CEBU INSTITUTE OF TECHNOLOGY
UNIVERSITY

IT342-G5
System Integration and Architecture

System Design Document (SDD)

Project Title: SynChef

Prepared By: Batawang, Vince Labadan

Version: 1.0

Date: 2/20/2026
Status: Draft

REVISION HISTORY TABLE

Version	Date	Author	Changes Made	Status
0.1	2/20/2026	Vince L. Batawang	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

TABLE OF CONTENTS

Contents

EXECUTIVE SUMMARY

1.1 Project Overview

SynChef is a smart culinary guidance system designed to transform traditional recipe-following into an interactive, real-time cooking experience. Unlike conventional recipe platforms that present static instructions, SynChef orchestrates cooking workflows through synchronized timers, adaptive ingredient scaling, and culturally contextualized recipe discovery.

The system integrates a ***Django REST backend***, ***PostgreSQL database***, and ***React-based frontend*** to deliver an execution-focused environment that assists users while they cook—not before or after.

SynChef supports sustainable consumption, improves cooking accessibility, and promotes global culinary literacy by embedding intelligent logic into recipe execution.

1.2 Objectives

1. Develop an interactive cooking platform that manages parallel cooking tasks in real time.
2. Implement dynamic ingredient scaling to reduce food waste and match household needs..
3. Provide a global culinary discovery interface through a visual flavor-mapping system.
4. Design a step-focused cooking mode to reduce user cognitive overload.
5. Build **RESTful APIs** enabling scalable communication between frontend and backend services.
6. Establish a modular architecture that supports future AI-driven features such as substitution recommendations and timing optimization.

1.3 Scope

Included Features:

- User registration and authentication
- Recipe browsing by country and category
- Parallel timer orchestration for cooking steps
- Dynamic ingredient quantity and time scaling
- Step-by-step Focus Mode interface
- Cultural ingredient substitution suggestions (rule-based MVP)
- REST API services for recipes, steps, and scaling logic
- PostgreSQL relational database

- Responsive web interface

Excluded Features:

- Full AI-based personalization engine
- Voice assistant integration
- Grocery delivery integration
- Computer vision cooking detection
- Social media sharing features
- Monetization or payment systems

1.0 INTRODUCTION

1.1 Purpose

This document serves as the comprehensive design specification for the SynChef system. It provides detailed functional requirements, architectural decisions, API communication structure, database design considerations, and an implementation roadmap to guide the development process and ensure all system components integrate seamlessly.

By defining these specifications early, the document minimizes integration risks, prevents design inconsistencies, and establishes a structured foundation for delivering a reliable, scalable, and user-centered guided cooking platform.

2.0 FUNCTIONAL REQUIREMENTS SPECIFICATION

2.1 Project Overview

Project Name: SynChef
Domain: Smart Food Technology / Guided Cooking System
Primary Users:

1. Home cooks
2. Learners
3. Food Enthusiasts

Problem Statement: Users lack an interactive system that can guide real-time cooking, coordinate simultaneous tasks, and adapt recipes dynamically to reduce waste and complexity.
Solution: SynChef provides an execution-driven cooking environment that converts recipes into synchronized workflows.

2.2 Core User Journeys

Journey 1: Guided Cooking Experience

1. User logs into the system.
2. Selects a recipe from the Flavor Map.
3. Inputs number of servings (Pax).
4. System recalculates ingredients and timing.
5. User enters Focus Mode.
6. Parallel timers guide task execution.
7. User completes cooking with synchronized alerts.

Journey 2: Exploring Global Cuisine

1. User opens Flavor Map
2. Selects continent → country
3. Views culturally authentic recipes
4. Reads ingredient background and preparation method
5. Saves recipe to personal collection

Journey 3: Adjusting Meal Portions

1. User selects a recipe
2. Changes serving size
3. System dynamically updates:
    - Ingredient quantities
    - Cooking durations
    - Required steps
4. User proceeds with updated workflow

2.3 Feature List (MoSCoW)

MUST HAVE

1. User authentication (register/login/logout)
2. Recipe management and retrieval
3. Parallel timer execution engine
4. Dynamic ingredient scaling
5. Step-by-step Focus Mode
6. Country-based recipe categorization
7. RESTful API communication

SHOULD HAVE

1. Ingredient substitution suggestions
2. Recipe filtering by dietary category

3. Saved recipes functionality
4. Responsive UI animations for transitions

COULD HAVE

1. AI-based timing optimization
2. Personalized recommendations
3. Multilingual instructions

WON'T HAVE

1. Full AI automation
2. Grocery ordering integration
3. Smart kitchen device integration
4. Voice-controlled cooking

2.4 Detailed Feature Specifications

Feature: User Authentication

- **Screens**: Registration, Login
- **Fields**: Name, Email, Password, Confirm Password
- **Validation**: Email format validation, password length (min. 8 characters), uniqueness of account
- **Security**: JWT-based authentication, password hashing using Django authentication system
- **API Endpoints**:

  POST /api/auth/register
  POST /api/auth/login
  POST /api/auth/logout

Feature: Recipe Catalog Management

- **Screens**: Recipe Listing, Recipe Detail
- **Display**: Recipe image, name, country of origin, preparation time, servings
- **Search**: By recipe name, category, or country
- **Filtering**: Vegan, Dessert, Main Dish, etc.
- **API Endpoints:**

GET /api/recipes
GET /api/recipes/{id}
GET /api/recipes/search
GET /api/recipes/country/{country}

Feature: Global Flavor Map Navigation

- **Screens**: Interactive Map View, Country Selection View
- **Display**: Continent → Country navigation with cuisine highlights
- **Functions**: Load culturally relevant recipes based on selected region
- **Purpose**: Promote culinary education and cultural exploration
- **API Endpoints:**
  GET /api/countries
  GET /api/recipes/country/{country}

Feature: Adaptive Parallel Timer System

- **Screens**: Cooking Mode, Active Timer Dashboard
- **Functions**: Run multiple cooking timers simultaneously, synchronize task completion
- **Display**: Countdown timers per step, alerts for next action
- **Logic**: Backend schedules step overlap to optimize cooking workflow
- **API Endpoints:**
  GET /api/recipes/timer-sequence/{id}
  POST /api/timers/start
  POST /api/timers/update

Feature: Dynamic Ingredient Scaling

- **Screens**: Recipe Detail, Serving Adjustment Panel
- **Fields**: Number of servings (Pax input)
- **Functions**: Automatically recompute ingredient quantities and cooking duration
- **Precision Handling**: Decimal-based calculations for accurate scaling
- **API Endpoints:**
  POST /api/recipes/scale
  GET /api/recipes/{id}

Feature: Progressive Step-by-Step Focus Mode

- **Screens**: Focus Cooking Interface

- **Display**: One instruction visible at a time with navigation controls
- **Functions**: Prevent user distraction, guide sequential execution
- **Integration**: Syncs with timer engine for automated progression
- **API Endpoints:**

  GET /api/recipes/{id}/steps

Feature: Ingredient Substitution Assistance (Rule-Based MVP)

- **Screens**: Ingredient Information Panel
- **Functions**: Suggest alternative ingredients based on availability or region
- **Logic**: Predefined substitution mappings stored in database
- **Purpose**: Improve accessibility and reduce food waste
- **API Endpoints:**

  GET /api/ingredients/substitutions/{ingredient}

Feature: Saved Recipes / Personal Collection

- **Screens**: User Dashboard, Saved Recipes List
- **Functions**: Bookmark recipes for future cooking
- **Persistence**: Stored per authenticated user
- **API Endpoints:**

  GET /api/users/saved
  POST /api/users/saved/{recipeId}
  DELETE /api/users/saved/{recipeId}

2.5 Acceptance Criteria

AC-1: Successful Recipe Execution

- Given a user selects a recipe
- When Focus Mode is activated
- Then the system must guide the user step-by-step
- And synchronize timers automatically
- And notify when each action must begin.

AC-2: Ingredient Scaling Accuracy

- Given a recipe designed for 4 servings

- When the user changes to 2 servings
- Then all ingredient values must adjust proportionally
- And updated measurements must display instantly.

AC-3: Parallel Task Coordination

- Given a recipe contains overlapping steps
- When cooking begins
- Then timers must run simultaneously
- And alerts must ensure synchronized completion.

AC-4: Cultural Recipe Discovery

- Given a user selects a country
- When recipes load
- Then the system must display cuisine-specific dishes
- And associated preparation context.

3.0 NON-FUNCTIONAL REQUIREMENTS

3.1 Performance Requirements

- API response time: ≤ 2 seconds for 95% of requests
- Web page load time: ≤ 3 seconds on broadband
- Mobile app cold start: ≤ 3 seconds
- Support 100 concurrent users
- Database queries complete within 500ms

3.2 Security Requirements

- HTTPS for all communications
- JWT token authentication
- Password hashing with bcrypt (salt rounds = 12)
- SQL injection prevention
- XSS protection
- Rate limiting: 100 requests/minute per IP
- Admin endpoints require role verification

3.3 Compatibility Requirements

- **Web Browsers:** Chrome, Firefox, Safari, Edge (latest 2 versions)
- **Android:** API Level 24+ (Android 7.0+)
- **Screen Sizes:** Mobile (360px+), Tablet (768px+), Desktop (1024px+)
- **Operating Systems:** Windows 10+, macOS 10.15+, Linux Ubuntu 20.04+

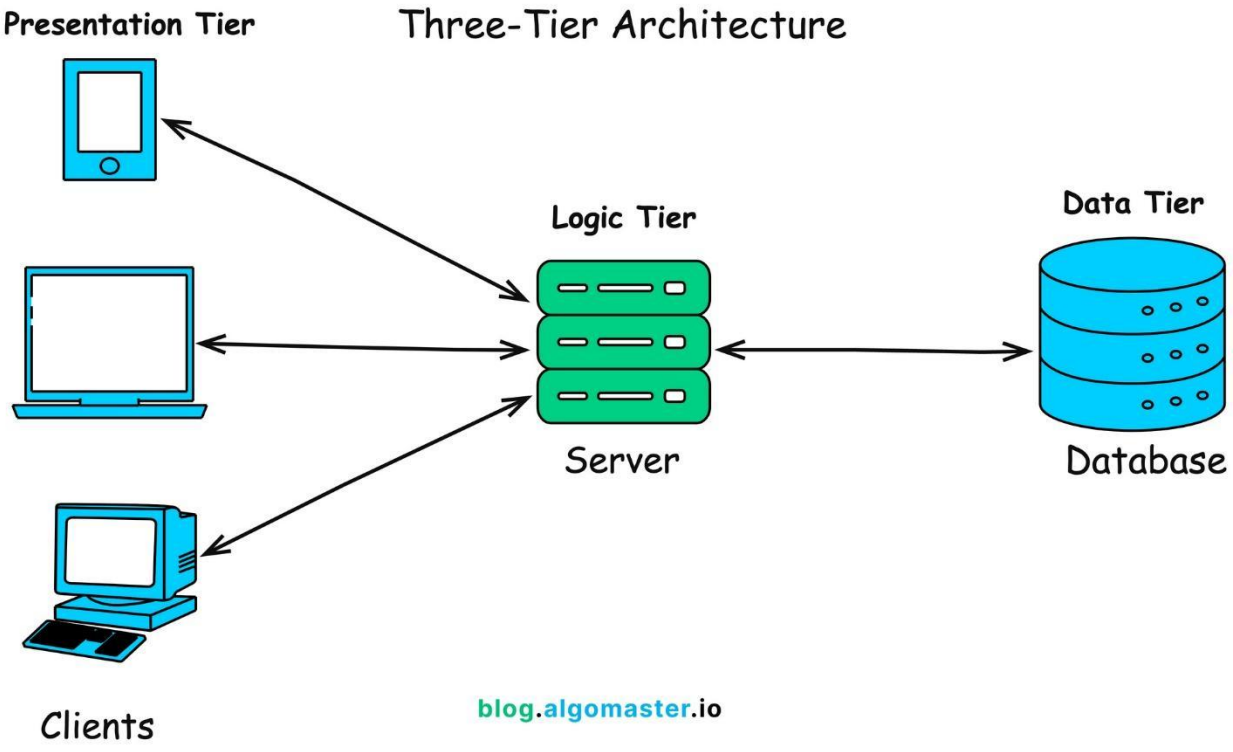
3.4 Usability Requirements

- Complete first purchase within 5 minutes for new users
- WCAG 2.1 Level AA compliance for web
- Consistent navigation across all pages
- Clear error messages with recovery options
- Touch targets minimum 44x44px on mobile
- Keyboard navigation support

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://[server_hostname]:[port]/api/v1
Format	JSON for all requests/responses
Authentication	Bearer token (JWT) in Authorization header

Response Structure	{
	"success": boolean,
	"data": object|null,
	"error": {
	"code": string,
	"message": string,

	"details": object|null }, "timestamp": string }

5.2 Endpoint Specifications

Authentication Endpoints

User Registration

Description	User Registration
API URL	/auth/register
HTTP Request Method	POST
Format	JSON for all requests/responses
Authentication	None

Request Payload	{
	"email": <email>,
	"password": <password>,
	"firstname": <firstname>,
	"lastname": <lastname>,
	}

Response Structure	{
	"success": boolean,
	"data": {
	user {
	"email": <email>,
	"firstname": <firstname>,
	"lastname": <lastname>,
	},
	"accessToken": <token>,
	"refreshToken": <token>,
	},
	"error": {
	"code": string,
	"message": string,
	"details": object|null
	},
	"timestamp": string
	}

User Login

Description	User Login
API URL	/auth/login
HTTP Method	POST

Format	JSON for all requests/responses
Authentication	None

Request Payload	{
	“email”: <email>,
	“password”: <password>,
	}

Response Structure	{
	"success": boolean,
	"data": {
	“user”: {
	“email”: <email>,
	“firstname”: <firstname>,
	“lastname”: <lastname>,
	“role”: <role>,
	},
	“accessToken”: <token>,
	“refreshToken”: <token>,
	},
	"error": {
	"code": string,
	"message": string,
	"details": object|null
	},
	"timestamp": string
	}

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

Example 1	json { "success": false,

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

Example 2	{
	"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 (Database Schema)

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