MOBILE PROGRAMMING

LAB 2

Contents

Create github Lab2 with Lab2 folder on computer. Submit assignments to the moodle system.

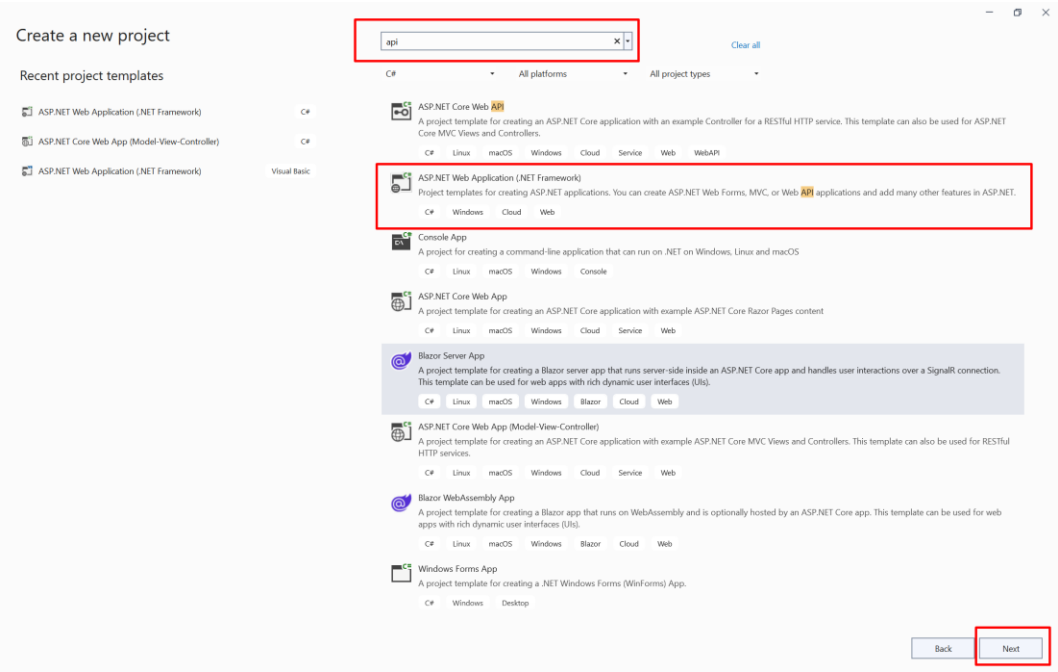
Question 1. Build APIs using MVC Asp.net.

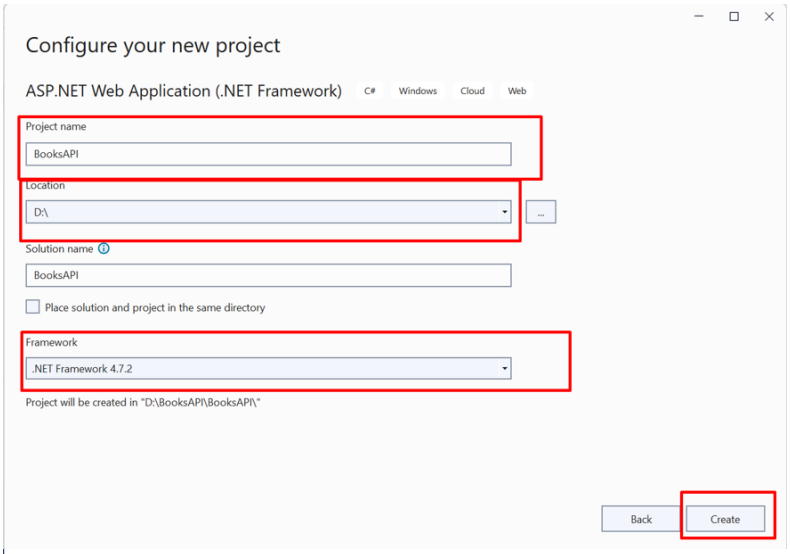
1. Create a project with the name: Q1WebAPI
2. Database
   Server: LibraryManagement.mssql.somee.com
   User: taitv_SQLLogin_1
   Pass: 12345678

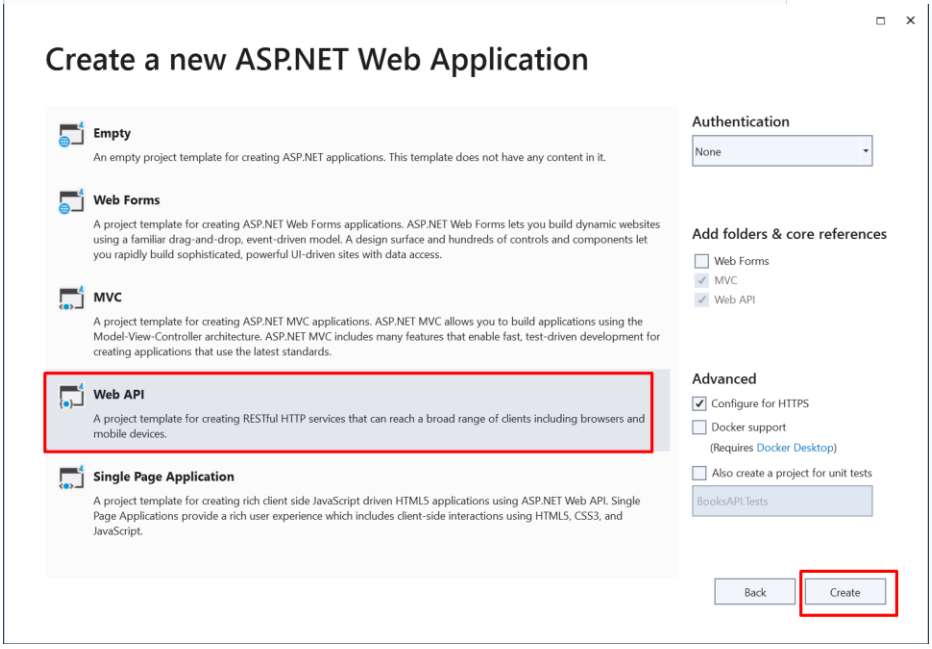
	Column Name	Data Type	Allow Nulls
🔑	bookid	int	☐
	name	nvarchar(100)	☑
	description	nvarchar(512)	☑
	price	decimal(18, 2)	☑
	note	nvarchar(128)	☑

bookid	name	description	price	note
3	Toán cao cấp	Các kiến thức toán học phục vụ cho chương trình đào tạo đại học	10000.00	Update
4	Kỹ thuật lập trình	Cung cấp các kiến thức lập trình cơ bản cho người mới bắt đầu học tin học	20000.00	New

3. Create a Web Api project with MS Visual Studio.

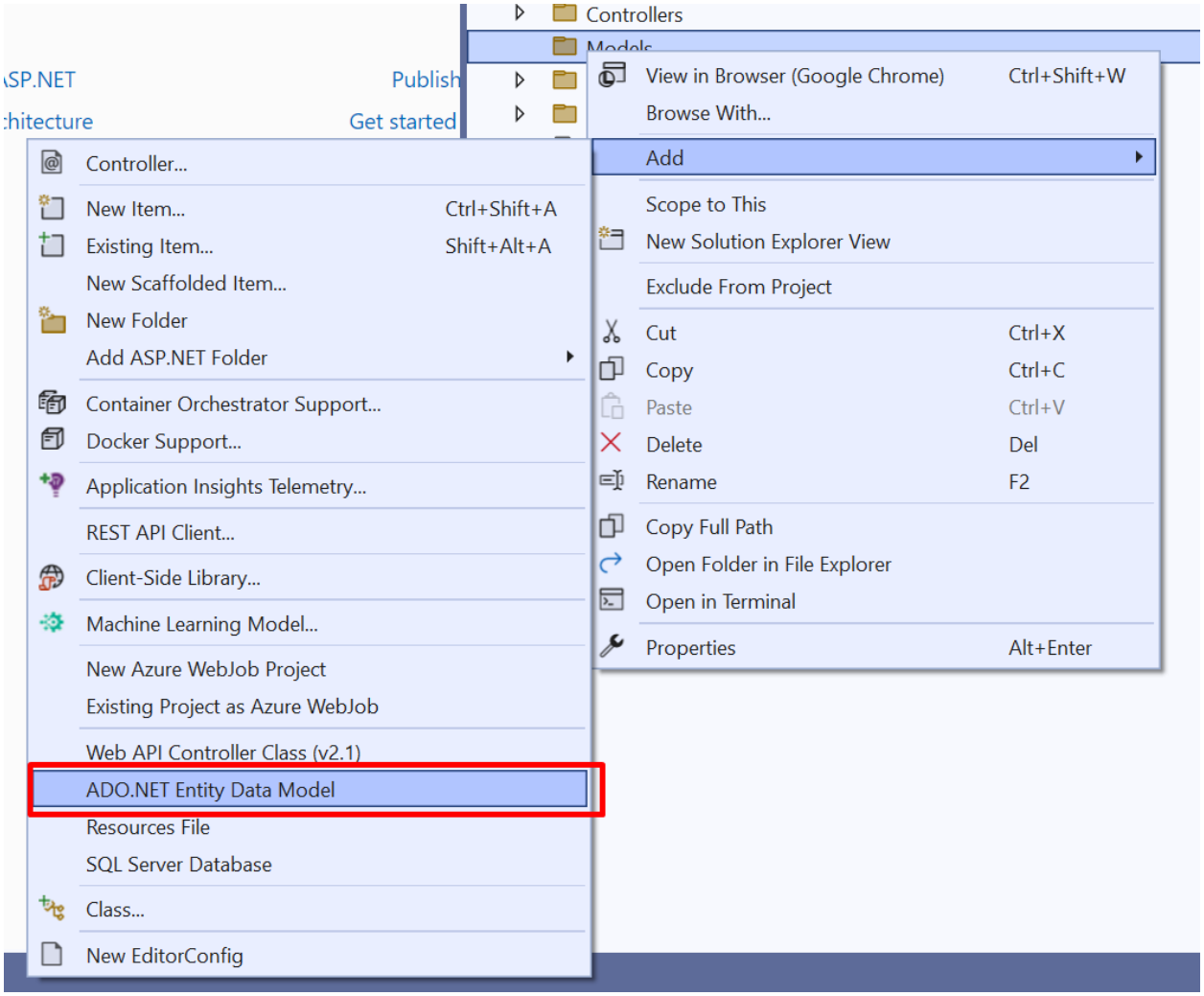




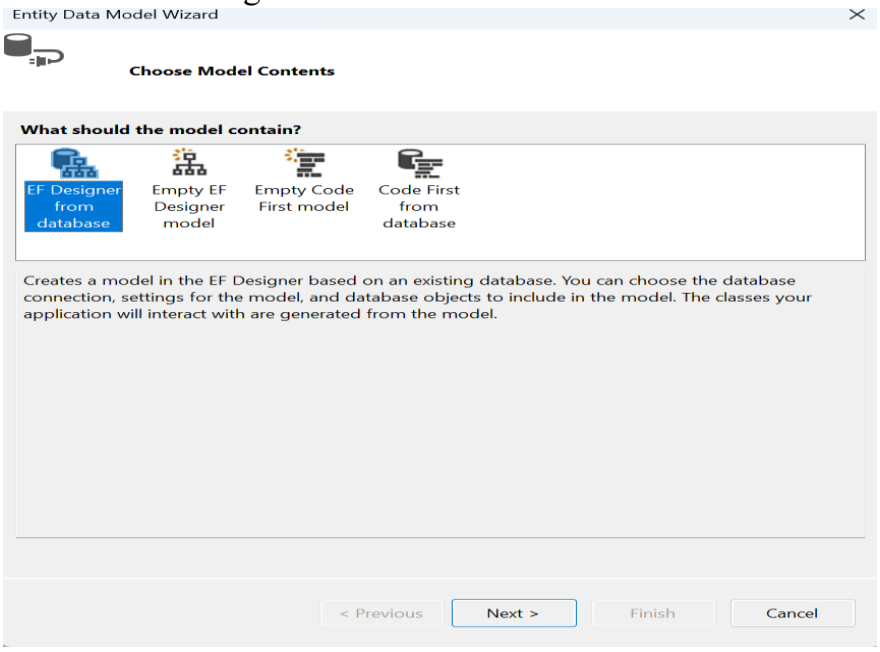


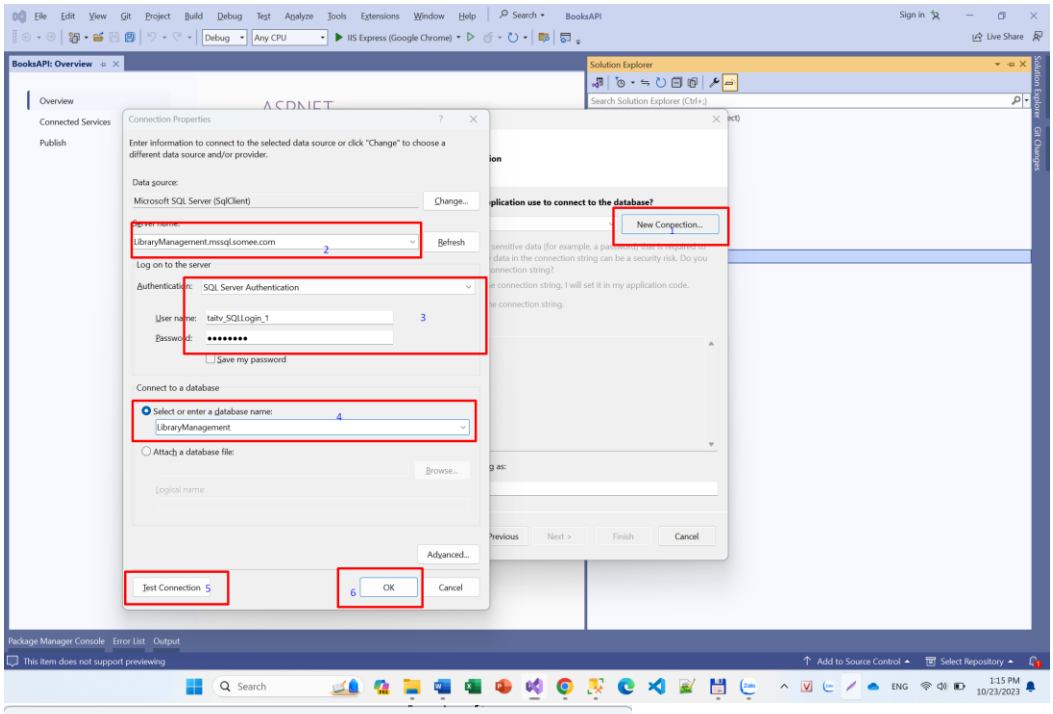
4. Create a Model using the Entity Framework.

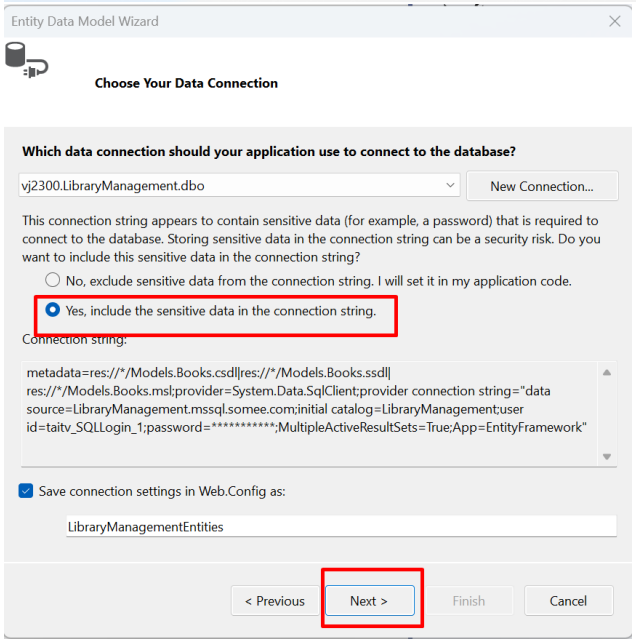
Right click on Model folder in Solution explorer, Click “ADO.net Entity Data Model”.

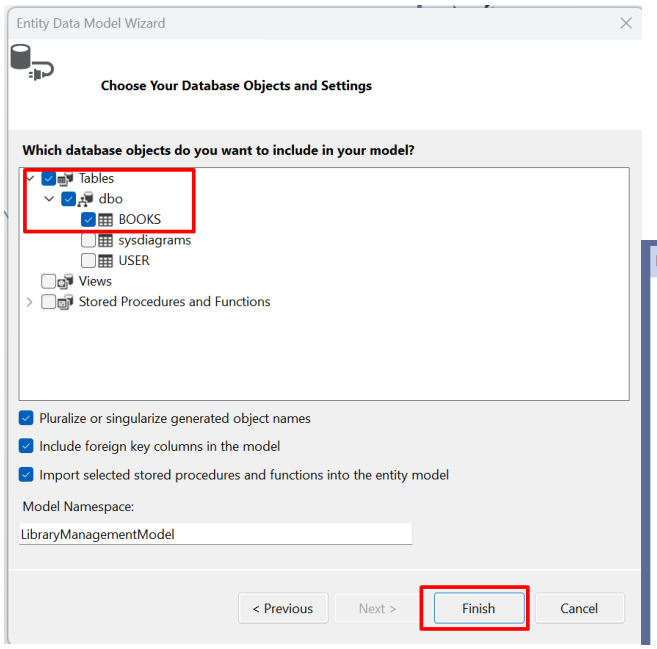


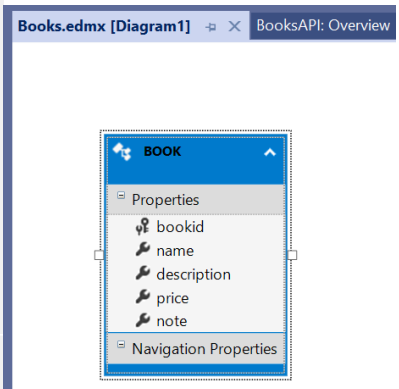
Choose “EF Designer from Database”.



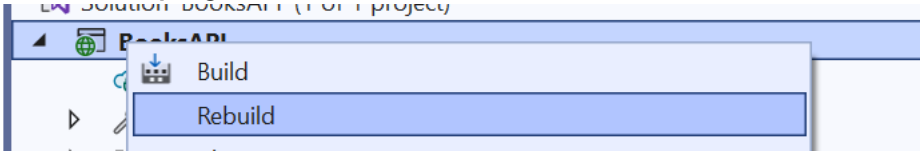

Connection Properties
Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.
Data source:
Microsoft SQL Server (SqlClient)
Change...
LibraryManagement.mssql.somee.com
Refresh
Log on to the server
Authentication:
SQL Server Authentication
User name:
taitv_SQLLogin_1
Password:
Save my password
Connect to a database
Select or enter a database name:
LibraryManagement
Attach a database file:
Browse...
Logical name:
Advanced...
Test Connection
OK
Cancel
New Connection...
Previous
Next >
Finish
Cancel


Entity Data Model Wizard
Choose Your Data Connection
Which data connection should your application use to connect to the database?
vj2300.LibraryManagement.dbo
New Connection...
This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?
No, exclude sensitive data from the connection string. I will set it in my application code.
Yes, include the sensitive data in the connection string.
Connection string:
metadata=res://*/Models.Books.csdl|res://*/Models.Books.ssdl|
res://*/Models.Books.msl;provider=System.Data.SqlClient;provider connection string="data source=LibraryManagement.mssql.somee.com;initial catalog=LibraryManagement;user id=taitv_SQLLogin_1;password=**********;MultipleActiveResultSets=True;App=EntityFramework"
Save connection settings in Web.Config as:
LibraryManagementEntities
< Previous
Next >
Finish
Cancel

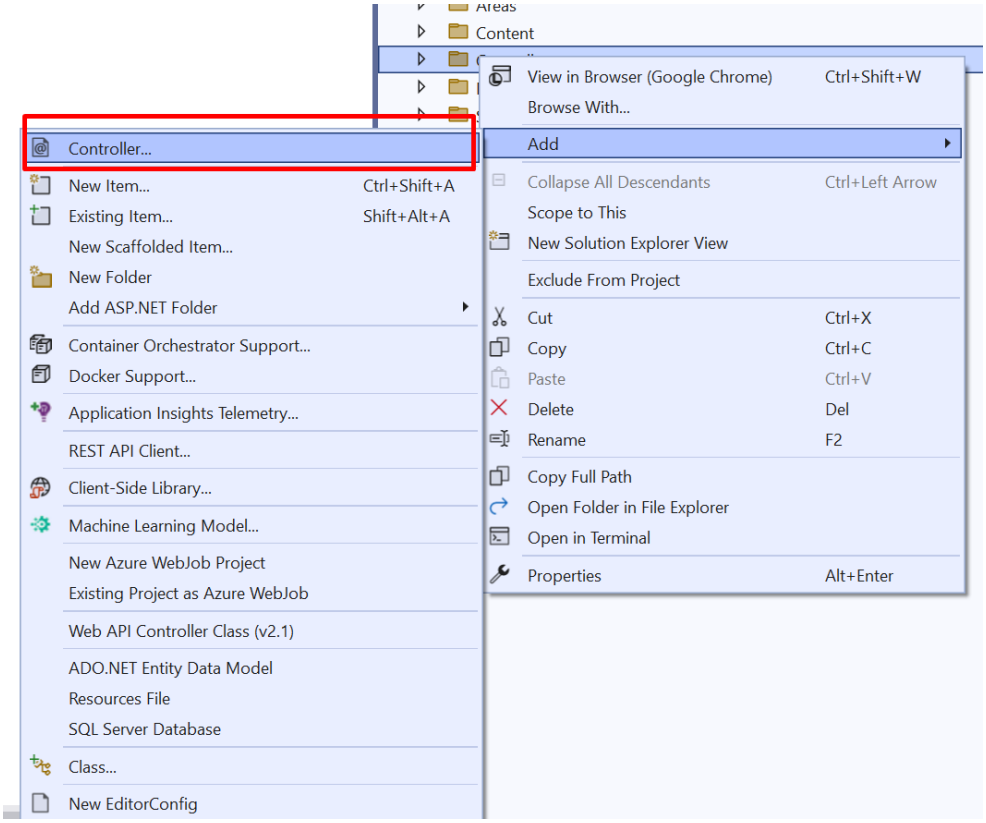


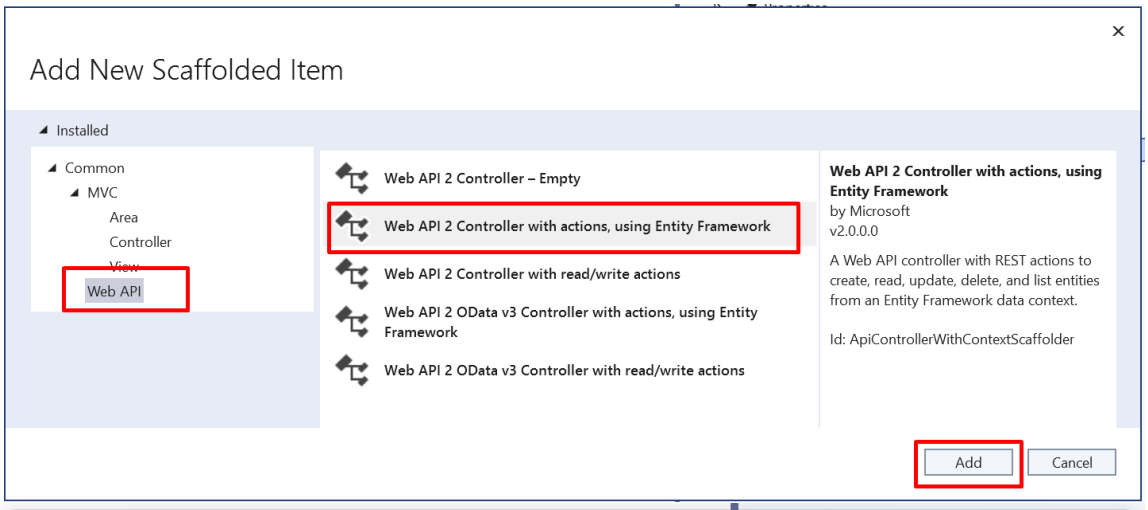


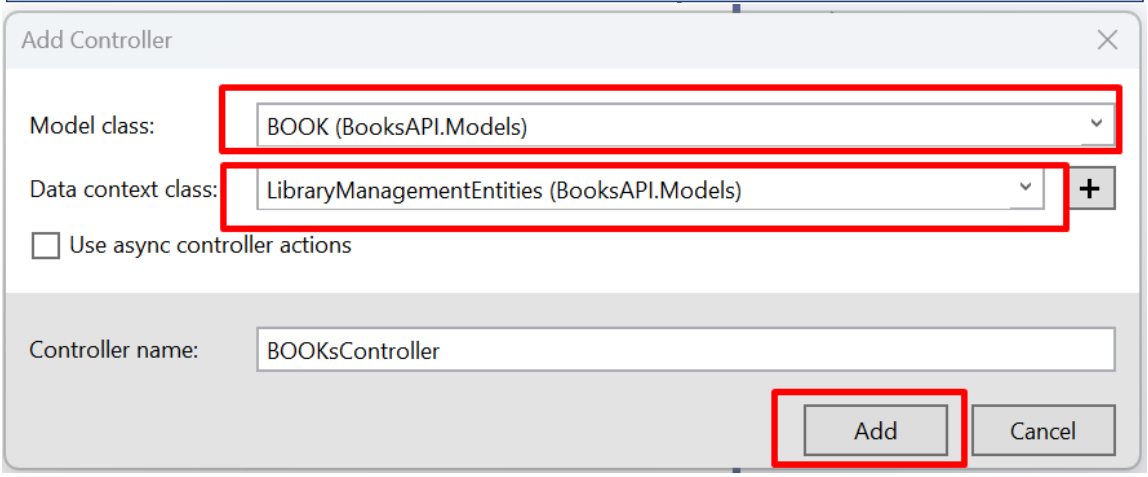
Rebuilt all project



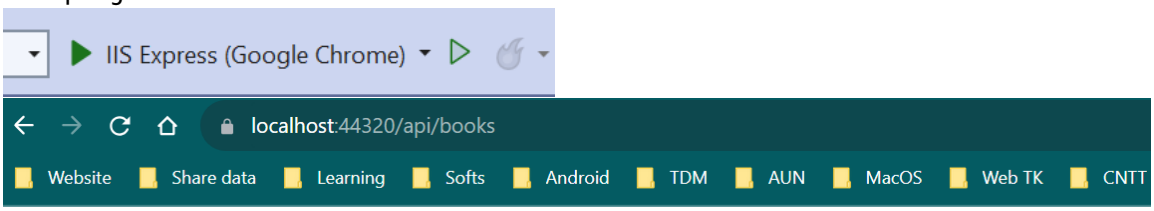
5. Create the BooksControler class







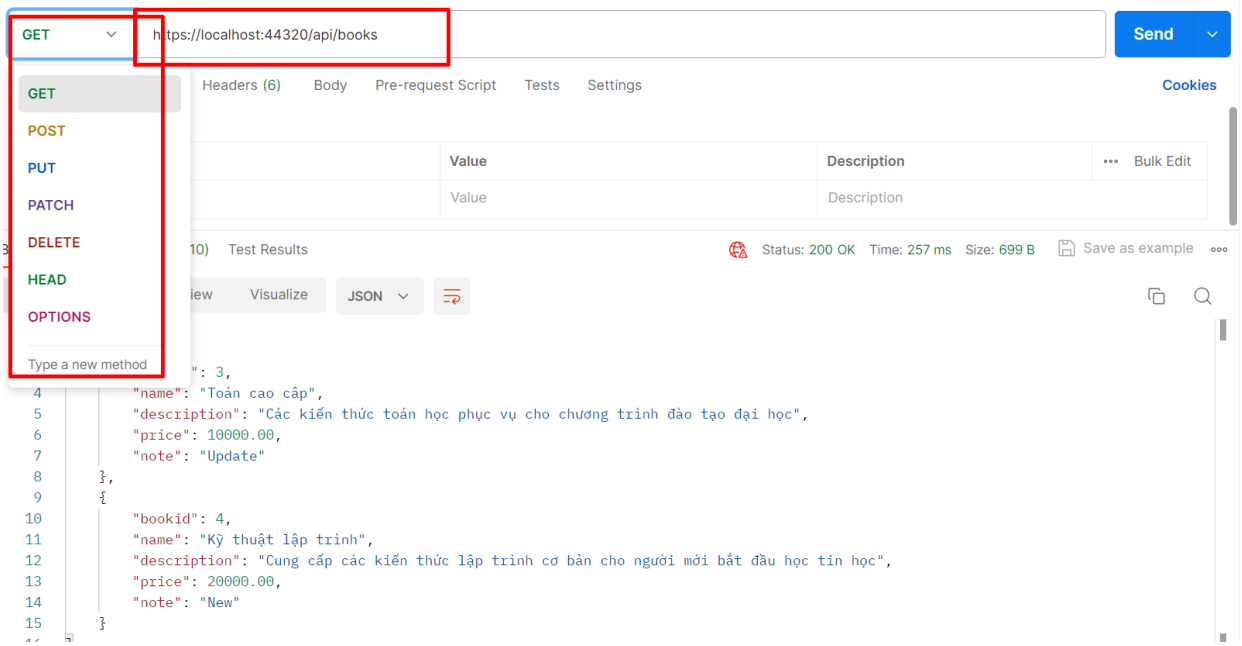
Run project:



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<ArrayOfBOOK xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/20
  ▼<BOOK>
     <bookid>3</bookid>
     <description>Các kiến thức toán học phục vụ cho chương trình đào tạo đại học</description>
     <name>Toán cao cấp</name>
     <note>Update</note>
     <price>10000.00</price>
   </BOOK>
  ▼<BOOK>
     <bookid>4</bookid>
     <description>Cung cấp các kiến thức lập trình cơ bản cho người mới bắt đầu học tin học</description>
     <name>Kỹ thuật lập trình</name>
     <note>New</note>
     <price>20000.00</price>
   </BOOK>
</ArrayOfBOOK>
```

6. Test the API with Postman.



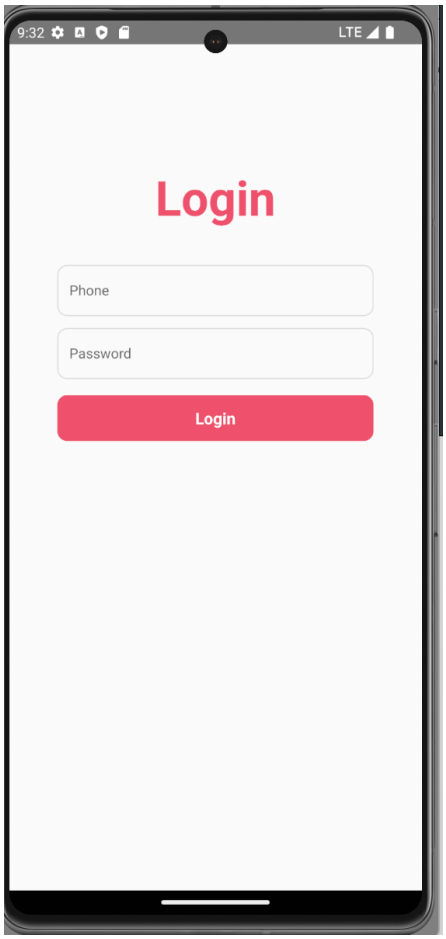
GET: http://localhost:xxxx/books (Get all books)

GET: http://localhost:xxxx/books/id (Get 1 book)

PUT: http://localhost:xxxx/books/id (Update 1 book)

DELETE: http://localhost:xxxx/books/id (Delete 1 book)

Question 2. Login screen interface design.



1. Create a project named Q2Login
2. Install the libraries to use: *npm i @react-navigation/native*
3. Create the Login folder.
4. Create a **Style.js**

```
import { DefaultTheme } from '@react-navigation/native';
import { StyleSheet } from 'react-native';

const AppTheme = {
  ...DefaultTheme,
  colors: {
    ...DefaultTheme.colors,
    primary: '#EF506B',
  },
};
export default styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    padding: 48,
  },
  title: {
    fontSize: 48,
    fontWeight: 'bold',
    color: AppTheme.colors.primary,
    marginBottom: 24,
    marginTop: 72,
  },
  input: {
    borderColor: AppTheme.colors.border,
    borderWidth: 1,
    width: '100%',
    marginTop: 12,
    borderRadius: 10,
    paddingLeft: 12,
  },
  button: {
    backgroundColor: AppTheme.colors.primary,
    borderRadius: 10,
    width: '100%',
    justifyContent: 'center',
    alignItems: 'center',
    padding: 12,
    marginTop: 16,
  },
  buttonText: {
    fontSize: 16,
    fontWeight: 'bold',
    color: '#FFF',
  },
});
```

5. Create a **Login.js** to build a login component

```
import {View, Text, TextInput, ScrollView,TouchableOpacity,} from 'react-native';
import Styles from './Styles';

const LoginScreen = () => {
  return (
   <ScrollView showsVerticalScrollIndicator={false}>
     <View style={Styles.container}>
       <Text style={Styles.title}>Login</Text>
       <TextInput
         style={Styles.input}
         placeholder="Phone"
       />
       <TextInput
         style={Styles.input}
         placeholder="Password"
         secureTextEntry
       />
       <TouchableOpacity style={Styles.button}>
         <Text style={Styles.buttonText}>Login</Text>
       </TouchableOpacity>
     </View>
   </ScrollView>
  );
};

export default LoginScreen;
```

6. Add the following command to the **App.tsx** file:

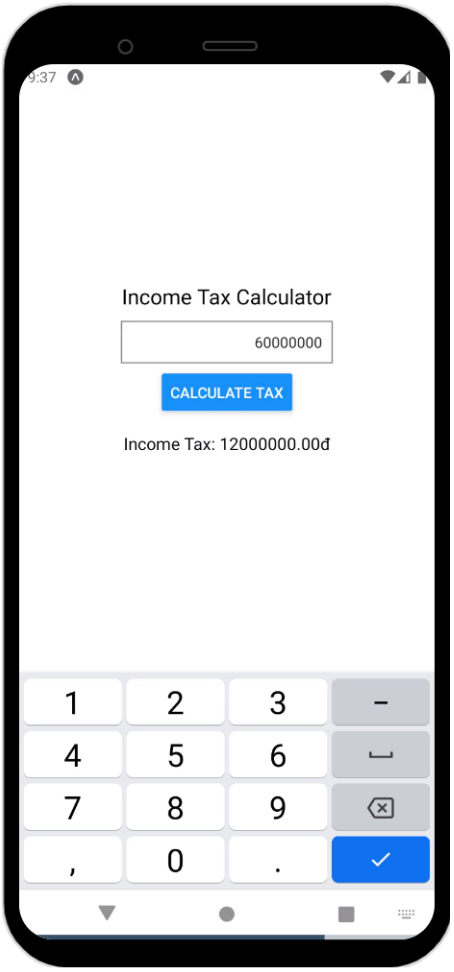
```
import React from 'react'
import LoginScreen from './Login/Login'

export default App =()=>{

  return (
   <LoginScreen/>
  );
};
```

7. Run command "**react-native run-android**" to run the program.

Question 3. Write the Personal Income Tax Calculation program as shown below.



1. Create a project with the name: Q3IncomeTax.
2. Create file style.js to create styles for the interface

```
import {StyleSheet} from 'react-native'
export default styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    padding: 16,
  },
  title: {
    fontSize: 20,
    marginBottom: 10,
  },
  input: {
    width: 200,
    height: 40,
    borderColor: 'gray',
    borderWidth: 1,
    marginBottom: 10,
    padding: 10,
  },
  result: {
    marginTop: 20,
    fontSize: 16,
  },
});
```

3. Open file App.tsx and edit the content as below.


```
import React, { useState } from 'react';
import { View, Text, TextInput, Button } from 'react-native';
import styles from './style'

const App = () => {
  const [income, setIncome] = useState('');
  const [tax, setTax] = useState('');

  const calculateTax = () => {
    const incomeAmount = parseFloat(income);

    if (isNaN(incomeAmount) || incomeAmount < 0) {
      setTax('Invalid income');
      return;
    }

    let taxAmount = 0;
    if (incomeAmount <= 10000000) {
      taxAmount = incomeAmount * 0.1;
    } else if (incomeAmount <= 50000000) {
      taxAmount = 10000000 * 0.1 + (incomeAmount - 10000) * 0.2;
    } else {
      taxAmount = 10000000 * 0.1 + 40000000 * 0.2 + (incomeAmount - 50000000) * 0.3;
    }

    setTax(`Income Tax: ${taxAmount.toFixed(2)}đ`);
  };
```

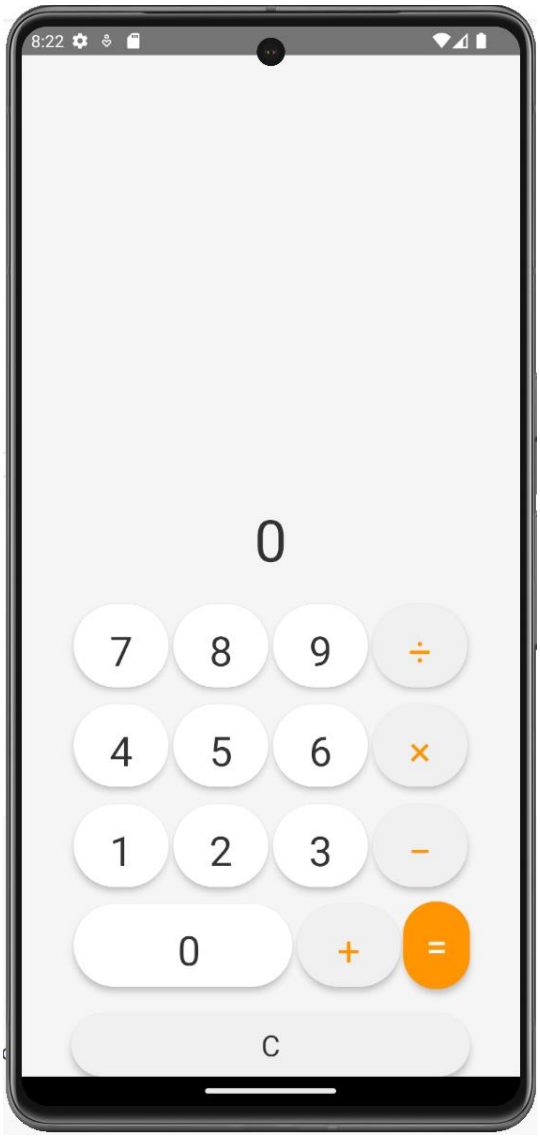
```
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Income Tax Calculator</Text>
      <TextInput
        textAlign = "right"
        style={styles.input}
        placeholder="Enter your income"
        keyboardType="numeric"
        value={income}
        onChangeText={text => setIncome(text)}

      />
      <Button title="Calculate Tax" onPress={calculateTax} />
      <Text style={styles.result}>{tax}</Text>
    </View>
  );
};

export default App;
```

4. Run command **"react-native run-android"** to run the program.

Question 4. Caculator screen interface design



1. Create a project with the name: Q3Caculator.
2. Create file style.js to create styles for the interface.
3. JavaScript functions that support execution. Call the event of the button.

```
// State variables
   const [displayValue, setDisplayValue] = useState('0');
   const [operator, setOperator] = useState(null);
   const [firstValue, setFirstValue] = useState('');

   // Function to handle number inputs
   const handleNumberInput = (num) => {

       if (displayValue === '0') {
          setDisplayValue(num.toString());
```

```
    } else {
        setDisplayValue(displayValue + num);
    }
};

// Function to handle operator inputs
const handleOperatorInput = (operator) => {

       setOperator(operator);
       setFirstValue(displayValue);
       setDisplayValue('0');
};

   // Function to handle equal button press
const handleEqual = () => {

    const num1 = parseFloat(firstValue);
       const num2 = parseFloat(displayValue);

       if (operator === '+') {
        setDisplayValue((num1 + num2).toString());
       } else if (operator === '-') {
        setDisplayValue((num1 - num2).toString());
    } else if (operator === '*') {
        setDisplayValue((num1 * num2).toString());
    } else if (operator === '/') {
        setDisplayValue((num1 / num2).toString());
       }

       setOperator(null);
       setFirstValue('');
   };

   // Function to handle clear button press
   const handleClear = () => {

       setDisplayValue('0');
       setOperator(null);
       setFirstValue('');
   };
```