**Overall Layout**

**ASP.NET Core API (Backend)**:

* Acts as the single point of interaction with the MySQL database.
* Uses **Entity Framework Core (EF)** for database operations.
* Implements business logic and exposes RESTful endpoints for web and desktop apps.

**Frontend (React + TypeScript)**:

* Makes HTTP requests to the API for data.
* Displays data to users and interacts with the API for CRUD operations.

**Desktop App (WinForms)**:

* Communicates with the API via HTTP requests.
* Focuses on UI and sending/receiving data from the centralized backend.

**Database**:

* MySQL database serves as the backend data store.
* Accessed exclusively by the ASP.NET API.

**Other Integrations:**

* Password hashing?
* HTTPS?
* JWT authentication to ensure only authorized clients can access the API?

**ASP.NET Core API**

1. **Endpoints**:
   * Each feature will have its respective controller (e.g., ProductsController, UsersController).
   * Attribute routing defines clear paths for all operations.
2. **EF Core Integration**:
   * Use EF Core for database interactions with models and DbContext.
   * Example: CRUD operations on products.
3. **Authentication**:
   * Implement JWT authentication for secure API access????
   * Desktop and web apps will send a JWT token with each request????

**Frontend: React + TypeScript**

1. **Data Fetching**:
   * Use libraries like axios or the Fetch API to call the REST API.
   * Handle JWT tokens for authenticated requests???
2. **State Management**:
   * Manage app state with Context API or libraries like Redux?????

**Desktop App: WinForms**

1. **HTTP Communication**:
   * Use HttpClient to make API requests.
   * Send data or fetch updates from the API and display them in the WinForms UI.
2. **Authentication**:
   * Store and include JWT tokens in API requests?????

A diagram of a model

Description automatically generated**Approach**

1. **User**: The client (either web or desktop) sends a request (e.g., login, data query) to the API.
2. **Router (Controller)**: The ASP.NET controller receives the request and determines the appropriate action. The controller serves as both the "router" and the "controller" in this case.
3. **Model (EF Core)**: The controller uses **Entity Framework (EF Core)** to interact with the database models, querying or updating data as needed.
4. **Database**: EF Core queries the MySQL database and returns the result to the controller.
5. **Model**: The controller processes the data returned from the database and maps it back into a format the client can understand (e.g., JSON).
6. **Router (Controller)**: The controller sends the processed data back to the client as a response.
7. **User**: The client receives the data and handles it, typically displaying it in the UI.