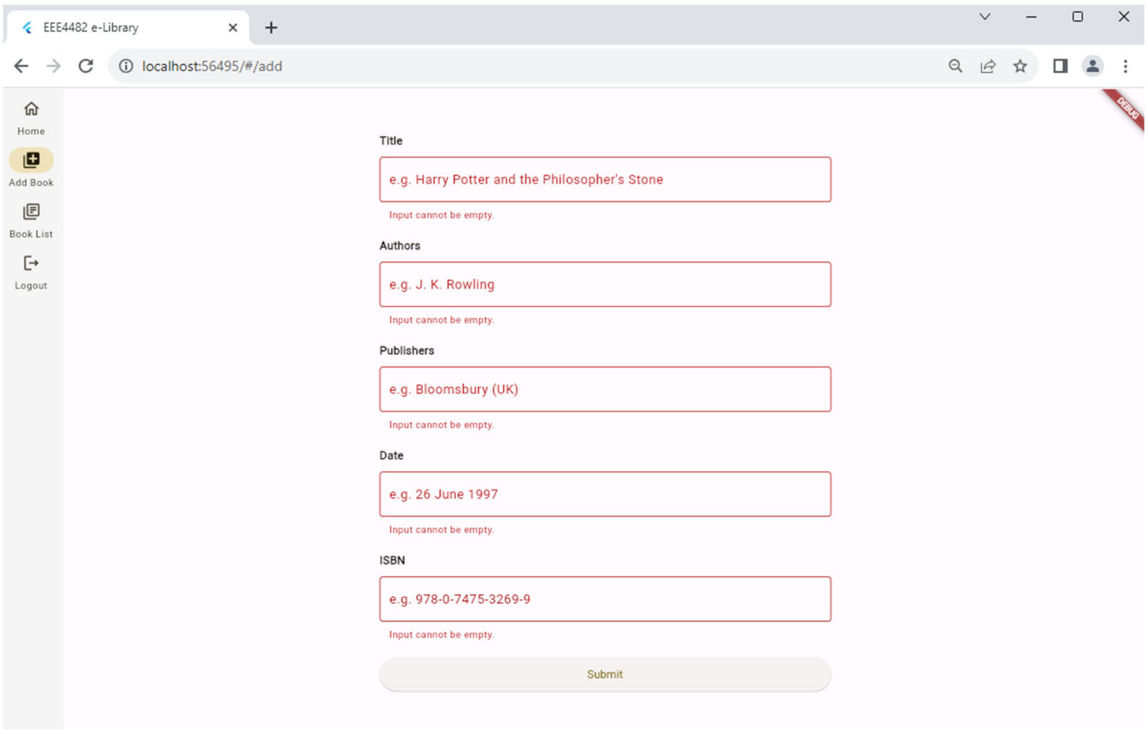
…

```
55           Container(
56               margin: EdgeInsets.only(left: 20, top: 10, bottom: 10, right: 20),
57               alignment: Alignment.centerLeft,
58               child: ElevatedButton(
59                 style: ElevatedButton.styleFrom(
60                     minimumSize: const Size.fromHeight(50),
61                     textStyle: const TextStyle(fontSize: 14)),
62                 onPressed: () {
63                   if (_formKey.currentState!.validate()) {
64                     if (widget.mode == 0) {
65                       addBook();
66                     } else {
67                       updateBook();
68                     }
69                   }
70                 },
71                 child: const Text('Submit'),
72               )),
```

In line 14: Please ensure a **GlobalKey** of **FormState** is already declared in the **State** class.

In line 63: The **GlobalKey** *_formKey* performs the validation and return a Boolean result for checking.

3. Refresh the program and try the validation function.



Title
e.g. Harry Potter and the Philosopher's Stone
Input cannot be empty.

Authors
e.g. J. K. Rowling
Input cannot be empty.

Publishers
e.g. Bloomsbury (UK)
Input cannot be empty.

Date
e.g. 26 June 1997
Input cannot be empty.

ISBN
e.g. 978-0-7475-3269-9
Input cannot be empty.

If you leave the input boxes blank, a warning message will be display below the **TextFormField** widgets as shown.

4. In practical situations, the validation methods of the input boxes should be different. The validator of **TextFormField** widget can be set as a parameter of **InputBox** and to be declared in **BookForm**.

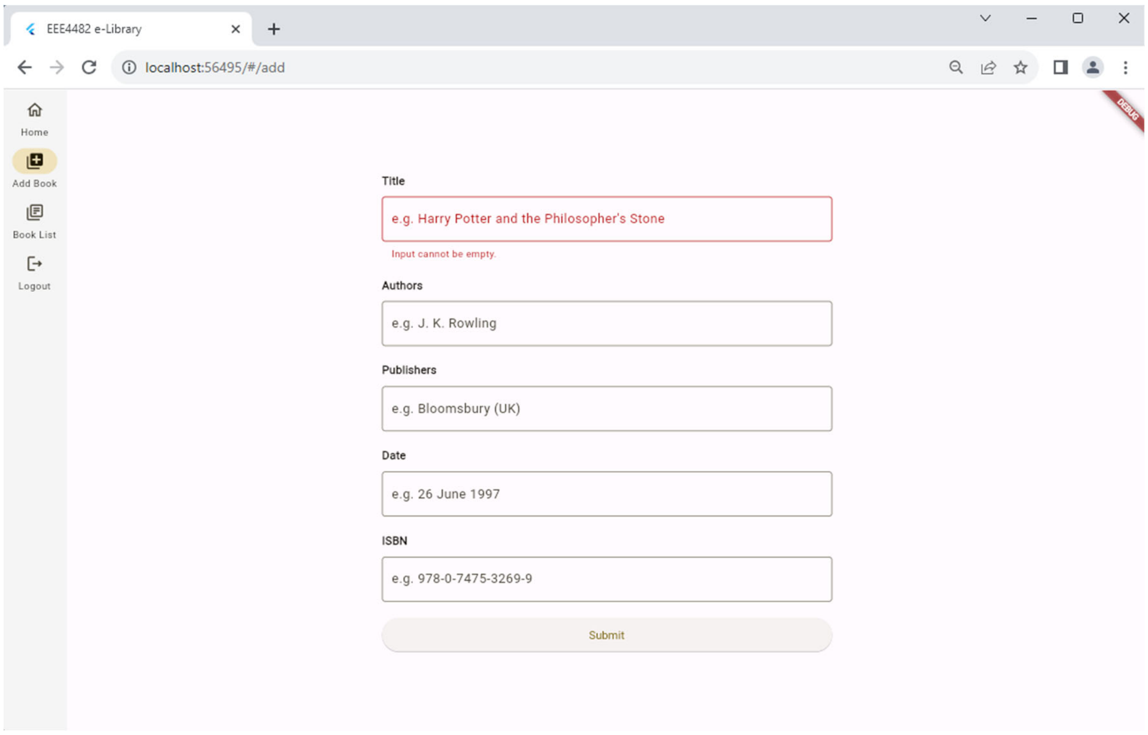
```
import 'package:flutter/material.dart';

class InputBox extends StatefulWidget {
  final String name;
  final String hint;
  final TextEditingController controller;
  final String? Function(String?)? validator;
  final bool obscureText;
  InputBox(
      {super.key,
      required this.name,
      required this.hint,
      required this.controller,
      this.validator,
      this.obscureText = false});
  @override
  State<InputBox> createState() => _InputBoxState();
}

class _InputBoxState extends State<InputBox> {
  @override
  Widget build(BuildContext context) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Container(
          margin: EdgeInsets.only(left: 20, top: 10, bottom: 10, right: 20),
          child: Text(
            widget.name,
            style: TextStyle(fontSize: 14, fontWeight: FontWeight.bold),
          ),
        ),
        Container(
            margin: EdgeInsets.only(left: 20, top: 0, bottom: 10, right: 20),
            alignment: Alignment.centerLeft,
            child: TextFormField(
              obscureText: widget.obscureText,
              controller: widget.controller,
              decoration: InputDecoration(
                border: OutlineInputBorder(),
                labelText: widget.hint,
              ),
              validator: widget.validator,
            )),
      ],
    );
  }
}
```

5. Add the validator method in **_BookFormState** class and put it as a parameters of the **InputBox** widgets.

```
12
13  class _BookFormState extends State<BookForm> {
14    final _formKey = GlobalKey<FormState>();
15    final TextEditingController _titleController = TextEditingController();
16    final TextEditingController _authorsController = TextEditingController();
17    final TextEditingController _publishersController = TextEditingController();
18    final TextEditingController _dateController = TextEditingController();
19    final TextEditingController _isbnController = TextEditingController();
20
21    String? _validateInput(String? value) {
22      if (value == null || value.isEmpty) return 'Input cannot be empty.';
23      return null;
24    }
25
26    @override
27    Widget build(BuildContext context) {
28      return Form(key: _formKey, child: _buildWidgets(context));
29    }
30
31    Widget _buildWidgets(BuildContext context) {
32      return Column(
33          mainAxisAlignment: MainAxisAlignment.center,
34          crossAxisAlignment: CrossAxisAlignment.start,
35          children: [
36            InputBox(
37              name: "Title",
38              hint: "e.g. Harry Potter and the Philosopher's Stone",
39              controller: _titleController,
40              validator: _validateInput,
41            ),
42            InputBox(
43              name: "Authors",
44              hint: "e.g. J. K. Rowling",
45              controller: _authorsController,
46            ),
47            InputBox(
48              name: "Publishers",
49              hint: "e.g. Bloomsbury (UK)",
50              controller: _publishersController,
51            ),
```

In line 21 – 24: The _*validateInput* function is declared for the validation usage of **InputBox** widgets in **BookForm** widget.
In line 40: the _*validateInput* is input to the **InputBox** for validation.

6. If you refresh the program and try to submit the blank form, only the title's **InputBox** will display a warning message.



7. For other validation functions, a dart package named "**validators**" can be installed for handling various input types.
   - Press "ctrl + C" to exit the programme in terminal.
   - Input "flutter pub add validators" to install the "**validators**" package.

```
flutter pub add validators
```

```
PS C:\Users\Ivan\Documents\Flutter WS\4482_lab7c\eee4482_elibrary> flutter pub add validators
Resolving dependencies...
  material_color_utilities 0.5.0 (0.8.0 available)
+ validators 3.0.0
Changed 1 dependency!
```

8. Import the package "validator" and add more validation functions in **_BookFormState** class as below.

```
import 'package:flutter/material.dart';
import '../api.dart';
import '../widgets/input_box.dart';
import 'package:validators/validators.dart'; //flutter pub add validators

class BookForm extends StatefulWidget {
```

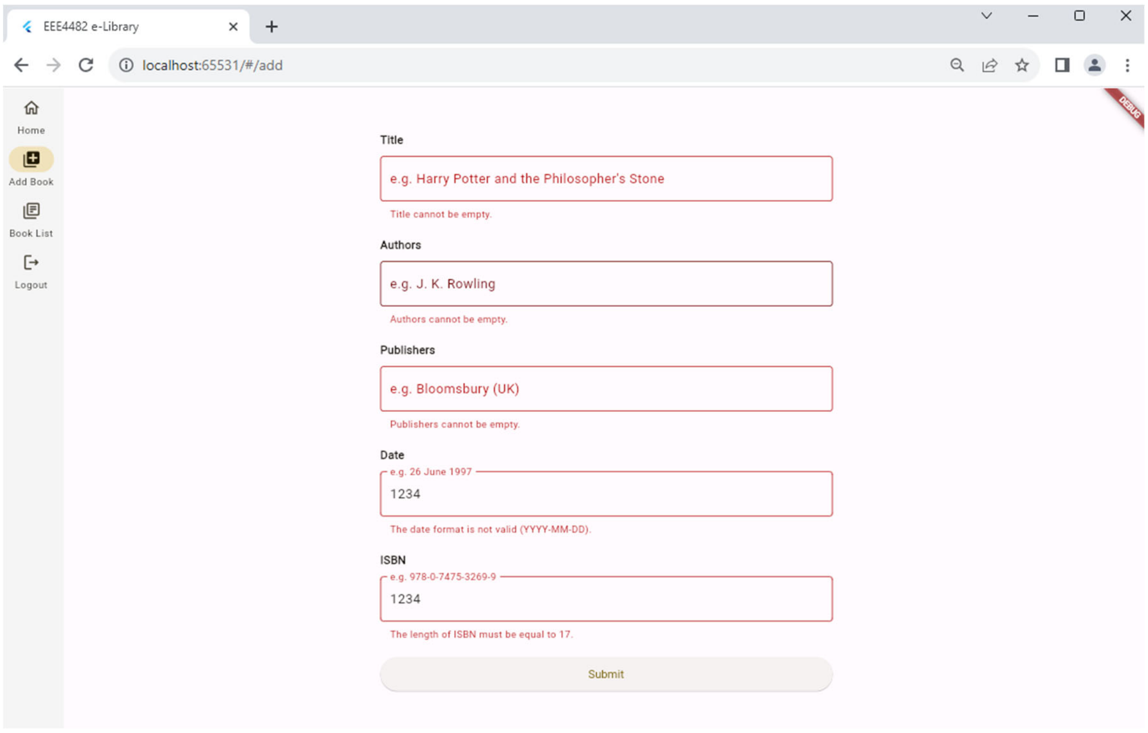
…

```
21
22   String? _validateTitle(String? value) {
23     if (value == null || value.isEmpty) return 'Title cannot be empty.';
24     return null;
25   }
26
27   String? _validateAuthors(String? value) {
28     if (value == null || value.isEmpty) return 'Authors cannot be empty.';
29     return null;
30   }
31
32   String? _validatePublishers(String? value) {
33     if (value == null || value.isEmpty) return 'Publishers cannot be empty.';
34     return null;
35   }
36
37   String? _validateDate(String? value) {
38     if (value == null || value.isEmpty) return 'Date cannot be empty.';
39     if (!isDate(value)) return 'The date format is not valid (YYYY-MM-DD).';
40     return null;
41   }
42
43   String? _validateISBN(String? value) {
44     if (value == null || value.isEmpty) return 'ISBN cannot be empty.';
45     if (value.length != 17) return 'The length of ISBN must be equal to 17.';
46     return null;
47   }
48
49   @override
50   Widget build(BuildContext context) {
51     return Form(key: _formKey, child: _buildWidgets(context));
52   }
53
54   Widget _buildWidgets(BuildContext context) {
55     return Column(
56         mainAxisAlignment: MainAxisAlignment.center,
57         crossAxisAlignment: CrossAxisAlignment.start,
58         children: [
59           InputBox(
60             name: "Title",
61             hint: "e.g. Harry Potter and the Philosopher's Stone",
62             controller: _titleController,
63             validator: _validateTitle,
64           ),
65           InputBox(
66             name: "Authors",
67             hint: "e.g. J. K. Rowling",
68             controller: _authorsController,
69             validator: _validateAuthors,
70           ),
71           InputBox(
72             name: "Publishers",
73             hint: "e.g. Bloomsbury (UK)",
74             controller: _publishersController,
75             validator: _validatePublishers,
76           ),
77           InputBox(
78             name: "Date",
79             hint: "e.g. 26 June 1997",
80             controller: _dateController,
81             validator: _validateDate,
82           ),
83           InputBox(
84             name: "ISBN",
85             hint: "e.g. 978-0-7475-3269-9",
86             controller: _isbnController,
87             validator: _validateISBN,
88           ),
89           Container(
90               margin: EdgeInsets.only(left: 20, top: 10, bottom: 10, right: 20),
91               alignment: Alignment.centerLeft,
```

In line 39: The *isDate* function check if the input string is following the date format in YYYY-MM-DD.

In line 45: It checks whether the length of input string is equal to 17.

9. Refresh the program and try your validators.



Title

e.g. Harry Potter and the Philosopher's Stone

Title cannot be empty.

Authors

e.g. J. K. Rowling

Authors cannot be empty.

Publishers

e.g. Bloomsbury (UK)

Publishers cannot be empty.

Date

e.g. 26 June 1997

1234

The date format is not valid (YYYY-MM-DD).

ISBN

e.g. 978-0-7475-3269-9

1234

The length of ISBN must be equal to 17.

Submit

References

https://docs.flutter.dev/development/ui/widgets/material
https://m3.material.io/
https://pub.dev/packages/validators/example

END