**Day 1: Project Initialization**

**1. Initialize Repository:**

* **Action: Create a new Git repository on GitHub or another platform.**
* **Folder Structure:**
  + **frontend/ - For React app.**
  + **backend/ - For Spring Boot app.**
  + **docs/ - For any documentation.**
* **Best Practice: Add a .gitignore file to ignore unnecessary files (e.g., node\_modules/, target/).**

**2. Set Up Frontend (React):**

* **Action: Use create-react-app to initialize the frontend.**
  + **npx create-react-app cyber-craft-frontend**
* **Dependencies: Install essential libraries:**
  + **react-router-dom for routing.**
  + **axios for API requests.**
  + **redux (optional) for state management.**
* **Folder Structure:**
  + **src/components/ - Reusable components.**
  + **src/pages/ - Pages like Login, Dashboard.**
  + **src/services/ - API calls.**
  + **src/store/ - Redux setup (if using).**

**3. Set Up Backend (Spring Boot):**

* **Action: Use Spring Initializr to set up the project with dependencies like:**
  + **Spring Web**
  + **Spring Data JPA**
  + **Spring Security**
  + **MySQL or H2 Database**
* **Folder Structure:**
  + **src/main/java/com/cybercraft/ - Core application files.**
  + **src/main/resources/ - Configuration files.**
  + **src/main/resources/static/ - Static resources (if needed).**

**4. Push Initial Commit:**

* **Action: Push the initial codebase to your Git repository.**
* **Best Practice: Write a clear commit message like "Initial project setup with React and Spring Boot."**

**Day 2-3: Implement Registration/Login (T1)**

**Frontend:**

* **Create Forms:**
  + **Design registration and login forms in React.**
  + **Use controlled components to handle form data.**
* **Form Validation:**
  + **Implement basic validation (e.g., email format, password strength) using a library like formik or custom logic.**

**Backend:**

* **User Entity:**
  + **Create a User entity with fields like id, username, password, email, roles.**
* **Repositories & Services:**
  + **Implement a UserRepository for database operations.**
  + **Create a UserService to handle business logic.**
* **Security Configuration:**
  + **Use Spring Security to handle authentication and password encryption (BCrypt).**
  + **Set up JWT for token-based authentication.**

**Integration:**

* **API Calls:**
  + **Use axios in React to send registration and login requests to the backend.**
* **State Management:**
  + **Store JWT tokens in localStorage or use Redux for managing the authenticated state.**

**Testing:**

* **Frontend: Test form submission and error handling.**
* **Backend: Test API endpoints using Postman or a similar tool.**

**Push Changes:**

* **Push the completed registration/login feature to your repository.**

**Day 4-5: Dashboard (T2)**

**Frontend:**

* **Layout:**
  + **Design the dashboard using a UI framework like Material-UI or Bootstrap.**
  + **Include a sidebar, header, and main content area.**
* **Components:**
  + **Create components to display user information, recent orders, and quick links.**

**Backend:**

* **Endpoints:**
  + **Implement endpoints to fetch user-specific data (e.g., /api/dashboard).**
* **Services:**
  + **Create services to aggregate data for the dashboard (e.g., recent orders, notifications).**

**Integration:**

* **API Integration:**
  + **Use axios to fetch dashboard data from the backend and display it in the React components.**

**Testing:**

* **Frontend: Ensure all components render correctly and data is fetched successfully.**
* **Backend: Test the dashboard API to ensure it returns correct and timely data.**

**Push Changes:**

* **Push the completed dashboard feature to your repository.**

**Day 6-7: Prebuilt PCs (T3)**

**Frontend:**

* **Page Design:**
  + **Create a page to display a list of prebuilt PCs with images, names, and prices.**
* **Reusable Components:**
  + **Develop reusable components like ProductCard for displaying PC details.**

**Backend:**

* **Product Entity:**
  + **Define a Product entity with fields like id, name, description, price, image, and category.**
* **Repositories & Services:**
  + **Implement a ProductRepository and ProductService to manage product data.**
* **Endpoints:**
  + **Create an endpoint (e.g., /api/products/prebuilt) to fetch prebuilt PCs.**

**Integration:**

* **API Calls:**
  + **Use axios to fetch the list of prebuilt PCs from the backend.**
* **UI Rendering:**
  + **Map through the product data to render ProductCard components dynamically.**

**Testing:**

* **Frontend: Verify that products load correctly and are displayed in the intended format.**
* **Backend: Test the product fetching API with different scenarios (e.g., empty list, multiple products).**

**Push Changes:**

* **Push the completed Prebuilt PCs feature to your repository.**

**Day 8: Search Bar (T4)**

**Frontend:**

* **Component Creation:**
  + **Implement a SearchBar component.**
  + **Capture user input and trigger search queries.**
* **Live Search (Optional):**
  + **Implement live search suggestions if time permits.**

**Backend:**

* **Search Logic:**
  + **Add search functionality to your product repository/service to filter products based on search terms.**
* **Endpoints:**
  + **Create an endpoint (e.g., /api/products/search) to handle search queries.**

**Integration:**

* **API Integration:**
  + **Use axios to send search queries to the backend and display the results in the frontend.**

**Testing:**

* **Frontend: Test the search functionality with various inputs (e.g., valid, invalid, partial matches).**
* **Backend: Ensure the search endpoint handles different scenarios and returns accurate results.**

**Push Changes:**

* **Push the completed search bar feature to your repository.**

**Day 9: Product Filter (T5)**

**Frontend:**

* **Filter UI:**
  + **Create filter options (e.g., dropdowns, checkboxes) for categories like price range, brand, and specifications.**
* **Filter Logic:**
  + **Implement logic to capture selected filters and trigger filtered queries.**

**Backend:**

* **Filtering Logic:**
  + **Extend your product repository/service to apply multiple filters based on frontend input.**
* **Endpoints:**
  + **Modify existing endpoints to support filtering or create new ones.**

**Integration:**

* **API Integration:**
  + **Use axios to send filter parameters to the backend and update the displayed products accordingly.**

**Testing:**

* **Frontend: Test filter combinations to ensure the UI updates correctly.**
* **Backend: Validate that the filtering logic returns accurate and expected results.**

**Push Changes:**

* **Push the completed product filter feature to your repository.**

**Day 10-11: Product Description (T6)**

**Frontend:**

* **Product Page:**
  + **Create a detailed product description page that includes images, specifications, customer reviews, and related products.**
* **UI Enhancements:**
  + **Use image sliders or tabs to enhance the UI.**

**Backend:**

* **Product Details Endpoint:**
  + **Create an endpoint (e.g., /api/products/{id}) to fetch detailed information about a specific product.**

**Integration:**

* **API Calls:**
  + **Use axios to fetch product details based on the product ID and render them on the product description page.**

**Testing:**

* **Frontend: Ensure the product description page displays all relevant information correctly.**
* **Backend: Test the product details endpoint with different product IDs to verify correct data retrieval.**

**Push Changes:**

* **Push the completed product description feature to your repository.**

**Day 12-14: Add to Cart (T7)**

**Frontend:**

* **Add to Cart Button:**
  + **Implement an "Add to Cart" button on the product description page.**
* **Shopping Cart UI:**
  + **Design a shopping cart page to display selected items, quantities, and total price.**

**Backend:**

* **Cart Entity:**
  + **Define a Cart entity to track items added to the cart.**
* **Repositories & Services:**
  + **Implement CartRepository and CartService to manage cart operations.**
* **Endpoints:**
  + **Create endpoints to add, update, and remove items from the cart (e.g., /api/cart).**

**Integration:**

* **API Integration:**
  + **Use axios to interact with the cart endpoints for adding/removing items and updating quantities.**
* **State Management:**
  + **Consider using Redux or React Context to manage cart state across the application.**

**Testing:**

* **Frontend: Verify that the cart updates correctly when items are added, removed, or quantities are changed.**
* **Backend: Test cart operations (add, update, delete) to ensure accuracy and persistence.**

**Push Changes:**

* **Push the completed add-to-cart feature to your repository.**

**Day 15-16: Custom PC Configurator (T8)**

**Frontend:**

* **Configurator UI:**
  + **Design an interface where users can select components (e.g., CPU, GPU, RAM) to build a custom PC.**
* **Validation:**
  + **Implement validation logic to ensure compatible components are selected.**

**Backend:**

* **Component Entity:**
  + **Define entities for components (e.g., CPU, GPU, Motherboard).**
* **Repositories & Services:**
  + **Implement repositories and services for fetching and validating components.**
* **Endpoints:**
  + **Create endpoints to fetch available components and validate custom PC configurations.**

**Integration:**

* **API Integration:**
  + **Use axios to fetch component options and validate the selected configuration against backend rules.**

**Testing:**

* **Frontend: Test the configurator for usability and validation accuracy.**
* **Backend: Ensure that component validation logic works as expected and handles all edge cases.**

**Push Changes:**

* **Push the completed custom PC configurator feature to your repository.**

**Day 17-18: Checkout Gateway (T9)**

**Frontend:**

* **Checkout Page:**
  + **Design the checkout page with order summary, payment options, and delivery information.**
* **Payment UI:**
  + **Integrate payment gateways (e.g., Stripe, PayPal) or implement a mock payment flow.**

**Backend:**

* **Order Entity:**
  + **Define an Order entity to capture details of the purchase.**
* **Repositories & Services:**
  + **Implement OrderRepository and OrderService to manage order creation and payment processing.**
* **Endpoints:**
  + **Create endpoints to submit orders and process payments (e.g., /api/orders).**

**Integration:**

* **API Integration:**
  + **Use axios to send order details to the backend and handle payment processing.**

**Testing:**

* **Frontend: Ensure the checkout flow is smooth, with proper error handling for payment failures.**
* **Backend: Test order creation and payment processing with different scenarios.**

**Push Changes:**

* **Push the completed checkout gateway feature to your repository.**

**Day 19: Shipping Options (T10)**

**Frontend:**

* **Shipping Selection UI:**
  + **Add UI components to select shipping methods (e.g., standard, express) during checkout.**

**Backend:**

* **Shipping Logic:**
  + **Implement logic to fetch and manage available shipping options.**
* **Endpoints:**
  + **Create an endpoint to fetch shipping options based on user location or cart details (e.g., /api/shipping-options).**

**Integration:**

* **API Integration:**
  + **Fetch available shipping options and update the order summary with the selected option.**

**Testing:**

* **Frontend: Verify that shipping options are displayed correctly and the selection affects the order total.**
* **Backend: Ensure that the shipping options endpoint returns accurate and context-specific data.**

**Push Changes:**

* **Push the completed shipping options feature to your repository.**

**Day 20: Order History (T11)**

**Frontend:**

* **Order History Page:**
  + **Design a page that lists past orders with details like order date, status, and total amount.**

**Backend:**

* **Order Fetching Logic:**
  + **Implement logic to retrieve a user's order history.**
* **Endpoints:**
  + **Create an endpoint to fetch a user's order history (e.g., /api/orders/history).**

**Integration:**

* **API Integration:**
  + **Use axios to fetch order history data and display it in a structured format.**

**Testing:**

* **Frontend: Ensure the order history page displays orders correctly and allows users to view details.**
* **Backend: Test the order history endpoint to verify that it returns the correct data for different users.**

**Push Changes:**

* **Push the completed order history feature to your repository.**

**Day 21: FAQ (T12)**

**Frontend:**

* **FAQ Page:**
  + **Create a static or dynamic FAQ page that answers common user questions.**
* **Accordion UI:**
  + **Use an accordion or expandable UI to organize FAQ content.**

**Backend:**

* **FAQ Management (Optional):**
  + **Implement a simple CMS or API to manage FAQs dynamically if needed.**
* **Endpoints (Optional):**
  + **Create an endpoint to fetch FAQ data (e.g., /api/faq).**

**Integration:**

* **Static/Dynamic Content:**
  + **Depending on your implementation, either hardcode the FAQ data in React or fetch it from the backend.**

**Testing:**

* **Frontend: Ensure the FAQ page is user-friendly and answers are easy to find.**
* **Backend: If dynamic, test the FAQ API for correct data retrieval.**

**Push Changes:**

* **Push the completed FAQ feature to your repository.**

**Day 22: Customer Rating (T13)**

**Frontend:**

* **Rating UI:**
  + **Add a rating component to product pages, allowing users to rate and review products.**

**Backend:**

* **Rating Entity:**
  + **Define a Rating entity with fields like userId, productId, rating, and review.**
* **Repositories & Services:**
  + **Implement RatingRepository and RatingService to handle rating submissions and fetching.**
* **Endpoints:**
  + **Create endpoints to submit and fetch ratings (e.g., /api/ratings).**

**Integration:**

* **API Integration:**
  + **Use axios to submit user ratings and fetch average ratings for products.**

**Testing:**

* **Frontend: Ensure the rating system works smoothly, with accurate star display and review submission.**
* **Backend: Test rating submission and retrieval to ensure correct functionality.**

**Push Changes:**

* **Push the completed customer rating feature to your repository.**

**Day 23: Notifications (T14)**

**Frontend:**

* **Notification System:**
  + **Implement a system to display real-time notifications (e.g., order updates, promotions) using a library like react-toastify.**

**Backend:**

* **Notification Logic:**
  + **Implement services to trigger notifications based on user actions (e.g., order status change).**
* **WebSocket (Optional):**
  + **Set up WebSocket or server-sent events (SSE) for real-time notifications.**

**Integration:**

* **Real-time Updates:**
  + **Use WebSocket/SSE or poll the backend periodically to fetch and display notifications.**

**Testing:**

* **Frontend: Ensure notifications are displayed correctly and are user-friendly.**
* **Backend: Test notification triggers and ensure they are sent at the correct times.**

**Push Changes:**

* **Push the completed notifications feature to your repository.**

**Day 24-25: Final Testing and Deployment**

**Final Testing:**

* **End-to-End Testing:**
  + **Test the entire application for any bugs or issues. Make sure all functionalities work together seamlessly.**
* **Cross-Browser Testing:**
  + **Test the application across different browsers to ensure compatibility.**
* **Performance Testing:**
  + **Use tools like Google Lighthouse to analyze and improve performance.**

**Deployment:**

* **Backend Deployment:**
  + **Deploy the Spring Boot backend to a cloud service (e.g., AWS, Heroku).**
* **Frontend Deployment:**
  + **Deploy the React frontend to a hosting service (e.g., Netlify, Vercel).**
* **Environment Configuration:**
  + **Set up environment variables for production (e.g., API URLs, keys).**
* **CI/CD (Optional):**
  + **Set up continuous integration/continuous deployment (CI/CD) pipelines for automated builds and deployments.**

**Push Final Changes:**

* **Push the final, tested code to your repository with a clear commit message like "Final build and deployment."**