



**CEBU INSTITUTE OF TECHNOLOGY**  
**U N I V E R S I T Y**

# IT342-G4

## System Integration and Architecture

---

### **System Design Document (SDD)**

---

Project Title: DailyThoughts

Prepared By: Moreno, Elaiza Jane R.

Version: 0.1

Date: February 02, 2026

Status: Draft

REVISION HISTORY TABLE

Version	Date	Author	Changes Made	Status
0.1	02/02/2026	Moreno, Elaiza Jane R.	Initial draft	Draft
0.2	[Date]	[Your Name]	Added API specifications	Review
0.3	[Date]	[Your Name]	Updated database design	Review
0.4	[Date]	[Your Name]	Added UI/UX designs	Review
0.5	[Date]	[Your Name]	Incorporated feedback	Revised
0.6	[Date]	[Your Name]	Final review and corrections	Final
1	[Date]	[Your Name]	Baseline version for development	Approved

## Contents

EXECUTIVE SUMMARY .....	3
1.0 INTRODUCTION .....	5
2.0 FUNCTIONAL REQUIREMENTS SPECIFICATION .....	5
3.0 NON-FUNCTIONAL REQUIREMENTS .....	9
4.0 SYSTEM ARCHITECTURE .....	10
5.0 API CONTRACT & COMMUNICATION .....	11
6.0 DATABASE DESIGN .....	15
7.0 UI/UX DESIGN .....	17
8.0 PLAN .....	19

## EXECUTIVE SUMMARY

### 1.1 Project Overview

A straightforward social web and mobile software called DailyThoughts lets users register, log in, and share brief text-based thoughts that resemble a Twitter feed. In addition to browsing public posts, users can make their own ideas, modify or remove their own posts, leave comments on other users' posts, and browse user profiles.

The system includes a Spring Boot backend API, React web application, and Android mobile app, all working together to give a consistent and easy-to-use experience across platforms.

## **1.2 Objectives**

1. Develop a simple social application for sharing daily thoughts
2. Implement user authentication (register, login, logout)
3. Provide a home feed where users can scroll and view posts
4. Allow users to create, edit, and delete their own thoughts
5. Enable users to comment on posts
6. Support both web and mobile platforms

## **1.3 Scope**

### **Included Features:**

- User registration and authentication (email/password)
- Home feed with scrollable posts
- Create daily thoughts (text posts)
- Edit and delete own posts
- Comment on posts
- User profile page
- Responsive web interface
- Mobile application
- MySQL database with relational schema

### **Excluded Features:**

- AI-generated content
- Payment system

- Email notifications
- Push notifications
- Cloud synchronization
- Advanced analytics

## 1.0 INTRODUCTION

### 1.1 Purpose

This document serves as the design and requirements specification for the DailyThoughts system. It defines the system features, user interactions, and technical requirements to guide the development of the web and mobile applications.

## 2.0 FUNCTIONAL REQUIREMENTS SPECIFICATION

### 2.1 Project Overview

**Project Name:** DailyThoughts

**Domain:** Lifestyle/Productivity

**Primary Users:** General User

**Problem Statement:**

Many people want a simple and easy way to express their daily thoughts, feelings, and reflections without the complexity of full-featured social media or journaling platforms.

**Solution:**

DailyThoughts is a minimalist social platform where users can quickly post, view, and manage short thoughts through a clean and user-friendly interface on both web and mobile.

### 2.2 Core User Journeys

**Journey 1: First-time user writing a thought**

1. User open application

2. Clicks "Sign Up" and creates account
3. Logs in successfully
4. Navigates to add thought
5. Writes a daily thought
6. Posts the thought
7. Sees confirmation and view the thought in the lists or homepage.

### **Journey 2: Returning User Managing Thoughts**

1. User logs in with existing credentials
2. View posting in the home page, and previous thoughts
3. Select a thought to view details.
4. Edits or delete a thought
5. Log out

### **2.3 Feature List (MoSCoW)**

#### **MUST HAVE**

1. User authentication (register, login, logout)
2. Create Dailythoughts (Posting)
3. View home feed with scrolling posts
4. Edit and delete own post
5. Comment on posts
6. View user profile

#### **SHOULD HAVE**

1. Thoughts Timestamps(date & time)
2. Display username on post

#### **COULD HAVE**

1. Mood selection (happy, sad, calm)
2. Like or heart reaction

#### **WON'T HAVE**

1. AI-generated thoughts
2. Cloud sync
3. Email notifications

## 2.4 Detailed Feature Specifications

### Feature: User Authentication

- **Screens:** Registration, Login
- **Fields:** Email, Username, Password
- **API Endpoints:** POST /auth/register, POST /auth/login, POST /auth/logout
- **Security:** JWT Authentication

### Feature: Thoughts Posting

- **Screens:** Product Listing, Product Detail
- **Functions:** Create, edit, delete own posts
- **Display:** Username, content, timestamp
- **API Endpoints:** GET /posts, POSTS /posts/,PUT/Posts/{id},DELETE/posts/{id}

### Feature: Shopping Cart

- **Screens:** Cart View
- **Functions:** Add product, remove product, update quantity
- **Persistence:** Database storage per user
- **API Endpoints:** GET /cart, POST /cart/items, PUT /cart/items/{id}, DELETE /cart/items/{id}

### Feature: Comments

- **Screens:** Post Detail
- **Functions:** Add and view comments
- **API Endpoints:**

GET /posts/{id}/comments

POST /posts/{id}/comments

### **Feature: Profile**

- **Screens:** Profile Page
- **Functions:** View user information and posts
- **API Endpoints:**

GET /users/{id}

GET /users/{id}/posts

## **2.5 Acceptance Criteria**

### **AC-1: Successful User Registration**

Given the user is new

When valid registration details are entered

Then the account is created

And the user is redirected to the home page

### **AC-2: Create Thoughts**

Given the user is logged in

When a thoughts is posted

Then the thoughts appears in the home feed

### **AC-3: Edit/Delete Own Post**

Given the user owns a post

When the user edits or deletes the post

Then the changes are reflected immediately

## 3.0 NON-FUNCTIONAL REQUIREMENTS

### 3.1 Performance Requirements

- API response time  $\leq 2$  seconds
- Feed loads within 3 seconds
- Mobile app cold start:  $\leq 3$  seconds
- Support 100 concurrent users

### 3.2 Security Requirements

- HTTPS for all communications
- JWT token authentication
- Password hashing
- Users can edit/delete only their own posts

### 3.3 Compatibility Requirements

- **Web Browsers:** Chrome
- **Android:** Android 7.0+
- **Responsive design for all screen sizes**

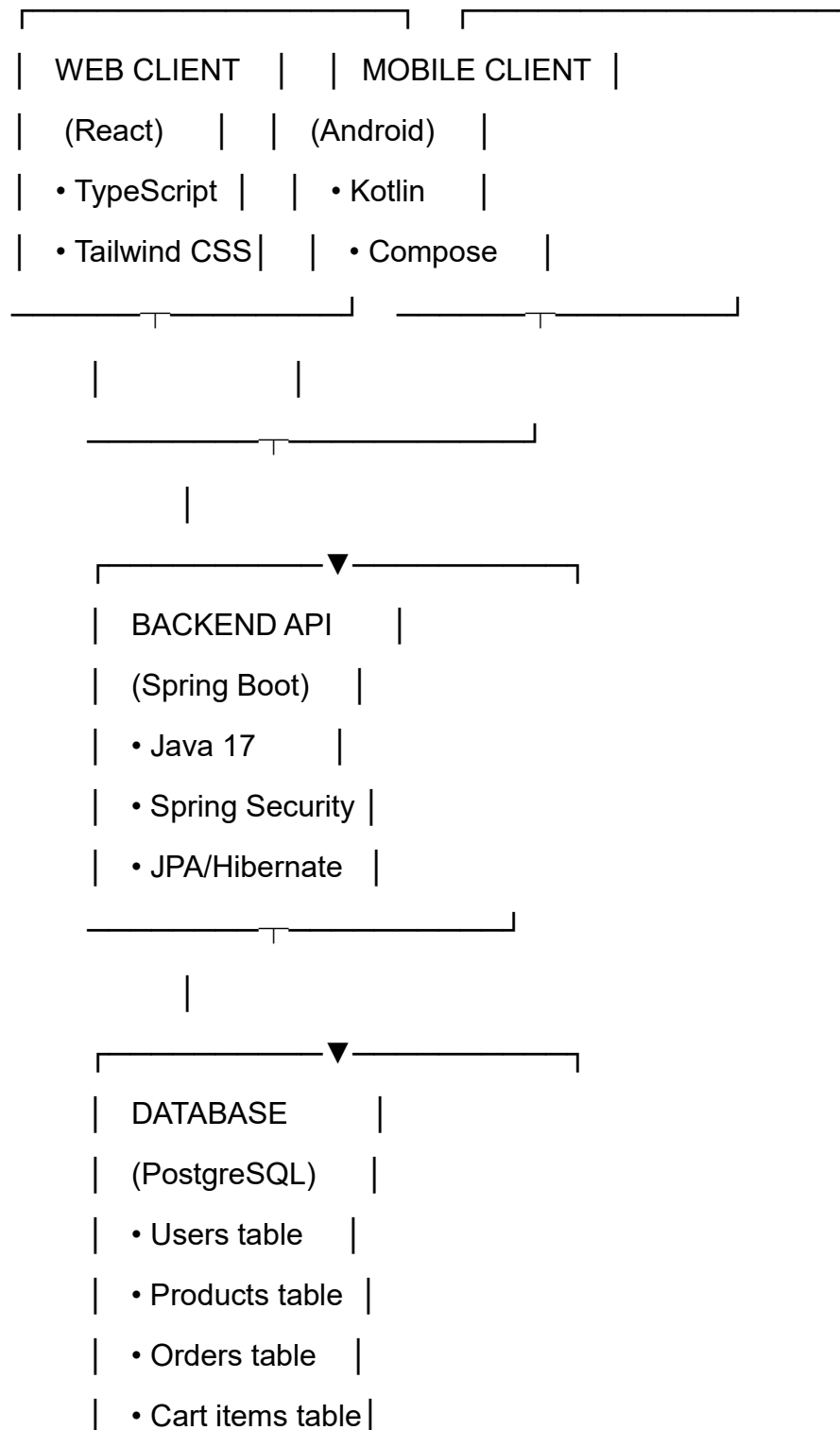
### 3.4 Usability Requirements

- Simple and clean interface
- Easy navigation similar to Twitter
- Clear buttons and readable text

## 4.0 SYSTEM ARCHITECTURE

### 4.1 Component Diagram

*Note: This should be a component diagram*



---

## Technology Stack:

- **Backend:** Java 17, Spring Boot 3.x, Spring Security, Spring Data JPA
- **Database:** PostgreSQL 14+
- **Web Frontend:** React 18, TypeScript, Tailwind CSS, Axios
- **Mobile:** Kotlin, Jetpack Compose, Retrofit, Room
- **Build Tools:** Maven (Backend), npm/yarn (Web), Gradle (Android)
- **Deployment:** Railway/Heroku (Backend), Vercel/Netlify (Web), APK (Mobile)

## 5.0 API CONTRACT & COMMUNICATION

### 5.1 API Standards

- **Base URL:** <https://api.marketplace.com/api/v1>
- **Format:** JSON for all requests/responses
- **Authentication:** Bearer token (JWT) in Authorization header
- **Response Structure:**

json

```
{  
  "success": boolean,  
  "data": object|null,  
  "error": {  
    "code": string,  
    "message": string,  
    "details": object|null  
  },  
  "timestamp": string  
}
```

### 5.2 Endpoint Specifications

## Authentication Endpoints

POST /auth/register

Body: {email, password, confirmPassword, fullName?}

Response: {user: {id, email, name}, token, refreshToken}

POST /auth/login

Body: {email, password}

Response: {user: {id, email, name, role}, token, refreshToken}

POST /auth/logout

Headers: Authorization: Bearer {token}

Response: {message: "Logged out successfully"}

## Product Endpoints

GET /products

Query: ?page=1&limit=20&search=keyword&category=electronics

Response: {products: [...], pagination: {page, limit, total, pages}}

GET /products/{id}

Response: {product: {id, name, description, price, stock, imageUrl, category}}

POST /products (Admin only)

Body: {name, description, price, stock, imageUrl, category}

Response: {product: {...}}

PUT /products/{id} (Admin only)

Body: {name?, description?, price?, stock?, imageUrl?, category?}

Response: {product: {...}}

## **Cart Endpoints**

GET /cart (Authenticated)

Response: {cart: {id, items: [...], total, itemCount}}

POST /cart/items

Body: {productId, quantity}

Response: {message: "Added to cart", cartItem: {...}}

PUT /cart/items/{itemId}

Body: {quantity}

Response: {message: "Cart updated", cartItem: {...}}

DELETE /cart/items/{itemId}

Response: {message: "Removed from cart"}

## **Order Endpoints**

POST /orders

Body: {shippingAddress: {fullName, address, city, zipCode, country}}

Response: {order: {id, orderNumber, total, status, items: [...], createdAt}}

GET /orders

Response: {orders: [...]}

GET /orders/{id}

Response: {order: {...}}

## 5.3 Error Handling

### HTTP Status Codes

- 200 OK - Successful request
- 201 Created - Resource created
- 400 Bad Request - Invalid input
- 401 Unauthorized - Authentication required/failed
- 403 Forbidden - Insufficient permissions
- 404 Not Found - Resource doesn't exist
- 409 Conflict - Duplicate resource
- 500 Internal Server Error - Server error

### Error Code Examples

json

```
{  
  "success": false,  
  "data": null,  
  "error": {  
    "code": "AUTH-001",  
    "message": "Invalid credentials",  
    "details": "Email or password is incorrect"  
  },  
  "timestamp": "2024-01-28T10:30:00Z"  
}
```

```
{  
  "success": false,  
  "data": null,
```

```

"error": {
  "code": "VALID-001",
  "message": "Validation failed",
  "details": {
    "email": "Email is required",
    "password": "Must be at least 8 characters"
  }
},
"timestamp": "2024-01-28T10:30:00Z"
}

```

### Common Error Codes

- AUTH-001: Invalid credentials
- AUTH-002: Token expired
- AUTH-003: Insufficient permissions
- VALID-001: Validation failed
- DB-001: Resource not found
- DB-002: Duplicate entry
- BUSINESS-001: Insufficient stock
- SYSTEM-001: Internal server error

## 6.0 DATABASE DESIGN

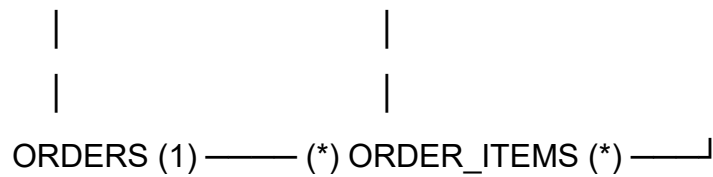
### 6.1 Entity Relationship Diagram

*Note: This should be an ERD*

```

USERS (1) — (1) CARTS (1) — (*) CART_ITEMS (*) — (1) PRODUCTS
|
|
(*)
|
|
(*)

```



### Detailed Relationships:

- **One-to-One:** User ↔ Cart (Each user has exactly one cart)
- **One-to-Many:** User → Orders (User can have multiple orders)
- **One-to-Many:** Cart → CartItems (Cart contains multiple items)
- **One-to-Many:** Order → OrderItems (Order contains multiple items)
- **Many-to-One:** CartItems → Product (Items reference products)
- **Many-to-One:** OrderItems → Product (Items reference products)

### Key Tables:

1. **users** - User accounts and authentication
2. **products** - Product catalog information
3. **carts** - Shopping cart per user
4. **cart\_items** - Items in shopping cart
5. **orders** - Customer orders
6. **order\_items** - Items in each order
7. **refresh\_tokens** - JWT refresh tokens

### Table Structure Summary:

- **users:** id, email, password\_hash, full\_name, role, created\_at
- **products:** id, name, description, price, stock, image\_url, category
- **carts:** id, user\_id, created\_at
- **cart\_items:** id, cart\_id, product\_id, quantity
- **orders:** id, order\_number, user\_id, total, status, shipping\_address
- **order\_items:** id, order\_id, product\_id, product\_name, quantity, price

## 7.0 UI/UX DESIGN

### 7.1 Web Application Wireframes

*Note: This should be wireframes from Figma*

#### Homepage (Product Listing)

Header: [Logo] [Search Bar] [Cart Icon] [User Menu]

Content: Product Grid (3 columns desktop)

Each Product Card: Image, Name, Price, "Add to Cart" button

Footer: Links, Copyright

#### Product Detail Page

Back Button

Product Image (large)

Product Name and Price

Description

Quantity Selector (1-10)

"Add to Cart" and "Buy Now" buttons

Product Specifications

#### Shopping Cart Page

Cart Title

List of Cart Items (Image, Name, Quantity, Price, Remove)

Order Summary: Subtotal, Shipping, Tax, Total

"Continue Shopping" and "Proceed to Checkout" buttons

#### Checkout Page

Shipping Address Form

Order Review (Items, Prices, Totals)

"Place Order" button

Terms and Conditions note

## **Admin Dashboard**

Sidebar Navigation: Dashboard, Products, Orders, Users

Product Management: Add New button, Product list with Edit/Delete

Order Management: Order list with status filters

## **7.2 Mobile Application Wireframes**

*Note: This should be wireframes from Figma*

### **Bottom Navigation**

[🏠 Home] [🔍 Search] [🛒 Cart] [👤 Profile]

### **Home Screen**

Search Bar

Product Grid (2 columns)

Swipe gestures for quick actions

Pull to refresh

### **Product Detail Screen**

Back arrow

Product image (swipeable gallery)

Product info

Quantity selector

"Add to Cart" fixed bottom button

### **Cart Screen**

Edit mode for quantity updates

Swipe to remove items

Order summary sticky bottom

Checkout button

### **Checkout Flow**

Step indicator: Cart → Shipping → Payment → Confirm

Address form (auto-complete)

Order summary

Place order button

### **Mobile-Specific Features:**

- Touch-optimized buttons (min 44x44px)
- Gesture support (swipe, pull-to-refresh)
- Offline caching for product images
- Bottom navigation for main actions
- Simplified forms for mobile input

### **Design System:**

- **Colors:** Primary (#2563EB), Secondary (#7C3AED), Success (#10B981), Error (#EF4444)
- **Typography:** Inter font family, responsive sizing
- **Spacing:** 8px grid system
- **Components:** Consistent buttons, inputs, cards, modals
- **Responsive:** Mobile-first approach, breakpoints at 640px, 768px, 1024px

## **8.0 PLAN**

### **8.1 Project Timeline**

#### **Phase 1: Planning & Design (Week 1-2)**

Week 1: Requirements & Architecture

Day 1-2: Project setup and documentation

Day 3-4: Complete FRS and NFR

Day 5-7: System architecture design

Week 2: Detailed Design

Day 1-2: API specification

Day 3-4: Database design

Day 5-6: UI/UX wireframes

Day 7: Implementation plan finalization

## **Phase 2: Backend Development (Week 3-4)**

Week 3: Foundation

Day 1: Spring Boot setup with dependencies

Day 2: Database configuration and entities

Day 3: JWT authentication implementation

Day 4: User management endpoints

Day 5: Product CRUD operations

Week 4: Core Features

Day 1: Cart functionality

Day 2: Order management

Day 3: Search and filtering

Day 4: Error handling and validation

Day 5: API documentation and testing

## **Phase 3: Web Application (Week 5-6)**

Week 5: Frontend Foundation

Day 1: React setup with TypeScript  
Day 2: Authentication pages (login, register)  
Day 3: Product listing page  
Day 4: Product detail page  
Day 5: Shopping cart implementation

#### Week 6: Complete Web Features

Day 1: Checkout flow  
Day 2: Order history and confirmation  
Day 3: Admin dashboard  
Day 4: Responsive design polish  
Day 5: API integration and testing

### **Phase 4: Mobile Application (Week 7-8)**

#### Week 7: Android Foundation

Day 1: Android Studio setup and project structure  
Day 2: Authentication screens  
Day 3: Product browsing  
Day 4: Shopping cart  
Day 5: API service layer

#### Week 8: Complete Mobile App

Day 1: Checkout flow  
Day 2: Order management  
Day 3: UI polish and animations  
Day 4: Testing on emulator/device  
Day 5: APK generation and documentation

## **Phase 5: Integration & Deployment (Week 9-10)**

### **Week 9: Integration Testing**

- Day 1: End-to-end testing across platforms
- Day 2: Bug fixes and optimization
- Day 3: Security review
- Day 4: Performance testing
- Day 5: Documentation updates

### **Week 10: Deployment**

- Day 1: Backend deployment (Railway/Heroku)
- Day 2: Web app deployment (Vercel/Netlify)
- Day 3: Mobile APK distribution
- Day 4: Final testing
- Day 5: Project submission

### **Milestones:**

- **M1 (End Week 2):** All design documents complete
- **M2 (End Week 4):** Backend API fully functional
- **M3 (End Week 6):** Web application complete
- **M4 (End Week 8):** Mobile application complete
- **M5 (End Week 10):** Full system deployed and integrated

### **Critical Path:**

1. Authentication system (Week 3)
2. Product catalog API (Week 3-4)
3. Shopping cart functionality (Week 4)
4. Checkout process (Week 6)
5. Cross-platform testing (Week 9)

**Risk Mitigation:**

- Start with simplest working version of each feature
- Test integration points early and often
- Keep backup of working versions
- Focus on core functionality before enhancements