**Birla Institute of Technology & Science, Pilani**

**Work Integrated Learning Programmes Division**

**First Semester 2022-2023**

**Comprehensive Examination**

**(EC-3 Regular )**

Course No. : SE ZG585

Course Title : Cross-Platform Application Development

Nature of Exam : Open Book

Weightage : 40%

No. of Pages = 7

# No. of Questions = 4

Duration : 2.5 Hours

Date of Exam : 25/11/2022 (FN)

Note to Students:

1. Please follow all the *Instructions to Candidates* given on the cover page of the answer book.
2. All parts of a question should be answered consecutively. Each answer should start from a fresh page.
3. Assumptions made if any, should be stated clearly at the beginning of your answer.
4. You are consulted to design and develop an online pharmacy aggregator mobile application which enables the users to place an order for the medicines which will be fulfilled by the nearby medical stores. You are evaluating the technology options available for the quick development of backend of this application. **[1 + 1 + 6 = 8]**
5. List two options discussed in the classroom suitable for this requirement.
6. Which of the option mentioned in part (a), you will prefer for this application? Why?
7. Justify/Invalidate the need of the following out-of-box features supported by option selected in part (b).
8. Database
9. Storage
10. GraphQL APIs
11. Notifications
12. Authentication
13. Custom functions

Ans:

a)

Options

* Ruby (Ruby on Rails)
* Python (Django, Flask, Pylons)
* PHP (Laravel)
* Java (Spring)
* Scala (Play)
* Javascript (Node.js)
* Serverless-AWS Cloud lambda functions
* Serverless-Azure Cloud functions, Google Cloud functions

b)

* Javascript (Node.js)

c)

a) **Two options that could be suitable for the backend development of the online pharmacy aggregator mobile application are:**

1. Node.js with Express.js: Node.js is a JavaScript runtime built on Chrome's V8 engine, and Express.js is a popular web application framework for Node.js. Node.js enables asynchronous and event-driven programming, making it efficient for handling concurrent requests. Express.js provides a robust set of features for building APIs and handling HTTP requests, making it suitable for creating a backend server for the application.
2. Django: Django is a high-level web framework written in Python. It follows the Model-View-Controller (MVC) architectural pattern and provides a rich set of features for rapid web development. Django offers built-in authentication, database management, and security features, making it suitable for building the backend of the online pharmacy aggregator application.

**(b) I would prefer Node.js with Express.js for this application.**

The choice of Node.js with Express.js is primarily based on the following factors:

1. JavaScript ecosystem: Since the online pharmacy aggregator application will also have a frontend developed using JavaScript or a JavaScript framework, using Node.js for the backend allows for code sharing and reuse between the frontend and backend. It simplifies the development process and reduces the learning curve for developers.
2. Asynchronous and scalable: Node.js, with its asynchronous nature, is well-suited for handling a large number of concurrent requests, which is crucial for a real-time application like an online pharmacy aggregator. It can efficiently handle multiple requests simultaneously, ensuring a smooth user experience.
3. Rich ecosystem and community support: Node.js has a vast ecosystem of libraries and modules available, which can help accelerate development and provide solutions for various requirements. The active community surrounding Node.js ensures continuous support, bug fixes, and enhancements, making it a reliable choice for long-term application development.

**(c) Justification for the need of out-of-box features supported by Node.js with Express.js:**

1. Asynchronous handling: The online pharmacy aggregator application may need to interact with multiple APIs, databases, and external services. Node.js with its asynchronous nature allows for efficient handling of these interactions, ensuring fast response times and scalability.
2. REST API development: Express.js provides a clean and intuitive way to build RESTful APIs, allowing the mobile application to communicate with the backend seamlessly. It offers features like routing, request handling, middleware support, and error handling, which are crucial for building a robust API layer.
3. Real-time updates: If the application requires real-time updates, such as notifying users about order status changes or new offers, Node.js with libraries like Socket.IO can facilitate bidirectional communication between the server and the client in real-time.
4. Integration with external services: Node.js has a wide range of libraries and modules available for integrating with various services and APIs, such as payment gateways, SMS gateways, and geolocation services. This flexibility allows for seamless integration with third-party services required for the online pharmacy aggregator application.

In conclusion, Node.js with Express.js provides the necessary features, scalability, and ecosystem support required for building the backend of an online pharmacy aggregator mobile application efficiently.

1. You are made a part of mobile application development team which is discussing the development approach to be used for this application. Two of the prominent choices being discussed are – native application and hybrid application. Provide your opinion about these two approaches based on the following points with brief justification. **[6]**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Native Application** | **Hybrid Application** |
| **App Performance** | High | Slow |
| **Readymade Libraries** | Lot of lib is available, including open source | Dependent of vendors and open source |
| **App Release time** | Slow, takes time | Faster |
| **Development cost** | High | Low |
| **Customizations** | Easy to customize | Difficult compare to native app |
| **Talent Pool availability (availability of resource)** | Less resource for expert skill | More |

Ans

Ans is filled in tables

**Chatgpt ans**

Opinion on Native Application vs. Hybrid Application for the online pharmacy aggregator mobile application:

App Performance: Native Application: Native applications are developed specifically for a particular platform (e.g., iOS or Android) using platform-specific programming languages (Swift/Objective-C for iOS, Java/Kotlin for Android). As a result, native apps have direct access to device hardware and APIs, offering superior performance and responsiveness.

Hybrid Application: Hybrid applications are built using web technologies (HTML, CSS, JavaScript) and wrapped within a native container. They rely on a WebView to render the user interface, which can introduce some performance overhead compared to native apps.

Justification: If app performance is a critical factor for the online pharmacy aggregator, native application development would be preferable. Direct access to platform-specific features and optimized code execution can result in a faster and more seamless user experience.

Readymade Libraries: Native Application: Native development platforms have extensive libraries and frameworks provided by the platform vendors, as well as a wide range of third-party libraries available. These libraries are often mature, well-documented, and optimized for the respective platforms.

Hybrid Application: Hybrid frameworks, such as React Native or Flutter, have their own libraries and ecosystem. Although the options might not be as extensive as in the native development world, they still offer a good selection of libraries for common functionalities.

Justification: Both native and hybrid approaches offer a decent selection of readymade libraries. Native platforms may have a slight advantage in terms of library availability and maturity, but hybrid frameworks have been growing rapidly and offer a solid ecosystem of their own.

App Release time: Native Application: Developing separate native applications for each platform (iOS and Android) can result in longer development time. Each platform requires its own codebase and development processes, which can extend the overall release time.

Hybrid Application: Hybrid applications can be developed once and deployed across multiple platforms. This approach significantly reduces the time required for development and release, as a single codebase can target both iOS and Android.

Justification: If a quick app release is a priority, the hybrid application approach is preferable. Building a single codebase and deploying it across multiple platforms saves development time and allows for simultaneous releases on iOS and Android.

Development cost: Native Application: Developing native applications requires separate codebases for iOS and Android, which increases development and maintenance costs. Each platform requires platform-specific expertise and resources.

Hybrid Application: Hybrid applications, with their single codebase, can reduce development and maintenance costs. Development teams can be smaller, and there is no need for separate platform-specific expertise.

Justification: If cost-effectiveness is a significant concern, the hybrid application approach is advantageous. By reducing development and maintenance efforts, hybrid development can result in lower overall costs.

Customizations: Native Application: Native applications allow for fine-grained customizations tailored to the platform-specific guidelines and user experience expectations. This level of customization provides a native look and feel, which can enhance user satisfaction.

Hybrid Application: Hybrid applications aim for a unified user interface across platforms, which may limit the degree of platform-specific customizations. However, modern hybrid frameworks have improved in this aspect, allowing for more extensive customizations and platform-specific adaptations.

Justification: If the online pharmacy aggregator requires extensive platform-specific customizations, the native application approach is recommended. Native apps offer greater flexibility to adhere to specific platform guidelines and optimize the user experience accordingly.

Talent Pool availability: Native Application: Native development requires platform-specific expertise, and there is usually a well-established talent pool available for both iOS and Android development. Finding experienced native developers should not be a major challenge.

Hybrid Application: Hybrid development using frameworks like React Native or Flutter requires knowledge of web technologies and the specific hybrid framework. The talent pool for hybrid

1. Consider the following table structure and data maintained in the database of an online library application. The book table contains details about the books available in the library. The reservation table contains the information about the allotted books to the users. Users table maintains data about customers of the library. You have been consulted to make this data available to the web / mobile applications.

Books

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Name | Author | Tag | Year |
| 1 | JS Defined | PQR | JS | 2015 |
| 2 | Java Distilled | ABC | Java | 2012 |
| 3 | Nodejs In Practice | WERT | JS, NodeJS | 2012 |
| 4 | Decipher UML | FGH | UML, Java | 2016 |

Reservation

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Book\_Id | User\_id | Date |
| 1 | 1 | 1 | 14 Oct |
| 2 | 2 | 1 | 14 Oct |
| 3 | 3 | 2 | 15 Oct |
| 4 | 1 | 3 | 13 Nov |

Users

|  |  |
| --- | --- |
| Id | Name |
| 1 | ABC |
| 3 | XYZ |
| 2 | EFG |
| 4 | PQR |

Using the given table structure and data, attempt the following questions:

[**1 + 1 + 2 + 2 + 4 + 1 = 11]**

1. If the client application wants to have more control over the response received from the server side, which API paradigm should be used for API development? Why?

Based on your answer for part (a),

1. How many API endpoints will be needed in this case?
2. Design an API query that will return all the details of the specified book. Show a sample response.
3. Write down an API query that will generate the response with following structure. [{

Book\_name,

User\_name,

Date\_of\_Reservation

}]

1. To send this response in part (d), present the supporting object’s structure required on sever side.
2. How you will support insert / update / delete operations on books?

**Ans:**

1. Restful service (or grapql)
2. As it is mentioned, to get only data in web/mobile application, we will go with **one endpoint** with get call

Method

Get

URL

https://<hostname>/api/v1/getBookReservation

for graphql

https://<hostname>/graphql/getBookReservation

c)

Response

[

{

Name : “JS Defined”,

Author : “PQR”,

Tag : “JS”,

Year : 2015,

UserName : “ABC”,

Date : “14-Oct”

},

….

]

d)

https://<hostname>/api/v1/getBookReservation

[

{

Book\_name : “JS Defined”,

User\_Name : “ABC”,

Date\_of\_Reservation : “14-Oct”

},

…

]

for graphql

https://<hostname>/graphql/getBookReservation

e)

**c#**

public class Reservation  
{  
    public int Id { get; set; }  
    public string Book\_name { get; set; }  
    public string User\_Name { get; set; }  
    public DateTime Date\_of\_Reservation { get; set; }  
}

**for node**

class Book {  
  constructor(id, title, author, publicationDate, genre) {  
    this.id = id;  
    this. Book\_name = title;  
    this. user\_name = author;  
    this.publicationDate = publicationDate;  
  }  
}

 module.exports = Book;

f)

Post – insert

Put – update

Delete – delete

* 1. Assume that you will be designing an API for a record label. The record label has a database of artists with the following information:
* Artist name
* Artist genre
* Number of albums published under the label
* Artist usernamea

The API will let consumers obtain the list of artists stored in the database and add a new artist to the database. Use the following OpenAPI specification designed for this API to answer the below questions **[1 + 2 + 2 + 1 + 3 + 2 + 4 = 15]**

openapi: 3.0.0

info:

version: 1.0.0

title: Simple API

description: A simple API to illustrate OpenAPI concepts

servers:

- url: https://example.io/v1/

security:

- BasicAuth: []

paths:

/

get:

description: Returns a list of artists

parameters:

- $ref: '#/components/parameters/PageLimit'

- $ref: '#/components/parameters/PageOffset'

responses:

'200':

description: Successfully returned a list of artists

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/Artist'

'400':

$ref: '#/components/responses/400Error'

post:

description: Lets a user post a new artist

requestBody:

required: true

content:

application/json:

schema:

$ref: '#/components/schemas/Artist'

responses:

'200':

description: Successfully created a new artist

'400':

$ref: '#/components/responses/400Error'

/artists/{username}:

get:

description: Obtain information about an artist from his or her unique username

parameters:

- name: username

in: path

required: true

schema:

type: string

responses:

'200':

description: Successfully returned an artist

content:

application/json:

schema:

type: object

properties:

artist\_name:

type: string

artist\_genre:

type: string

albums\_recorded:

type: integer

'400':

$ref: '#/components/responses/400Error'

components:

securitySchemes:

BasicAuth:

type: http

scheme: basic

schemas:

Artist:

type: object

required:

- username

properties:

artist\_name:

type: string

artist\_genre:

type: string

albums\_recorded:

type: integer

username:

type: string

parameters:

PageLimit:

name: limit

in: query

description: Limits the number of items on a page

schema:

type: integer

PageOffset:

name: offset

in: query

description: Specifies the page number of the artists to be displayed

schema:

type: integer

responses:

400Error:

description: Invalid request

content:

application/json:

schema:

type: object

properties:

message:

type: string

1. If a new path “/albums” needs to be added in this specification, what will be change required in the base URL?
2. Assume that there are 100 labels available in the database. How a consumer application can retrieve record labels between ids 41 to 60?
3. Showcase a sample response generated for the following API invocation.

GET <https://example.io/v1/artists>

1. Showcase a sample response generated if the API mentioned in part (c ) fails.
2. Provide a snippet to be used under “/artists” GET path to enable passing the path parameter for restricting the number of returned record labels. How the API endpoint will look like?
3. Provide a snippet to be used under “/artists” GET path to handle 500 status code.
4. Provide a snippet to return record labels based on the artist name.

Ans:

a)

There is no change in base url, but we need to add one specification in open-api

/ albums:

get:

description: get albums list

b)

PageOffset and pagelimit

**/labels?start=41&end=60**.

c)

[  
{  
  artist\_name: Javed Ali  
  artist\_genre: test  
  albums\_recorded: 2  
},  
  {  
  artist\_name: Javed Ali  
  artist\_genre: test  
  albums\_recorded: 2  
}  
]

d)

{

statusCode: 400,

message: “Invalid request”

}

e)

<https://example.io/v1/?PageLimit=100>

/artists:  
  get:  
    description: Returns a list of artists  
    parameters:  
      - $ref: '#/components/parameters/PageLimit'  
      - $ref: '#/components/parameters/PageOffset'  
      - name: limit  
        in: query  
        description: Restricts the number of returned record labels  
        schema:  
          type: integer  
        example: 10  
    responses:  
      '200':  
        description: Successfully returned a list of artists  
        content:  
          application/json:  
            schema:  
              type: array  
              items:  
                $ref: '#/components/schemas/Artist'  
      '400':  
        $ref: '#/components/responses/400Error'

Or

paths:

/artists:

get:

summary: Retrieve artists

parameters:

- name: limit

in: query

description: Number of record labels to return

required: false

schema:

type: integer

minimum: 1

responses:

'200':

description: Successful response

# Response schema definition for 200 status

# Other properties and parameters for the GET operation

f)

responses:

500Error:

description: Internal server error

content:

application/json:

schema:

type: object

properties:

message:

type: string

or

'500':

         $ref: '#/components/responses/500Error'

paths:

/artists:

get:

summary: Retrieve artists

responses:

'200':

description: Successful response

# Response schema definition for 200 status

'500':

description: Internal Server Error

content:

application/json:

schema:

$ref: '#/components/schemas/Error'

# Other properties and parameters for the GET operation

g)

/artists/{artisitname}:

get:

description: Obtain information about an artist from his or her unique username

parameters:

- name: artistname

in: path

required: true

schema:

type: string

Or

paths:

/record-labels:

get:

summary: Retrieve record labels by artist name

parameters:

- name: artistName

in: query

description: Name of the artist

required: true

schema:

type: string

responses:

'200':

description: Successful response

# Response schema definition for 200 status

# Other properties and parameters for the GET operation

\*\*\*\*\*\*\*\*\*\*\*