Industrix Full Stack Engineer Intern - Coding Challenge

Overview

Welcome to the Industrix coding challenge! You'll be building a **Full Stack Todo List Application** that demonstrates the core technologies and practices we use in our development workflow.

Timeline: 3 days from receiving this challenge

Submission: Please submit via GitHub repository link

Project Requirements

The Challenge

Build a full-stack todo list web application with the following features:

Core Features

1. **Todo Management**
    - Create new todo items with title and description
    - Mark todos as completed/incomplete
    - Edit existing todo items
    - Delete todo items
    - View all todos in a clean, organized list
2. **Todo Categories**
    - Assign categories to todos (Work, Personal, Shopping, etc.)
    - Create and manage custom categories (basic CRUD)
3. **Todo List Management**
    - **Basic pagination:** Show 10-20 items per page with simple pagination controls
    - **Simple search:** Search todos by title
    - View all todos in a clean, organized list

Technical Requirements

Required

- **Frontend:** React application with modern hooks and state management
    - Use React Context API for state management (or show understanding in extra credit)
    - **Use Ant Design (antd) as the UI framework**
    - **Responsive design:** Application must work on desktop, tablet, and mobile devices
    - Clean, intuitive user interface
- **Backend:** RESTful API service
    - Choose your preferred language (Go preferred, but Python/Node.js acceptable)
    - PostgreSQL database integration
    - **Encourage using an ORM** (GORM for Go, SQLAlchemy for Python, Sequelize for Node.js)
    - **Database migrations in SQL files** (up and down migrations)
    - Proper error handling and validation
- **Documentation:**
    - README with setup instructions
    - API documentation (endpoints, request/response formats)
    - Brief explanation of your technical decisions

Good to Have (Bonus Points)

- **Backend Unit Tests:** Comprehensive unit tests for backend services (+10 points - highest value)
- **React Context API:** Demonstrate proper use of React Context for state management (+6 points)
- **Advanced Filtering:** Filter by completion status, category, priority (+5 points)
- **Docker:** Containerized backend and database with docker-compose file (+3 points)
- **TypeScript:** Use TypeScript for frontend development (+2 points)

Sample Data Structure

Todo Object

json

```
{ "id": 1, "title": "Complete coding challenge", "description": "Build a full-stack todo application for Industrix", "completed": false, "category_id": 2, "priority": "high", "due_date": "2024-08-03T23:59:59Z", "created_at": "2024-07-31T10:00:00Z", "updated_at": "2024-07-31T10:00:00Z" }
```

Category Object

json

```
{ "id": 1, "name": "Work", "color": "#3B82F6", "created_at": "2024-07-31T10:00:00Z" }
```

Priority Levels

- `high` (red indicator)
- `medium` (yellow indicator)
- `low` (green indicator)

API Endpoints (Suggested)

`# Todos GET /api/todos # List todos with pagination and optional filters # Query params: page, limit, search, sort_by, sort_order POST /api/todos # Create new todo GET /api/todos/:id # Get specific todo PUT /api/todos/:id # Update todo DELETE /api/todos/:id # Delete todo PATCH /api/todos/:id/complete # Toggle completion status

Categories

GET /api/categories # List all categories POST /api/categories # Create new category PUT /api/categories/:id # Update category DELETE /api/categories/:id # Delete category`

Example API Response with Pagination

json

```
{ "data": [ { "id": 1, "title": "Complete coding challenge", "description": "Build a full-stack todo application", "completed": false, "category": { "id": 2, "name": "Work", "color": "#3B82F6" }, "created_at": "2024-07-31T10:00:00Z", "updated_at": "2024-07-31T10:00:00Z" } ], "pagination": { "current_page": 1, "per_page": 10, "total": 25, "total_pages": 3 } }
```

Evaluation Criteria (100 Points Total)

Core Functionality (45 points)

- **App runs successfully** following README instructions (15 points)
- CRUD operations work (create, read, update, delete todos) (12 points)
- Categories functionality works (8 points)
- Basic pagination works (5 points)
- Search functionality works (5 points)

Code Quality & Structure (25 points)

- Clean, readable code structure (15 points)
- Proper error handling (10 points)

Frontend Implementation (20 points)

- Proper React component structure (8 points)
- Effective use of Ant Design components (7 points)
- Responsive design (works on mobile/desktop) (5 points)

Documentation (10 points)

- **Clear, accurate setup instructions** - Write as if for a team member who has never seen the project (5 points)
- Complete answers to technical questions (5 points)

Extra Credit Bonus Points (up to 26 additional points)

- **Backend Unit Tests:** Comprehensive unit tests for backend services (+10 points - highest value)
- **React Context API implementation** (+6 points)
- **Advanced filtering system** (status, category, priority filters) (+5 points)
- **Docker containerization** (backend + database) (+3 points)
- **TypeScript usage** (+2 points)

Maximum possible score: 126 points (100 base + 26 bonus)

Passing threshold: 70+ points

Submission Guidelines

1. **GitHub Repository:**
    - Create a public GitHub repository
    - Include all source code and documentation
    - Provide the repository URL in your submission
2. **README Requirements:**
    - Project overview and features implemented
    - **Step-by-step setup and installation instructions** - Write as if a new team member with no context is trying to run your project
    - How to run the application locally
    - How to run tests (if implemented)
    - API documentation
    - **Answer the technical questions below** (mandatory)
    - Screenshots or demo (optional but appreciated)
3. **What to Include:**
    - Complete source code
    - **Database migrations (SQL files with up and down migrations)**
    - Docker configuration (if implemented)
    - Unit tests (if implemented for bonus points)
    - **Accurate setup instructions that work**
    - Answers to technical questions in README

Required Technical Questions

Please answer these questions in your README.md file:

Database Design Questions

1. **What database tables did you create and why?**
    - Describe each table and its purpose
    - Explain the relationships between tables
    - Why did you choose this structure?
2. **How did you handle pagination and filtering in the database?**
    - What queries did you write for filtering and sorting?
    - How do you handle pagination efficiently?
    - What indexes (if any) did you add and why?

Technical Decision Questions

1. **How did you implement responsive design?**
    - What breakpoints did you use and why?
    - How does the UI adapt on different screen sizes?
    - Which Ant Design components helped with responsiveness?
2. **How did you structure your React components?**
    - Explain your component hierarchy
    - How did you manage state between components?
    - How did you handle the filtering and pagination state?
3. **What backend architecture did you choose and why?**
    - How did you organize your API routes?
    - How did you structure your code (controllers, services, etc.)?
    - What error handling approach did you implement?
4. **How did you handle data validation?**
    - Where do you validate data (frontend, backend, or both)?
    - What validation rules did you implement?
    - Why did you choose this approach?

Testing & Quality Questions

1. **What did you choose to unit test and why?**
    - Which functions/methods have tests?
    - What edge cases did you consider?
    - How did you structure your tests?
2. **If you had more time, what would you improve or add?**
    - What technical debt would you address?
    - What features would you add?
    - What would you refactor?

Sample UI Flow

1. **Main Page:**
    - **Desktop:** Main content area with todo list and pagination
    - **Tablet/Mobile:** Responsive layout that adapts to smaller screens
2. **Todo List:**
    - Display todos in a clean, organized format
    - Use Ant Design components (Table, Cards, or List)
    - Include pagination at the bottom
3. **Todo Form:** Modal or drawer for creating/editing todos
4. **Search:** Simple search input to find todos by title/description

Notes

- **AI/LLM Usage Encouraged:** Feel free to use AI tools (ChatGPT, Claude, Copilot, etc.) to help with coding, but make sure to review and understand the final code you submit
- **No authentication required** - focus on the core todo functionality
- **Prioritize working features over perfect styling** - we value functionality more than visual polish
- **Partial implementations acceptable** - if you can't complete everything, document what's missing and explain why (time constraints, technical challenges, etc.)
- **Keep sample data simple** - 5-10 example todos and 3-4 categories are sufficient
- **Ask questions if needed** - feel free to email us if anything is unclear

Hints & Tips for Success

Getting Started (Hour 1)

- Start with the backend API - get basic CRUD working first
- **Create database migrations as SQL files** (001_create_todos.up.sql, 001_create_todos.down.sql, etc.)
- **Use migration tools** like `golang-migrate/migrate` for Go, `Alembic` for Python, or `Sequelize CLI` for Node.js
- Use a simple database structure (2-3 tables maximum)
- **Consider using an ORM** (GORM for Go, SQLAlchemy for Python, Sequelize for Node.js) - it will save you time
- Test your API endpoints with Postman or similar before building frontend

Frontend Development (Hours 2-4)

- Begin with basic React components - don't worry about state management initially
- Use Ant Design's Table or List components for displaying todos
- Get basic functionality working before adding search/pagination

Common Pitfalls to Avoid

- Don't spend too much time on perfect UI design - focus on functionality
- Don't over-engineer the database schema - keep it simple
- Don't try to implement all bonus features - pick 1-2 you're most confident with

Time Management

- **Hour 1-2:** Backend API + database setup
- **Hour 3-4:** Basic React frontend with CRUD operations
- **Hour 5-6:** Pagination + search functionality
- **Hour 7-8:** Polish, testing, documentation, bonus features

Scoring Strategy

- **Aim for 70+ points first** - ensure all core features work
- **Add bonus features only after core is solid** - don't risk breaking working features
- **Document everything clearly** - good README can earn you extra points

Technical Priorities

1. **Make it work** (45 points available)
2. **Clean code structure** (15 points available)
3. **Write documentation for team members** - assume they know nothing about your project (5 points - easy wins)
4. **Then add bonus features** - prioritize unit tests for maximum bonus (+10 points)

Questions?

If you have any questions about the requirements or need clarification, please don't hesitate to reach out. We're here to help!

Good luck, and we look forward to seeing your solution! 🚀

This challenge is designed to assess your full-stack development skills and should take approximately 4-8 hours to complete.