Here’s a step-by-step plan to build and demo a full-stack financial dashboard using React + TypeScript (frontend) and SpringFramework.NET (C# backend) with SQL Server as the database, as per your requirements:

**1. Project Setup**

**Backend (SpringFramework.NET, C#)**

1. Create a new .NET Web API project (C#).
2. Add SpringFramework.NET for dependency injection, AOP, etc.
3. Add Entity Framework Core (EF Core) for ORM and SQL Server integration.
4. Configure connection string for the Elfshire SQL Server database.
5. Define data models (e.g., User, Tenant, FinancialRecord, Report).
6. Set up EF Core migrations and seed initial data.

**Frontend (React + TypeScript)**

1. Create a new React app with TypeScript template.
2. Install libraries:
   * Chart.js or D3.js for charts
   * Material-UI or Ant Design for data grids
   * Axios or Fetch for API calls
   * Libraries for Excel/PDF export (e.g., xlsx, jspdf)
3. Set up routing and authentication (e.g., JWT).

**2. Backend Implementation**

1. **User Authentication & Multi-Tenancy**
   * Implement JWT-based authentication.
   * Add user roles (admin, user, etc.) and tenant support.
2. **RESTful APIs**
   * Endpoints for:
     + User management
     + Financial data (CRUD)
     + Reports (with drill-down support)
     + Data for charts and grids
     + Audit logs
3. **Data Aggregation & Scheduled Jobs**
   * Use a scheduler (e.g., Quartz.NET) for periodic data aggregation.
4. **External API Integration**
   * Create services to fetch stock/currency data and store in DB.
5. **Validation, Error Handling, Logging**
   * Use SpringFramework.NET features for validation and AOP for logging.

**3. Frontend Implementation**

1. **Login Page**
   * Secure login, JWT token storage, role-based UI.
2. **Dashboard**
   * Fetch and display financial data from backend.
   * Dynamic charts (Chart.js/D3.js) for visualizations.
   * Data grids for tabular data.
   * Export buttons for Excel/PDF.
3. **User Customization**
   * Allow users to customize dashboard layout and save preferences.
4. **Drill-Down Reports**
   * Click on chart/grid items to view detailed reports.
5. **Multi-Tenant Support**
   * Show/hide data based on user’s tenant.

**4. Data Management**

* **Use Entity Framework Core** for data access and migrations.
  + It’s the most common, robust, and maintainable way to interact with SQL Server in .NET.
  + SpringFramework.NET can be used for DI, AOP, and service wiring, but EF Core is best for ORM.
* **Data Loading:**
  + Use EF Core’s migrations and seeders for initial data.
  + For ongoing data entry, build admin UI or use SQL scripts.

**5. Demo Integration in data/demo/DemoTsx**

1. Create a dashboard page in DemoTsx.
2. Fetch all data (charts, grids, user info) from the backend via REST API.
3. Render charts, grids, and export buttons.
4. Implement login and role-based UI.
5. Add drill-down and customization features.
6. Show code snippets or comments highlighting advanced TypeScript usage.

**6. Summary Table**

| **Layer** | **Technology** | **Key Features/Tasks** |
| --- | --- | --- |
| Frontend | React + TypeScript | Charts, grids, export, login, customization |
| Backend | .NET + Spring.NET | REST API, auth, jobs, validation, logging |
| Data | SQL Server + EF Core | Data models, migrations, seeding, queries |

**7. Impress Interviewers By:**

* Explaining your architecture and tech choices.
* Showing advanced TypeScript (types, generics, API contracts).
* Demonstrating secure, scalable backend with Spring.NET and EF Core.
* Highlighting real-time data, external API integration, and multi-tenancy.