🛠️ Section 1: Project Overview

1️⃣ Can you briefly describe your project and its key features?

This is a full-stack Expense Tracker that allows users to manage their finances by tracking transactions, setting budgets, and categorizing expenses. It includes authentication, role-based access control, data visualization, and a calendar-based transaction overview.

Key Features:
✅ User Authentication & Role-Based Access
✅ CRUD Operations for Transactions, Budgets, and Categories
✅ Filtering, Sorting, and Pagination
✅ Data Visualization with Pie Charts
✅ Calendar-Based Transaction Tracking

2️⃣ What technologies did you use in this project and why?

- **Frontend:** React.js (for component-based UI and routing)
- **Backend:** Node.js with Express.js (for handling API requests)
- **Database:** PostgreSQL / MongoDB (for structured transaction data)
- **Authentication:** JWT (for secure user sessions)
- **Styling:** Bootstrap, CSS
- **State Management:** React Hooks (`useState` , `useEffect`)

Why these technologies?

- React provides a **fast and dynamic UI** with reusable components.
- Node.js is **non-blocking**, making API requests efficient.
- JWT ensures **secure authentication** without storing sensitive data in cookies.

3️⃣ What was the biggest challenge you faced while building this project, and how did you solve it?

Challenge: Handling **large transaction datasets** efficiently.
Solution: Implemented **pagination** (`slice()`) and **local filtering** (`useState`) to reduce unnecessary API calls.

4️⃣ If you had more time, what improvements would you make to this project?

✅ **Implement Redux** for global state management.
✅ **Enhance security** by using HttpOnly cookies instead of `sessionStorage`.
✅ **Add real-time updates** using WebSockets for instant transaction tracking.
✅ **Integrate bank APIs** to fetch transactions automatically.

🚀 Section 2: React & Component Optimization

5️⃣ How does React handle rendering, and how do you prevent unnecessary re-renders?

React re-renders a component when **state or props change**.

To prevent unnecessary re-renders:
✅ Use `React.memo()` for functional components.
✅ Use `useCallback()` to memoize functions.
✅ Use `useMemo()` to optimize expensive computations.
✅ Ensure correct dependency arrays in `useEffect()`.

6️⃣ How would you optimize performance in your project's components?

✅ **Lazy loading** routes to prevent loading all pages at once.
✅ **Debounce search inputs** to minimize API requests.
✅ **Optimize API calls** with caching and pagination.
✅ **Use virtualized lists** for large transaction tables.

7️⃣ What is the difference between `useState` and `useReducer`, and where would you use `useReducer`?

- `useState` is **simple** and works well for independent states.
- `useReducer` is better for **complex state logic** (e.g., managing multiple filters in `TransactionFilter.jsx`).

8️⃣ Why did you choose React instead of other frontend frameworks like Angular or Vue?

✅ React has a **strong community** and easy integration with libraries.
✅ It supports **component reusability** and **virtual DOM** for better performance.
✅ **Compared to Angular**, React is **lighter** and easier to learn.

🌍 Section 3: Routing & Navigation

9️⃣ How does your project handle route management?

- **React Router (`routes.jsx`)** defines different pages.
- `ProtectedRoute.jsx` ensures only logged-in users access certain pages.

🔟 What is the role of `ProtectedRoute.jsx`, and how does it ensure security?

✅ Checks if the **JWT token exists** in `sessionStorage`.
✅ Redirects users to `/login` if they are **unauthorized**.
✅ Restricts admin-only pages (`Categories.jsx`, `Users.jsx`).

🔐 Section 4: Authentication & Authorization

1️⃣2️⃣ How does authentication work in your project?

1️⃣ User logs in (`/auth/login`).
2️⃣ Backend returns a **JWT token**.
3️⃣ Token is stored in `sessionStorage` and added to every request.

1️⃣3️⃣ Where do you store the authentication token, and why?

- Stored in `sessionStorage` so it clears when the browser closes.
- More secure alternative: **HttpOnly cookies** (prevents XSS attacks).

1️⃣4️⃣ How does role-based access control work in your application?

- User **roles (USER, ADMIN)** are stored in the database.
- Admin-only routes (`Users.jsx` , `Categories.jsx`) are protected via `ProtectedRoute.jsx` .

📊 Section 5: Transactions & Budget Management

1️⃣6️⃣ How are transactions stored and managed in your project?

- Transactions are **fetched from the API (`/expenses/get/all`)** and stored in `useState` .
- Users can **filter, add, edit, delete transactions** (`POST` , `PUT` , `DELETE`).

1️⃣7️⃣ How does filtering work in `TransactionFilter.jsx`?

- Uses `useState` to track filter inputs.
- **Filters transactions locally** to reduce API calls.

1️⃣8️⃣ How does the budget system work, and how do you ensure accurate calculations?

- Users **set budgets per category**.
- Budget tracking compares **total expenses vs. budget limit**.

🛠️ Section 6: API Handling & Backend Communication

2️⃣0️⃣ How does your project handle API requests?

- Uses `api.js` to create an **Axios instance** with a base URL.
- **JWT token is attached automatically** to every request.

2️⃣3️⃣ How do you handle errors in API requests, and how do you display error messages?

- **Try-Catch blocks** handle API failures.
- `toast.error()` is used to show user-friendly error messages.

⚡ Section 7: Performance Optimization

2️⃣5️⃣ How does pagination improve performance, and where is it implemented in your project?

- **Only loads 5-10 transactions per page** to reduce memory usage.
- Implemented in `TransactionList.jsx` and `UserList.jsx`.

🛡️ Section 8: Security Best Practices

2️⃣9️⃣ How would you prevent Cross-Site Scripting (XSS) in your project?

- Sanitize user inputs before rendering (`DOMPurify`).
- Avoid `dangerouslySetInnerHTML` .

3️⃣0️⃣ How would you prevent Cross-Site Request Forgery (CSRF)?

- Use **CSRF tokens** or **SameSite cookies**.

🚀 Section 1️⃣0️⃣: Deployment & Scalability

3️⃣5️⃣ How would you deploy this project, and what hosting services would you use?

- **Frontend** → Vercel or Netlify
- **Backend** → Heroku or AWS
- **Database** → PostgreSQL on AWS RDS

✅ Final Step: Behavioral & Conceptual Questions

3️⃣9️⃣ Can you explain a technical decision you made in this project and why?

- **Used pagination for transactions** to reduce API load instead of fetching all data at once.

4️⃣0️⃣ What is one thing you learned while working on this project?

- Learned how to **implement secure role-based access control** with JWT.