**Grocery To-Do (Shopping List) Application**

*SDLC Phase: Design Document*

Individual Assignment 1 – Software Development and DevOps

Student Name: Sofia Boicenco  
Course: Software Development and DevOps  
Date: 30.09.2025

**1. High-Level Design (HLD)**

**System Architecture**

The Grocery To-Do application follows a simple **three-layer architecture**:

* **Presentation Layer**: Minimal user interaction through FastAPI’s Swagger UI (and optional frontend later).
* **Application Layer**: FastAPI backend handling business logic and CRUD operations.
* **Data Layer**: SQLite database providing persistent storage.

**Main Modules**

* main.py – API entry point.
* models.py – defines Item entity.
* crud.py – CRUD logic.
* database.py – database setup.

**Interactions Diagram:**

+-----------+ +-------------------+ +---------------+

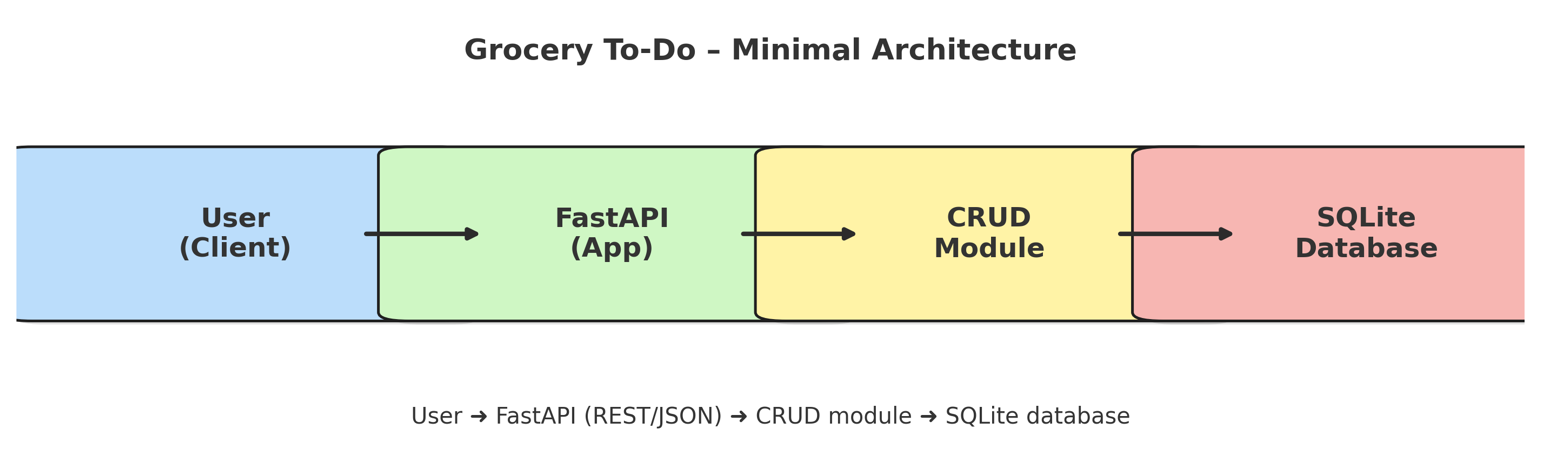
| User | ---> | FastAPI Backend | ---> | SQLite DB |

+-----------+ +-------------------+ +---------------+

^ |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

API Responses



**2. Low-Level Design (LLD)**

**Module Logic**

* crud.py: functions for create, read, update, delete, toggle purchased
* database.py: initializes SQLite connection and tables

**Class Diagram:** The application’s core entity is the **Item**.

+------------------+

| Item |

+------------------+

| id : integer |

| name : string |

| quantity : int |

| category : string|

| purchased : bool |

+------------------+

**Database Schema**: Matches Item entity, implemented in SQLite.

**3. Data & Interface Design**

 **Technology Choices**: FastAPI (lightweight, auto-generates Swagger UI for testing) and SQLite (simple, file-based database suitable for small projects).

 **Data Model**: Single entity (Item) with attributes (id, name, quantity, category, purchased).

 **APIs**: CRUD endpoints exposed via FastAPI and testable through Swagger UI.

 **UI/UX**: Minimal interface through Swagger UI, with the option to extend to a basic HTML frontend.

### 4. Security & Performance

* **Security**
  + Input validation (non-empty names, limits on string length).
  + SQLite ensures safe data persistence.
  + Authentication out of scope for Version 1.
* **Performance**
  + CRUD operations expected to complete in <150ms for up to 100 items.
  + Lightweight FastAPI backend ensures responsiveness.
* **Maintainability**
  + Modular folder structure, separation of concerns.
* **Scalability**
  + Future scaling possible: replace SQLite with MySQL, deploy via Docker for DevOps pipeline.

**5. Proposed Folder Structure**

The project will use modular organization:

grocery\_todo/

main.py # FastAPI entry point

models.py # Defines Item model

crud.py # CRUD operations

database.py # SQLite setup

docs/ # Planning, Requirements, Design docs

frontend/ # HTML + JS user interface

grocery.db # Local SQLite database

### 6. API Endpoints Specification

### 