

Full Stack Application Java + React

Defining business requirements using AI

You can use the [BRD-ToDoApp.md](#) prompt to generate a Business Requirements Document (BRD) for a full-stack Java Spring Boot + React application. This document will help you outline the business context, goals, users, stakeholders, and core workflows for the To-Do application.

Generate Bussiness Requirement Document using below prompt:

You are a Business Analyst and Domain Architect.

I want to define clear, complete, structured business requirements for a new application.

Context:

- Build a full-stack Java Spring Boot + React application.
- The application must follow clean architecture, domain-driven concepts, and industry standards.
- Requirements should be user-centric, focusing on real-world business workflows.

Your task:

1. Ask me clarifying questions to fully understand the domain, actors, workflows, goals, and constraints.
2. Produce:
 - A problem statement
 - Primary actors & user personas
 - High-level features (epics)
 - Detailed functional requirements
 - Non-functional requirements (security, scalability, performance)
 - System assumptions & constraints
3. Present everything in a structured, professional document.

Begin by asking the most critical clarifying questions.

This prompt will guide the AI to create a comprehensive business requirements document that can be used as a foundation for developing the To-Do application.

Provide summary of bussiness requirements.

****To-Do App – Business & Functional Requirements****

****Context & Goals:****

Internal task-management tool to improve productivity, visibility, and accountability. Sponsored by IT/Operations to replace scattered manual tracking.

****Users:****

Employees, managers, and admins. No external users. Executives and compliance teams may consume reports.

****Core Domain & Workflows:****

The system manages **tasks**.

Workflow: create task → assign → track status/due date → reminders → completion → reporting.

Supports priority, overdue flagging, and simple automation.

****Data & Integrations:****

Stores task details (title, description, priority, status, due date, assignee, tags, comments).

Possible integrations: SSO, email/Slack notifications, calendar sync.

Reports: completed tasks, overdue tasks, productivity metrics.

Use H2 in memory database for the application.

****Rules & Compliance:****

Basic internal policies; role-based visibility; overdue auto-flag; managers can reassign tasks.

****Security & Roles:****

Auth via SSO or internal login.

User: manage own tasks.

Manager: manage team tasks + reports.

Admin: configure system.

****Non-Functional:****

Fast load (<2s), high availability (99.9%), cloud hosting, daily backups, responsive UI.

****UX & Devices:****

Simple, clean UX. Desktop-first but mobile responsive. English-only for MVP.

****Scope:****

MVP includes authentication, tasks CRUD, assignment, priorities/due dates, notifications, basic reporting.

Future: recurring tasks, calendar view, integrations, Kanban boards.

Action:

Create a BRD-ToDoApp.md file in the project root with the above content.

This will generate a `BRD-ToDoApp.md` file in the project root directory containing the summarized business requirements for the To-Do application.

Writing OpenAPI specifications with Cursor AI

You can use Cursor AI to help you write OpenAPI specifications for your full-stack Java Spring Boot + React application. Below is a sample prompt you can use to generate an OpenAPI specification for the To-Do application.

You are an API architect. Based on the business requirements provided, generate a clean, well-structured OpenAPI 3.0 specification.

Requirements:

- Include tags, summary, description, request bodies, responses, components, schemas.
- Use camelCase for JSON fields.
- Ensure proper response codes (200, 201, 400, 404, 500).
- Add pagination schema where applicable.
- Ensure error schema includes code, message, timestamp.
- Add sample requests/responses.
- Apply REST and resource naming best practices.

Output:

- A complete OpenAPI YAML document ready for Spring Boot, React, and Postman import.

Actions:

- Create a file named ``openapi.yaml`` in the project root with the generated specification.

This prompt will guide the AI to create a comprehensive OpenAPI specification that can be used for developing and documenting the API for the To-Do application.

Generating sequence, class & architecture diagrams

You can use AI tools to generate sequence diagrams, class diagrams, and architecture diagrams for your full-stack Java Spring Boot + React application. Below is a sample prompt you can use to generate these diagrams.

You are a Software Architect. Based on the provided OpenAPI specification and business requirements, generate the following diagrams:

1. **Sequence Diagrams**

- For each major API operation
- Include actors, frontend, backend layers, service, repository, DB

2. **Class Diagrams**

- Show entities, DTOs, services, controllers
- Include relationships (aggregation, composition, inheritance)

3. ****Architecture Diagram****

- Represent a full-stack Java Spring Boot + React application
- Show layers: UI, API Gateway (optional), Controllers, Services, Repos, DB
- Include external integrations if any
- Show communication flow

Format:

- Use Mermaid where applicable.
- Provide explanation under each diagram.

Output should be copy-paste ready.

Actions:

- Create a file named ``software-architecture.md`` in the project root with the generated diagrams and explanations.

This prompt will guide the AI to create detailed diagrams that illustrate the architecture and design of the To-Do application, which can be included in a `software-architecture.md` file in the project root.

Generating backend (Java Spring Boot) + frontend (React) scaffolding

You can use Cursor AI to help you generate the backend and frontend scaffolding for your full-stack Java Spring Boot + React application. Below is a sample prompt you can use to generate the necessary code structure.

Act as a Senior Full-Stack Architect.

Using the OpenAPI specification provided, generate full backend + frontend scaffolding.

Backend Requirements (Spring Boot):

- Generate folder structure (controller, service, repository, domain, dto, config, exception)
- Generate controller stubs matching OpenAPI routes
- Generate service interfaces + impl classes
- Generate entity + DTO classes
- Add validation annotations
- Create central exception handler
- Add sample unit test stubs

Frontend Requirements (React):

- Setup React with TypeScript
- Generate API service wrappers from OpenAPI
- Create pages, components, routing structure
- Add forms with validation
- Provide example container + presentational components
- Include basic UI for CRUD operations

Output:

- File structure

- Code stubs for each major file
- Commands to run backend & frontend
- What to implement next (TODO roadmap)

Actions:

- Create backend in `backend/` folder
- Create frontend in `frontend/` folder
- generate project using the above requirements

This prompt will guide the AI to create a comprehensive code structure for both the backend and frontend of the To-Do application, providing you with a solid foundation to build upon.

Generating Postman Collections from OpenAPI

You can use Cursor AI to help you generate Postman collections from your OpenAPI specification for your full-stack Java Spring Boot + React application. Below is a sample prompt you can use to generate the Postman collection.

You are an API Documentation Engineer.

Based on the OpenAPI 3.0 specification, generate a Postman Collection.

Requirements:

- Each API operation should map to a Postman request.
- Group requests using meaningful folders (based on tags or resources).
- Add environment variables for baseUrl, authToken, etc.
- Provide sample payloads for POST/PUT.
- Provide negative test cases as additional requests.
- Output Postman v2.1 JSON export format.

Finally:

- Include instructions on how to import into Postman.

Output:

- A Postman Collection JSON file named `todo-app-postman-collection.json`.

This prompt will guide the AI to create a Postman collection that can be imported into Postman for testing and documenting the API for the To-Do application.
