

Hectre Demo Documentation

By

Yotam Lin

Architecture Overview:

ASP.NET Core with React

**Frontend - React:**

* React is used for building the user interface of the web application.
* It is a JavaScript library for building modern, responsive, and interactive user interfaces.
* React components are used to create reusable UI elements that represent different parts of the application's views.

**Backend - ASP.NET Core:**

* ASP.NET Core is the server-side framework used to build the backend of the web application.
* It provides a robust and scalable infrastructure for handling HTTP requests, routing, and data processing.
* The backend is responsible for processing requests from the frontend, interacting with the database, and sending responses back to the client.

**Database - MS SQL Server:**

* MS SQL Server is used as the relational database management system for storing and managing data.
* It provides a secure and scalable solution for storing application data in a structured format.
* The database stores various entities and their relationships, including user information, application data, and other business-related data.

**API Endpoints:**

* ASP.NET Core exposes various API endpoints that handle HTTP requests from the frontend.
* These endpoints are responsible for receiving and processing data from the client and returning relevant data or responses.
* Endpoints are defined in the ASP.NET Core controllers, each serving specific functionalities of the application.

**Authentication and Authorization:**

* ASP.NET Core provides built-in support for authentication and authorization.
* Authentication ensures that users are who they claim to be, and authorization controls what resources and actions each user can access.
* JSON Web Tokens (JWT) are commonly used for authentication, and roles-based access control is used for authorization.

**Redux-Toolkit:**

* Redux is a state management library that can be used to manage application state in a more predictable and centralized way.
* It is commonly used in React applications to handle complex state management and data flow between components.
* Redux stores application state and provides a way to update and access the state across the entire application.

**Testing:**

* Unit testing and integration testing are essential parts of the development process to ensure the quality and correctness of the application.
* Unit tests are written for individual components or functions to verify their behavior in isolation.
* Integration tests are used to test the interaction between different components or modules.

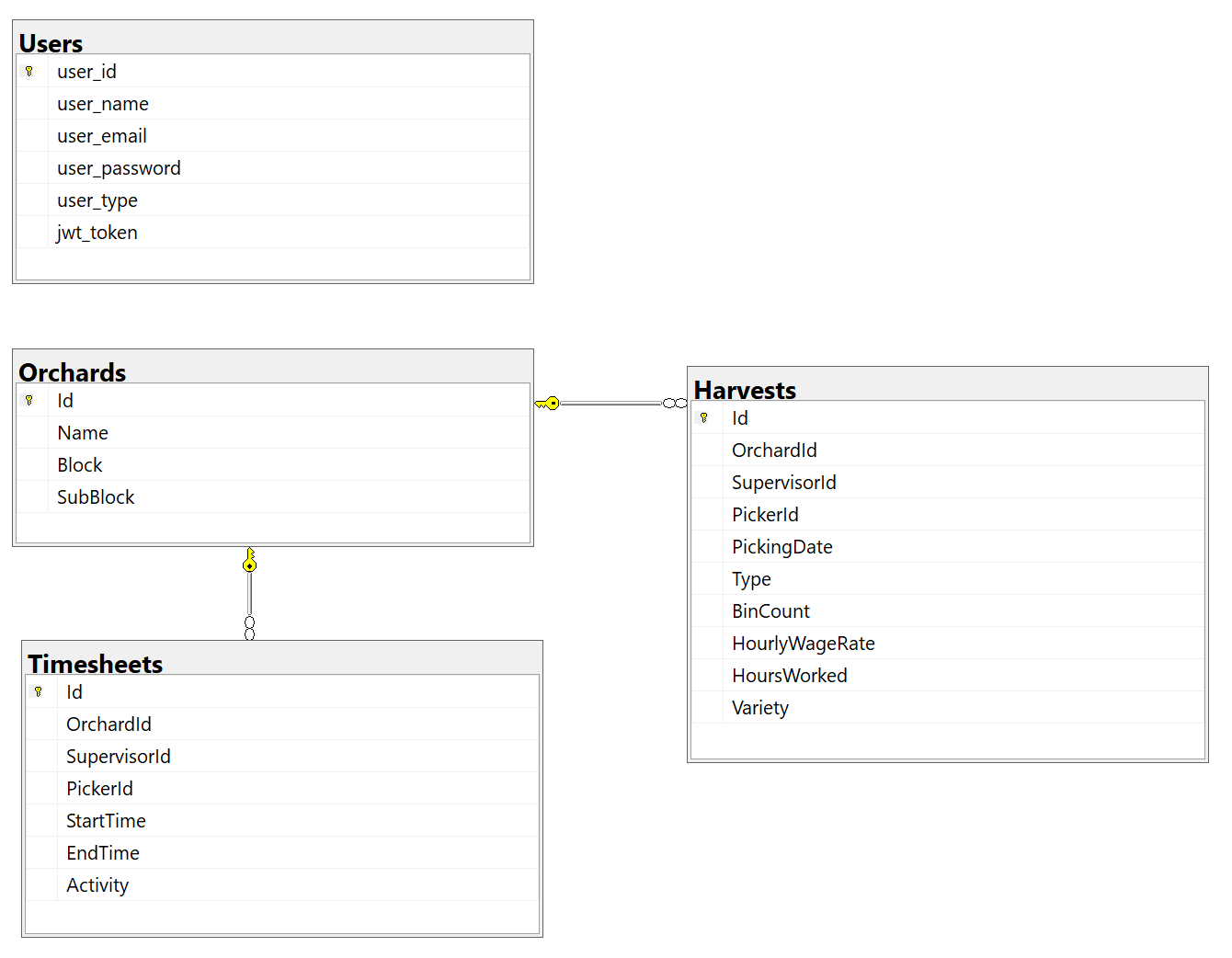
Overall Workflow:

* The frontend built with React sends HTTP requests to the ASP.NET Core backend.
* The ASP.NET Core backend processes the requests, authenticates and authorizes users (if required), and performs the necessary data operations on the MS SQL Server database.
* The backend then sends the response back to the frontend, which updates the UI based on the received data.
* Redux manages the state of the application, ensuring a consistent flow of data and actions between components.
* Unit tests and integration tests are written and executed to verify the correctness and functionality of different components and the system as a whole.

Advantages of the Chosen Architecture:

* Separation of concerns: The architecture separates frontend and backend logic, making the application more maintainable and scalable.
* Performance: ASP.NET Core provides excellent performance for handling a large number of requests.
* Reusability: React components can be reused throughout the application, leading to a more modular codebase.
* Scalability: The architecture can easily scale to handle increasing user demands.
* Security: ASP.NET Core provides robust security features, and JWT authentication enhances the security of user data.

Database Diagram & Tables



Backend Design & Components:

**Data Access Layer:**

* The Data Access Layer is responsible for interacting with the database using Entity Framework and the HectreContext class.
* It contains entity classes like Harvest, Orchard, Timesheet, and User, representing the data structure of the corresponding database tables.
* The Data Access Layer handles database operations such as querying, inserting, updating, and deleting data.
* It provides an abstraction over the underlying database, allowing other layers to work with data without worrying about the database details.

**Business Model Layer:**

* The Business Model Layer builds upon the Data Access Layer and includes the structure of each class from the Data Access Layer.
* This layer performs data validation and annotation, ensuring that the data passed to the Data Access Layer is accurate and meets specific requirements.
* It may include data transformation and normalization to present a consistent and user-friendly interface to the Business Logic Layer.
* The Business Model Layer serves as a data model for the application, exposing only the properties necessary for other layers to use.

**Business Logic Layer:**

* The Business Logic Layer contains all the application's core logic and functionality.
* It handles use cases and business rules, orchestrating interactions between the Data Access Layer and Business Model Layer.
* This layer contains methods for creating new Harvests, registering users, and other application-specific operations.
* It also includes a CryptHelper class for encrypting and decrypting passwords, enhancing security for user authentication and storage.

**REST API Layer:**

* The REST API Layer acts as an interface for clients to interact with the backend application.
* It includes controllers for different resources like AuthController, HarvestsController, OrchardsController, and TimesheetsController.
* Each controller contains action methods that handle HTTP requests, process data, and return responses.
* The REST API Layer is responsible for accepting requests from clients, invoking the appropriate methods in the Business Logic Layer, and returning the results.

**Helper Folder:**

* The Helper folder contains two classes—one for unit tests and another for JWT (JSON Web Token) configuration.
* The unit test class is used to write and execute unit tests for different components of the application, ensuring their correctness and functionality.
* The JWT configuration class manages the configuration and setup of JWT for authentication and authorization in the application.

Frontend Design & Components:

The frontend project is built around three main folders, each responsible for different aspects of the application:

**Reducers:**

* This folder contains two slices of the Redux Toolkit: `authSlice.jsx` and `dataSlice.jsx`.
* The `authSlice` manages the state of the authentication functionality, such as user login, logout, and authentication status.
* The `dataSlice` manages the state of the data from the backend server to display on the frontend.
* These slices provide a clean and efficient way to manage the application's state, making it easy to track and modify data across components.

**Actions:**

* The Actions folder includes `authActions.jsx` and `dataActions.jsx`, which are asynchronous thunks for sending and receiving data between the frontend and backend server.
* `authActions` handles asynchronous actions related to authentication, such as logging in, signing up, and managing user tokens.
* `dataActions` deals with data-related asynchronous actions, such as fetching data from the server and updating the data state in Redux.
* These actions facilitate communication with the backend and ensure seamless data flow within the application.

**Components:**

The Components folder includes various reusable components that form the frontend user interface:

* **Auth**: This component presents a sign-in and sign-up dialog screen. It ensures that users must log in or register successfully before accessing the website's main content.
* **Chart**: A customized component using the Recharts library to display pie charts on the screen.
* **ChartLabel**: Responsible for showing the correct labels on each pie slice in the chart.
* **ChartTooltip**: Manages the display of tooltips on each pie slice, providing relevant information to the user.
* **CustomTabs**: A customized component using the Material-UI library, enabling users to select different data formats to display on charts.
* **Filter**: A section on the main page that allows users to filter data using date ranges and orchards, enhancing data exploration and visualization.
* **Header**: The header component displays the Hectre logo and includes a logout button for users to log out from the website. It clears user cookies and tokens upon logout.
* **Helper**: This component includes various calculating functions that help in grouping data based on different filters and requirements.
* **LegendBox**: Sets the legend area of the charts, making it easy for users to understand the data represented in the chart.
* **Main**: The main section of the webpage that includes the charts, legend, tabs, and data. It presents the primary content of the website and displays relevant information to the user.

website design (Frontend)

