

Hectre Demo

Install Instructions

&

UniTests Instructions

By

Yotam Lin

Database Instructions

1. **Install Microsoft SQL Server Management Studio (SSMS):**

If you don't have SSMS installed, you can download it from the official Microsoft website. SSMS is a powerful tool for managing SQL Server databases and allows you to execute SQL queries.

2. **Download the SQL Script:**

Locate the "DB.sql" file in your project folder "hectre-demo\DB Scripts\DB.sql". This file contains the SQL script required to create your database schema and tables.

3. **Open SSMS and Connect to SQL Server:**

Launch Microsoft SQL Server Management Studio (SSMS) and connect to your SQL Server instance where you want to create the database. You'll need the appropriate server credentials to connect.

4. **Open a New Query Window:**

In SSMS, select "New Query" from the "File" menu or use the keyboard shortcut (Ctrl+N) to open a new query window or press the “New Query” button on the toolbar.

5. **Drag and Execute the SQL Script:**

Drag the "DB.sql" file into the query window. The contents of the SQL script will be displayed in the query editor.

6. **Execute the SQL Script:**

Press the "Execute" button (or use the F5 key) to run the SQL script. This will create the database, along with the necessary schema and tables, based on the instructions in the "DB.sql" script.

7. **Verify Database Creation:**

After the script execution is completed, check the "Hectre" database has been created successfully. You can expand the "Databases" node in SSMS to see the list of databases, and "Hectre" should be listed.

Congratulations, you have now successfully set up the "Hectre" database for your project!

**\*\* Notes:**

Please note that the steps mentioned above assume that you have a running SQL Server instance and the necessary privileges to create databases and execute SQL scripts. If you encounter any issues or errors during the process, ensure that you have the required permissions and that the SQL Server instance is properly configured.

Please note that there are more scripts in the “DB Scripts” library. You can use them to create tables individually (you don’t have to because the DB.sql creates everything you need including the data)

Backend Instructions

1. **Open Visual Studio Solution:**

Navigate to the "hectre-demo\Hectre API" folder and locate the "Hectre API.sln" file. Double-click on the "Hectre API.sln" file to open it in Visual Studio.

2. **Verify Database Name:**

Before running the backend project, you need to verify the name of your SQL Server instance. Open SQL Server Management Studio (SSMS) and find the name of your server instance. It should look something like: "Server=DESKTOP-P17TH39\\SQLEXPRESS;". Copy this server name for the next step. (It will be in the top of the Object explorer)

3. **Update Connection String in HectreContext.cs:**

In Visual Studio, locate the "HectreContext.cs" file under the "01 - Data Access Layer" folder. Inside this file, you'll find the `OnConfiguring` method with the connection string:

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) => optionsBuilder.UseSqlServer("Server=DESKTOP-P17TH39\\SQLEXPRESS;Database=Hectre;Trusted\_Connection= True;TrustServerCertificate=True");

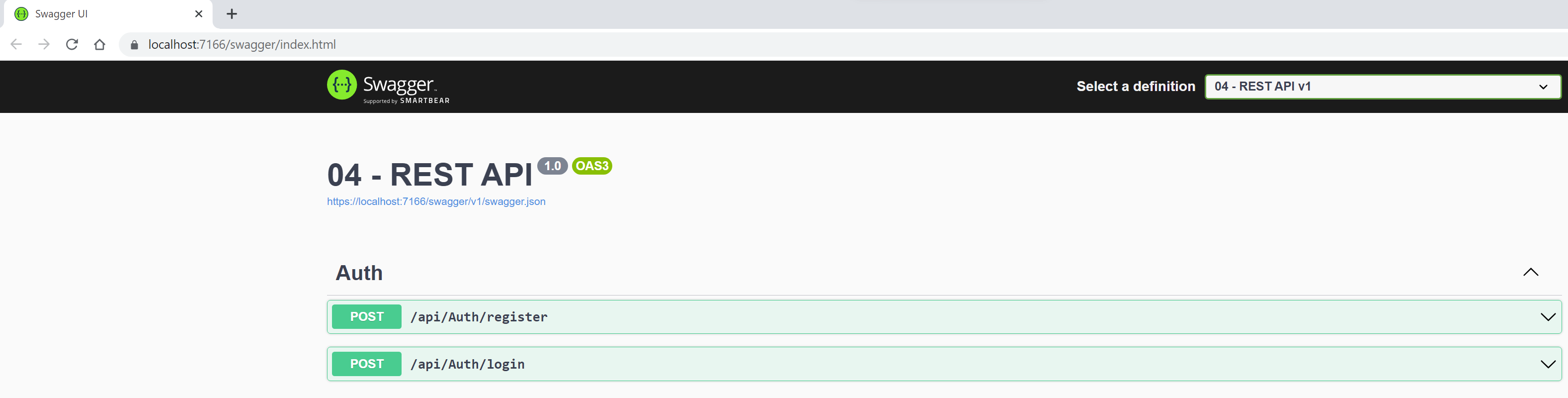
Replace the server name "DESKTOP-P17TH39\\SQLEXPRESS" with your SQL Server instance name that you copied in the previous step. For example:

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) => optionsBuilder.UseSqlServer("Server=YOUR\_SERVER\_NAME\\SQLEXPRESS;Database=Hectre;Trusted\_Connection=True;TrustServerCertificate=True");

4. **Start the Backend Project:**

In Visual Studio, ensure that the backend project (Hectre API) is selected as the startup project. Click on the "Start Debugging" button (green play icon) or press the F5 key to start the backend project. This will run the backend server, and it will be accessible at a specified URL (usually "https://localhost:portNumber"). \*\*

**A Swagger page will open up (great for testing) you can see there the url.  
save the url – you’ll need it for the frontend project.**



5. **Keep the Backend Running:**

While you are using the website or frontend, keep the backend project running in Visual Studio. The backend server needs to be active to handle requests from the frontend and interact with the database.

Congratulations, you have successfully set up the backend project and connected it to your database! Now, you can use the website and interact with the backend to perform various actions and retrieve data from the database.

Frontend Instructions

1. **Open Frontend Project in Visual Studio Code:**

Navigate to the "hectre-demo\client" folder and open Visual Studio Code. From the menu, select "File" > "Open Folder" and choose the "client" folder.

2. **Install Required Modules:**

Open a new terminal in Visual Studio Code. If not already in the "client" folder, navigate to it using the `cd` command in the terminal. For example:

cd hectre-demo/client

Once inside the "client" folder, run the following command to install all the required modules specified in the "package.json" file:

npm install

This command may take some time as it installs all the dependencies needed for the frontend project.

3. **Configure Backend URL:**

You need to ensure that the frontend communicates with the correct backend server. Locate the following file:

`src/config.js`

Replace the URL in line 1 (const API\_PARTIAL\_URL = 'https://localhost:7166/api';) with the URL of your backend server.

For example:

const API\_ PARTIAL \_URL = 'https://your-backend-server-url/api';

4. **Start the Frontent:**

In the terminal (still in the "client" folder), run the following command to start the frontend server:

npm start

This will open a new window in your default browser, and you should see the sign-in/sign-up window.

5. **Access the Website:**

You can register a new user or use the provided credentials to log in:

Email: [linyotam@gmail.com](mailto:linyotam@gmail.com)  
 Password: bkwz6133

After logging in, you will have access to the website, and you can explore its features and functionalities.

Congratulations, you have successfully set up and started the frontend project! You can now interact with the website and explore its various components and capabilities. Enjoy using the website!

Backend API Instructions

**Instruction to Use API from Postman and Swagger**

1. **Postman**:

a. Open Postman, either the standalone application or the browser version.

b. Set the HTTP method (GET, POST, PUT, DELETE) for the API request you want to test.

c. Enter the API endpoint URL in the address bar. For example:

* Register (POST): `https://your-api-url/api/Auth/register`
* Login (POST): `https://your-api-url/api/Auth/login`
* Get Harvests (GET): `https://your-api-url/api/Harvests`
* Get Orchards (GET): `https://your-api-url/api/Orchards`

d. If the API endpoint requires authentication, you need to add a bearer token to your request's headers.

* To add the bearer token to your request headers:
* Obtain a valid JWT (JSON Web Token) by first making a successful login request. The response will include an "access\_token" property.
* Copy the JWT token from the login response.
* In Postman, go to the "Headers" section of your request.
* Add a new header with the key "Authorization" and the value "Bearer {paste-your-jwt-token-here}".
* Replace "{paste-your-jwt-token-here}" with the actual JWT token you copied.

e. If the API endpoint requires additional parameters, such as request body or query parameters, provide them in the respective sections.

f. Click the "Send" button to send the API request.

g. Observe the response in the "Body" section of Postman. The response will contain the data or error message returned by the API.

2. **Swagger:**

a. Open your web browser and enter the URL for the Swagger UI documentation provided by your API. Usually, it is something like: `https://your-api-url/swagger`.

b. In the Swagger UI, navigate through the available endpoints to find the service you want to test, such as "Register," "Login," "Get Harvests," or "Get Orchards."

c. For each endpoint, you can click on the "Try it out" button to interact with the API.

d. If the API endpoint requires authentication, you will find an "Authorize" button in the top-right corner of the Swagger UI.

* Click on the "Authorize" button, a popup will appear.
* Enter your JWT token (obtained from a successful login) in the "Value" field, and click "Authorize."

e. Fill in any required parameters, such as request body or query parameters, for the specific API request.

f. Click the "Execute" button to send the API request.

g. Observe the response data in the "Response Body" section below the API request parameters. The response will contain the data or error message returned by the API.

Backend UniTests Instructions

**Test file location:**

04 – REST API 🡪 Helpers 🡪 HarvestsTests.cs

**Test Class:**

HarvestsTests

**Purpose:**

Contains test methods to validate the functionality of the HarvestsLogic class.

**Test Method:**

GetAllHarvests\_ShouldReturnValidHarvests

**Purpose:**

Test that the GetAllHarvests method of the HarvestsLogic class returns a valid list of harvests, and all BinCounts are non-negative.

**Test Execution and Assertions:**

1. Call the GetAllHarvests method from the HarvestsLogic class.

2. Assert that the returned list of harvests is not null.

3. Assert that the returned list of harvests is not empty.

4. For each harvest in the list, assert that the BinCount property is non-negative.

**How to Run the Tests:**

1. Open the Visual Studio solution "Hectre API.sln."

2. Locate the "HarvestsTests" class in the "HarvestsTests.cs" file.

3. Right-click on the "HarvestsTests" class or individual test methods.

4. Select "Run Tests" or use the test runner (e.g., MSTest, NUnit, xUnit) of your choice.

**Conclusion:**

The "GetAllHarvests\_ShouldReturnValidHarvests" test method successfully validated that the GetAllHarvests method returns a valid list of harvests, and all BinCounts are non-negative as expected.

**Test Class:**

HarvestsTests

**Purpose:**

Contains test methods to validate the functionality of the HarvestsLogic class.

**Test Method:**

GetHarvestsByOrchardAndDate\_ShouldReturnValidHarvests

**Purpose:**

Test that the GetHarvestsByOrchardAndDate method of the HarvestsLogic class returns a valid list of harvests based on the specified orchardIds, startDate, and endDate.

**Test Method Setup:**

1. Create an instance of the HarvestsLogic class.

2. Create test data for orchardIds, startDate, and endDate.

**Test Execution and Assertions:**

1. Call the GetHarvestsByOrchardAndDate method from the HarvestsLogic class, passing the test data for orchardIds, startDate, and endDate.

2. Assert that the returned list of harvests is not null.

3. Assert that the returned list of harvests is not empty.

**How to Run the Tests:**

1. Open the Visual Studio solution "Hectre API.sln."

2. Locate the "HarvestsTests" class in the "HarvestsTests.cs" file.

3. Right-click on the "HarvestsTests" class or individual test methods.

4. Select "Run Tests" or use the test runner (e.g., MSTest, NUnit, xUnit) of your choice.

**Conclusion:**

The "GetHarvestsByOrchardAndDate\_ShouldReturnValidHarvests" test method successfully validated that the GetHarvestsByOrchardAndDate method returns a valid list of harvests for the specified orchardIds, startDate, and endDate as expected.

**Test Class:**

HarvestsTests

**Purpose:**

Contains test methods to validate the functionality of the HarvestsLogic class.

**Test Method:** AddNewHarvest\_WhenOrchardExists\_ShouldUseExistingOrchard

**Purpose:**

Test that the AddNewHarvest method of the HarvestsLogic class correctly uses an existing orchard when adding a new harvest.

**Test Method Setup:**

1. Create an instance of the HarvestsLogic class, injecting a mock HectreContext that provides the necessary mock behavior for the method under test.

2. Create an existing orchard with a unique identifier and relevant properties (Id, Name, Block, SubBlock).

3. Set up the mock HectreContext to return the existing orchard when queried for an existing orchard with a specified Id.

4. Create a new HarvestModel instance and set its Orchard property with the Id of the existing orchard.

5. Instantiate the HarvestsLogic class with the mock HectreContext.

**Test Execution and Assertions:**

1. Call the AddNewHarvest method from the HarvestsLogic class, passing the newly created harvest model.

2. Assert that the returned HarvestModel instance contains the same reference to the existing orchard as expected.

3. Verify that the Add method on the Orchards DbSet was not called (as the orchard already exists).

4. Verify that the Add method on the Harvests DbSet was called once (to add the new harvest).

5. Verify that the SaveChanges method on the DbContext was called once to save the changes.

**How to Run the Tests:**

1. Open the Visual Studio solution "Hectre API.sln."

2. Locate the "HarvestsTests" class in the "HarvestsTests.cs" file.

3. Right-click on the "HarvestsTests" class or individual test methods.

4. Select "Run Tests" or use the test runner (e.g., MSTest, NUnit, xUnit) of your choice.

**Conclusion:**

The "AddNewHarvest\_WhenOrchardExists\_ShouldUseExistingOrchard" test method successfully validated that the AddNewHarvest method uses an existing orchard when adding a new harvest, as expected.

**Test Class:**

HarvestsTests

**Purpose:**

Contains test methods to validate the functionality of the HarvestsLogic class.

**Test Method:** AddNewHarvest\_WhenOrchardDoesNotExist\_ShouldCreateNewOrchard

**Purpose:**

Test that the AddNewHarvest method of the HarvestsLogic class correctly creates a new orchard when adding a new harvest and the orchard does not exist.

**Test Method Setup:**

1. Create an instance of the HarvestsLogic class, injecting a mock HectreContext that provides the necessary mock behavior for the method under test.

2. Set up the mock HectreContext to return null when queried for an existing orchard with a specified Id (simulating that the orchard does not exist).

3. Create a new Orchard instance (addedOrchard) to track the added orchard when the Add method is called on the Orchards DbSet.

4. Set up the mock HectreContext to capture the added orchard when the Add method is called on the Orchards DbSet.

5. Create a new HarvestModel instance and set its Orchard property with relevant properties (Id, Name, Block, SubBlock) of the new orchard.

6. Instantiate the HarvestsLogic class with the mock HectreContext.

**Test Execution and Assertions:**

1. Call the AddNewHarvest method from the HarvestsLogic class, passing the newly created harvest model.

2. Assert that the returned HarvestModel instance contains the same reference to the newly added orchard as expected.

3. Verify that the Add method on the Orchards DbSet was called once (to add the new orchard).

4. Verify that the Add method on the Harvests DbSet was called once (to add the new harvest).

5. Verify that the SaveChanges method on the DbContext was called once to save the changes.

**How to Run the Tests:**

1. Open the Visual Studio solution "Hectre API.sln."

2. Locate the "HarvestsTests" class in the "HarvestsTests.cs" file.

3. Right-click on the "HarvestsTests" class or individual test methods.

4. Select "Run Tests" or use the test runner (e.g., MSTest, NUnit, xUnit) of your choice.

**Conclusion:**

The"AddNewHarvest\_WhenOrchardDoesNotExist\_ShouldCreateNewOrchard" test method successfully validated that the AddNewHarvest method creates a new orchard when adding a new harvest and the orchard does not exist, as expected.

Frontend UniTests Instructions

1. Open Visual Studio Code or any other text editor/IDE.

2. Navigate to the frontend project folder, specifically the "client" folder, using the terminal.

3. Locate the test file for the Helper component. It should be named "Helper.test.js" and located under the "components\Helper" folder.

4. Make sure that all the necessary test dependencies are installed by running the following command in the terminal:

npm install

5. You can run the unit tests using the following command:

npm test

6. After running the `npm test` command, the test suite for the Helper component will start running. You will see a summary of the test results in the terminal.

7. The test results will display a list of test suites and their corresponding pass or fail status. Green checkmarks indicate successful tests, while red crosses indicate test failures.

8. If any tests fail, the terminal will provide information about which test cases failed and the reason for the failure. Review the test results to identify the specific issues in the code.

9. Once all the tests have been executed, the terminal will provide a summary of the total number of tests executed, the number of test suites, the number of tests passed, and the number of tests failed.

10. If any tests fail, carefully review the test code and the Helper component code to identify and fix the issues.

11. After making necessary code changes, rerun the tests using the `npm test` command to ensure that all tests pass successfully.

12. Continue writing and running additional tests for other components as needed.

Conclusion:

Running `npm test` will execute the unit tests for the Helper component and display the test results in the terminal. Use this process to regularly test the frontend React components and ensure the code is functioning correctly and without any regressions. Always run the tests before deploying changes to the production environment to maintain code quality and reliability.