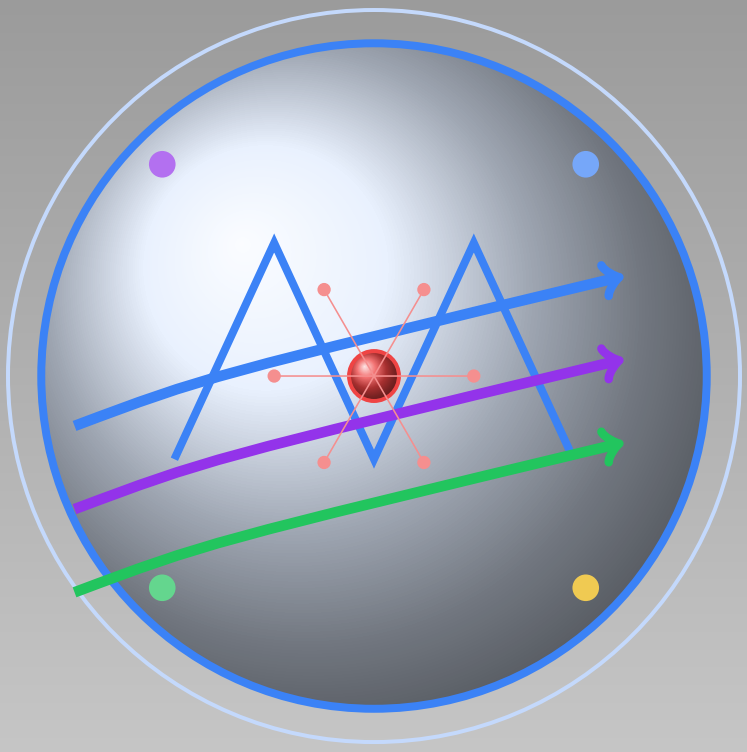


MOMENTUM

AI-Powered Time Management & Productivity Platform

Technical Documentation

Document Information

Version: v1.0.0
Date: November 14, 2025
Author: Ayush Pandey
GitHub: github.com/AyushPandey003

Contents

Abstract

Momentum is a cutting-edge AI-powered time management and productivity platform designed for students, professionals, and anyone seeking to optimize their daily workflow. By combining intelligent task decomposition, automated scheduling, gamification mechanics, and real-time competitive features, Momentum transforms how users approach productivity and time management.

The platform leverages modern web technologies including Next.js 14, React Server Components, Drizzle ORM with PostgreSQL, and WebSocket-based real-time communication to deliver a seamless, responsive user experience. Advanced AI capabilities powered by Google's Gemini model enable intelligent task breakdown and personalized productivity coaching.

Key innovations include:

- **AI Task Decomposition** - Automatically breaks complex tasks into manageable subtasks
- **Intelligent Scheduling** - Smart time-blocking with Pomodoro integration
- **Real-time Contests** - Competitive knowledge challenges with friends
- **Gamification Engine** - Achievements, levels, and leaderboards
- **Wellness Integration** - Automated break reminders and health tracking
- **Manager Verification** - Task accountability through email verification

Chapter 1

Introduction

1.1 Overview

Momentum addresses the universal challenge of productivity management by providing an intelligent, adaptive system that understands user behavior and optimizes task completion. Unlike traditional to-do list applications, Momentum actively coaches users, prevents procrastination, and makes productivity engaging through game mechanics.

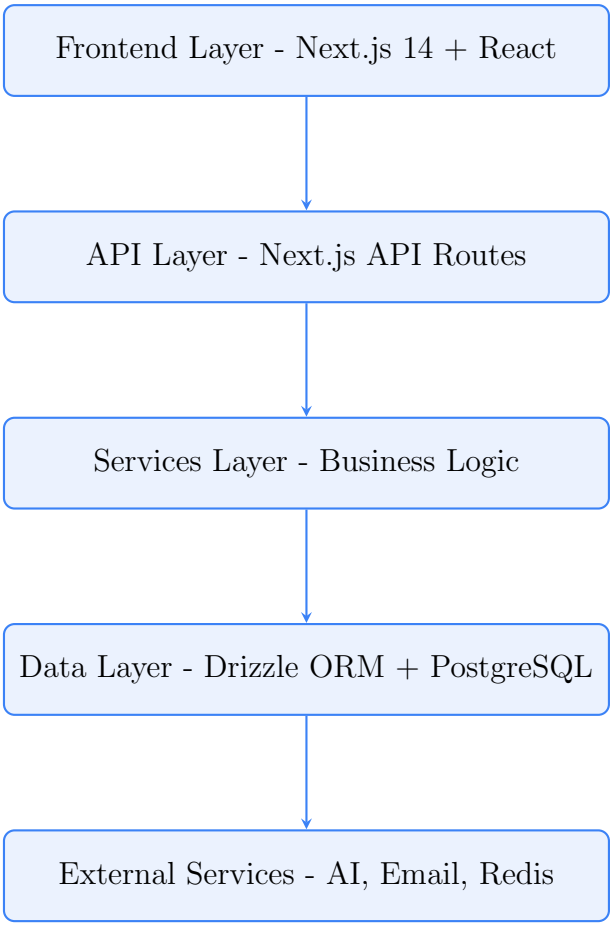
1.2 Problem Statement

Modern professionals and students face several productivity challenges:

1. **Task Overwhelm** - Large projects appear insurmountable
2. **Poor Time Estimation** - Difficulty accurately estimating task duration
3. **Procrastination Patterns** - Unconscious avoidance of difficult tasks
4. **Lack of Accountability** - No external verification of task completion
5. **Motivation Decay** - Difficulty maintaining long-term productivity habits

1.3 Solution Architecture

Momentum employs a multi-layered architecture to address these challenges:



1.4 Technology Stack

Frontend Technologies

- **Next.js 14** - React framework with App Router and Server Components
- **TypeScript** - Type-safe development environment
- **Tailwind CSS** - Utility-first CSS framework
- **Radix UI** - Accessible component primitives
- **Shadcn/ui** - Beautiful, customizable UI components

Backend Technologies

- **Next.js API Routes** - Serverless API endpoints
- **Better Auth** - Modern authentication solution
- **Drizzle ORM** - Type-safe SQL ORM
- **PostgreSQL (Neon)** - Serverless PostgreSQL database
- **Upstash Redis** - Serverless Redis for caching and rate limiting

AI & External Services

- **Google Gemini AI** - Task decomposition and AI mentoring
- **Gmail API** - Email notifications and invitations
- **UploadThing** - File upload and storage
- **React Email** - Beautiful transactional emails

Chapter 2

Core Features

2.1 AI-Powered Task Decomposition

2.1.1 Overview

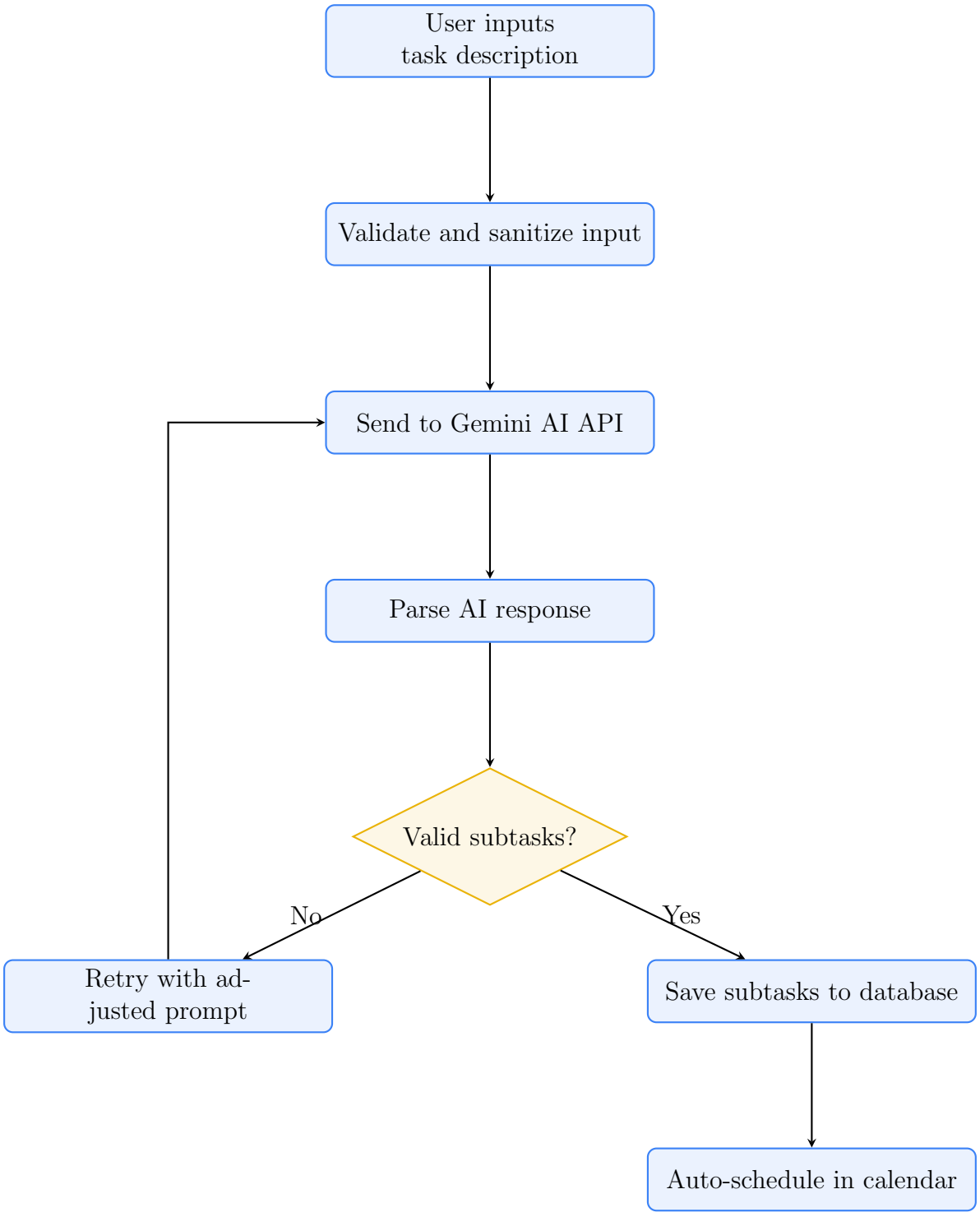
The task decomposition engine uses Google's Gemini AI model to analyze complex tasks and break them into manageable subtasks with estimated time allocations.

2.1.2 Technical Implementation

Listing 2.1: AI Task Decomposition Service

```
// lib/ai.ts
import { GoogleGenerativeAI } from "@google/generative-ai";

const genAI = new GoogleGenerativeAI(
    process.env.GOOGLE_GEMINI_API_KEY!
);

export async function decomposeTask(taskDescription: string) {
    const model = genAI.getGenerativeModel({
        model: "gemini-1.5-flash"
    });

    const prompt = `Break down this task into subtasks...`;
    const result = await model.generateContent(prompt);
    return parseSubtasks(result.response.text());
}
```

2.1.3 Algorithm Flow



2.1.4 Subtask Schema

Each generated subtask contains:

- **Title** - Clear, actionable description
- **Estimated Time** - AI-predicted duration in minutes

- **Priority** - Relative importance (low, medium, high, urgent)
- **Dependencies** - Sequential ordering if required
- **Completion Status** - Boolean tracking

2.2 Intelligent Scheduling System

2.2.1 Pomodoro Integration

The scheduler integrates the Pomodoro Technique with intelligent time-blocking:

$$\text{Pomodoro Session} = \begin{cases} \text{Focus Block} & : 25 \text{ minutes (default)} \\ \text{Short Break} & : 5 \text{ minutes} \\ \text{Long Break} & : 15 \text{ minutes (every 4 sessions)} \end{cases}$$

2.2.2 Schedule Block Types

Task Block	Break Block	Wellness Block	Event Block
Focused work	Short rest	Exercise/stretch	Calendar sync

2.2.3 Auto-Scheduling Algorithm

The scheduler considers:

1. User-defined work hours (default: 9 AM - 5 PM)
2. Task priority and deadlines
3. Estimated task duration
4. Existing calendar commitments
5. Mandatory wellness breaks every 30 minutes
6. User's historical productivity patterns

2.3 Procrastination Detection & Prevention

2.3.1 Detection Mechanisms

The system monitors for procrastination patterns:

1. **Deadline Approaching** - Task due within 24 hours but not started
2. **Task Avoidance** - High-priority task skipped multiple times
3. **Pattern Detection** - Consistent delays on specific task types

2.3.2 Alert Severity Levels

Low - Gentle reminder

Medium - Suggest rescheduling

High - AI coaching intervention

2.4 Gamification System

2.4.1 Points & Experience

Users earn points through various activities:

Activity	Points	XP
Complete Task	10-50	10-50
Complete Pomodoro	5	5
Maintain Streak	20/day	20/day
Unlock Achievement	Varies	Varies
Win Contest	100	100

Table 2.1: Points and XP earning system

2.4.2 Level Progression Formula

The level calculation uses a square root scaling function:

$$\text{Level} = \lfloor \sqrt{\frac{\text{Total Points}}{100}} \rfloor + 1$$

Points required for next level:

$$\text{Points}_{\text{next}} = \text{Current Level}^2 \times 100$$

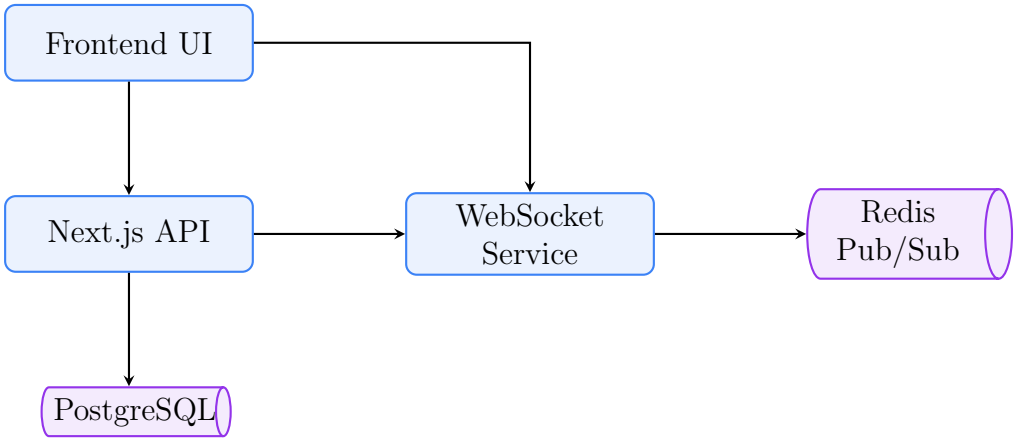
2.4.3 Achievement System

Achievement Categories

- **Task Achievements** - Complete 1, 10, 50, 100 tasks
- **Pomodoro Achievements** - Focus mastery milestones
- **Streak Achievements** - Consistency rewards (3, 7, 30 days)
- **Contest Achievements** - Competition participation and victories
- **Special Achievements** - Unique accomplishments

2.5 Contest System

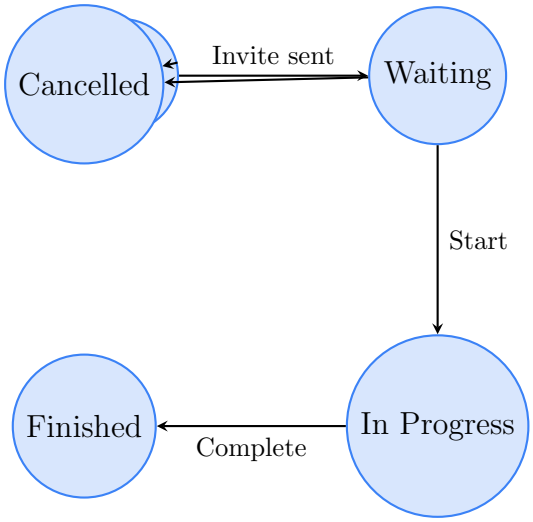
2.5.1 Contest Architecture



2.5.2 Contest Types

1. **Quick Fire** (15 min) - Fast-paced, 10 questions
2. **Standard** (30 min) - Balanced competition
3. **Marathon** (60 min) - Endurance challenge

2.5.3 Contest Flow State Machine



2.5.4 Real-time Scoring Algorithm

Scoring considers both accuracy and speed:

$$\text{Points} = \text{Base Points} \times \left(1 + \frac{\text{Time Remaining}}{\text{Total Time}} \times 0.5\right)$$

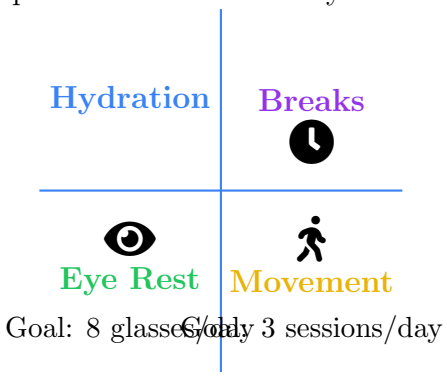
Where:

- Base Points = 10 (configurable per question)
- Time bonus up to 50% of base points
- Faster correct answers earn more points

2.6 Wellness & Health Integration

2.6.1 Wellness Metrics

The platform tracks four key wellness dimensions:



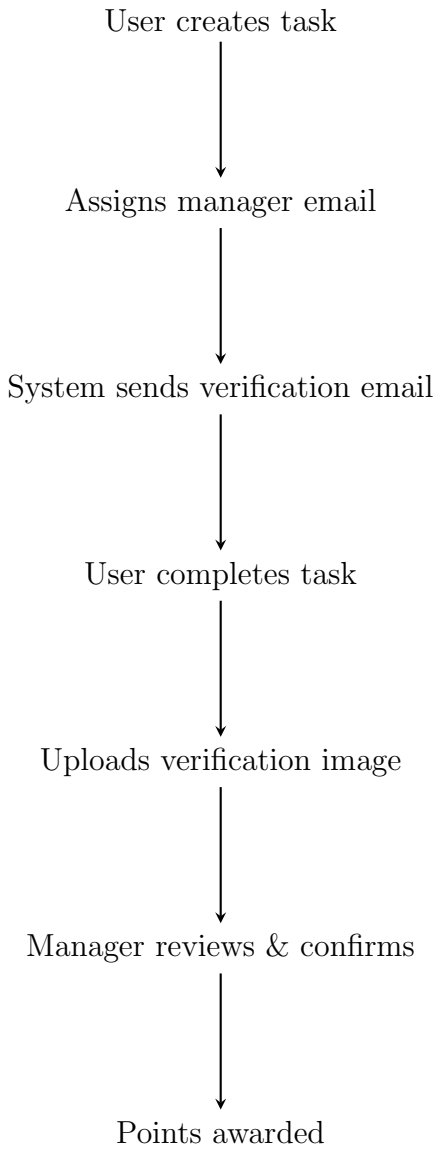
2.6.2 Reminder System

Automatic reminders based on activity patterns:

- **Every 30 min** - Stand and stretch reminder
- **Every 60 min** - Hydration reminder
- **Every 20 min** - 20-20-20 eye rest rule
- **Configurable** - User preferences override defaults

2.7 Manager Verification System

2.7.1 Workflow



2.7.2 Security Measures

Verification Token System:

- Unique cryptographic tokens per task
- Token expiration (7 days default)
- One-time use validation
- Email domain verification
- Image upload validation via UploadThing

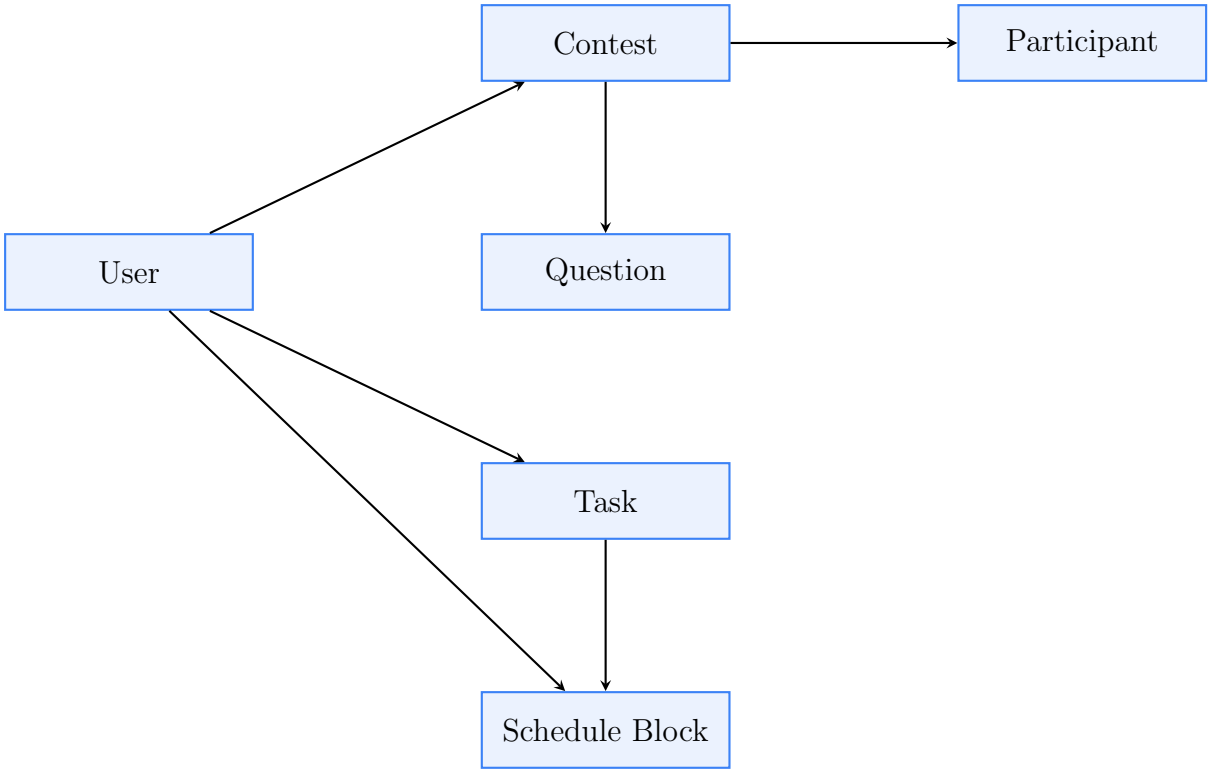
Chapter 3

Database Architecture

3.1 Schema Overview

The database uses PostgreSQL with Drizzle ORM for type-safe queries. The schema consists of 20+ tables organized into logical domains.

3.2 Entity Relationship Diagram



3.3 Core Tables

3.3.1 User Table

Listing 3.1: User Schema

```sql
CREATE TABLE "user" (
    id TEXT PRIMARY KEY,
    name TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    email_verified BOOLEAN DEFAULT false,
    image TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
```

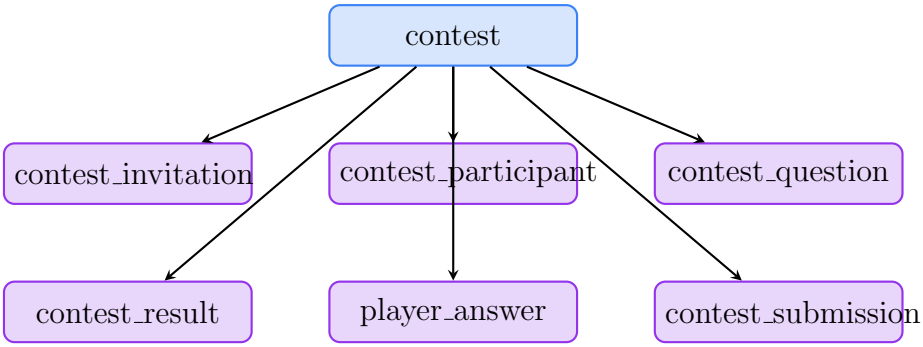
3.3.2 Task Table

Task Schema Fields

- `id` - Unique identifier (ULID)
- `userId` - Foreign key to user
- `title` - Task description
- `description` - Detailed information
- `dueDate` - Deadline (ISO 8601)
- `priority` - ENUM: low — medium — high — urgent
- `status` - ENUM: todo — in-progress — completed — overdue
- `estimatedTime` - Duration in minutes
- `actualTime` - Tracked completion time
- `subtasks` - JSON array of subtasks
- `aiDecomposed` - Boolean flag
- `managerEmail` - Verification manager
- `verificationImageUrl` - Proof of completion

3.3.3 Contest Tables

The contest system uses 7 interconnected tables:



3.3.4 Gamification Tables

1. **user_stats** - Points, level, streaks, completion counts
2. **user_achievements** - Unlocked achievement tracking
3. **procrastination_alert** - Alert history and dismissals
4. **user_preferences** - Personal settings and preferences

3.4 Indexing Strategy

Performance-critical indexes:

Listing 3.2: Database Indexes

```
-- User queries
CREATE INDEX idx_task_user_status
    ON task(user_id, status);

-- Contest queries
CREATE INDEX idx_contest_status
    ON contest(status, start_date);

-- Leaderboard queries
CREATE INDEX idx_user_stats_points
    ON user_stats(total_points DESC);
```

Chapter 4

API Reference

4.1 Authentication Endpoints

4.1.1 POST /api/auth/signup

Create a new user account.

Listing 4.1: Request Body

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "SecurePass123!"
}
```

Listing 4.2: Response (201)

```
{
  "user": {
    "id": "usr_01HKXYZ...",
    "email": "john@example.com",
    "name": "John Doe"
  },
  "session": {
    "token": "sess_token_...",
    "expiresAt": "2024-12-31T23:59:59Z"
  }
}
```

4.2 Task Management Endpoints

4.2.1 POST /api/tasks

Create a new task with optional AI decomposition.

Request Parameters

- `title` (required) - Task title
- `description` (required) - Detailed description
- `dueDate` (required) - ISO 8601 deadline
- `priority` (required) - low — medium — high — urgent
- `estimatedTime` (required) - Minutes
- `aiDecompose` (optional) - Boolean to trigger AI
- `managerEmail` (optional) - Verification manager

4.2.2 GET /api/tasks

Retrieve user's tasks with optional filtering.

Listing 4.3: Query Parameters

```
GET /api/tasks?status=todo&priority=high&limit=20
```

4.3 Contest Endpoints

4.3.1 POST /api/contest

Create a new contest.

Listing 4.4: Request Body

```
{
  "name": "Friday Quiz Night",
  "description": "Weekly trivia challenge",
  "contestType": "standard",
  "difficulty": "medium",
  "category": "Programming",
  "questionCount": 10,
  "durationMinutes": 30,
  "maxParticipants": 5,
  "invites": [
    "friend1@example.com",
    "friend2@example.com"
  ]
}
```

4.3.2 WebSocket Connection

Real-time contest communication via WebSocket.

Listing 4.5: WebSocket Connection

```
ws://localhost:8080/ws/contest/{contestId}

// Authentication via query params
?userId={userId}&token={sessionToken}
```

WebSocket Message Types

- `join_lobby` - User joins waiting room
- `start_contest` - Organizer initiates contest
- `question_display` - New question broadcast
- `answer_submit` - User submits answer
- `score_update` - Leaderboard update
- `contest_complete` - Final results

4.4 Gamification Endpoints

4.4.1 GET /api/leaderboard

Retrieve global or friend leaderboard.

Listing 4.6: Response

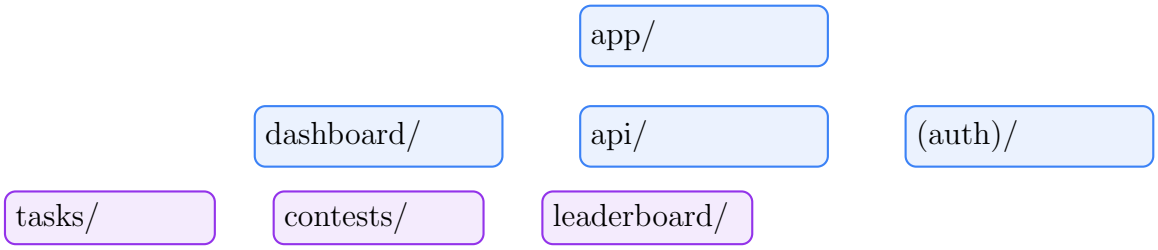
```
{
  "leaderboard": [
    {
      "userId": "usr_01...",
      "userName": "John Doe",
      "points": 4850,
      "level": 7,
      "rank": 1,
      "streak": 15
    }
  ],
  "currentUser": {
    "rank": 1,
    "points": 4850
  }
}
```

Chapter 5

Frontend Architecture

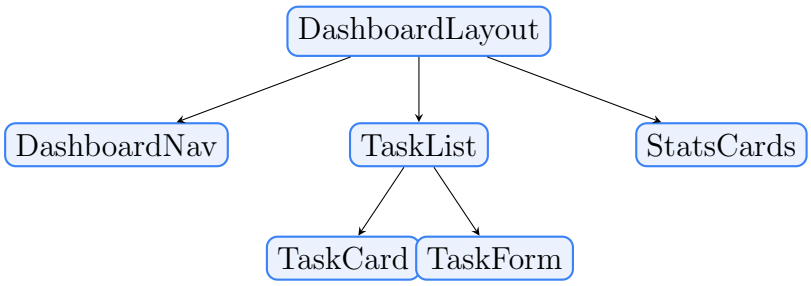
5.1 Next.js App Router Structure

Momentum uses Next.js 14's App Router with React Server Components for optimal performance.



5.2 Component Architecture

5.2.1 Component Hierarchy



5.2.2 State Management

Momentum uses multiple state management strategies:

State Management Layers

1. **Server State** - React Query for API data
2. **Client State** - Zustand stores for UI state
3. **Form State** - React Hook Form for forms
4. **URL State** - Next.js router for navigation state

5.3 UI Component Library

Built on Shadcn/ui and Radix UI primitives:

Component	Usage
Button	Actions and CTAs
Card	Content containers
Dialog	Modals and overlays
Form	Data input
Toast	Notifications
Badge	Status indicators
Progress	Loading states
Tabs	Content organization

Table 5.1: Core UI components

5.4 Styling System

5.4.1 Tailwind Configuration

Custom design tokens extend Tailwind's defaults:

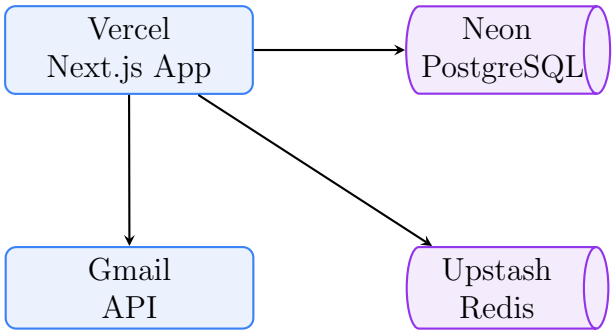
Listing 5.1: tailwind.config.js

```
module.exports = {
  theme: {
    extend: {
      colors: {
        primary: 'hsl(var(--primary))',
        secondary: 'hsl(var(--secondary))',
        accent: 'hsl(var(--accent))',
      },
      animation: {
        'fade-in': 'fadeIn 0.3s ease-in',
        'slide-up': 'slideUp 0.4s ease-out',
      },
    },
  },
}
```

Chapter 6

Deployment & DevOps

6.1 Deployment Architecture



6.2 Environment Variables

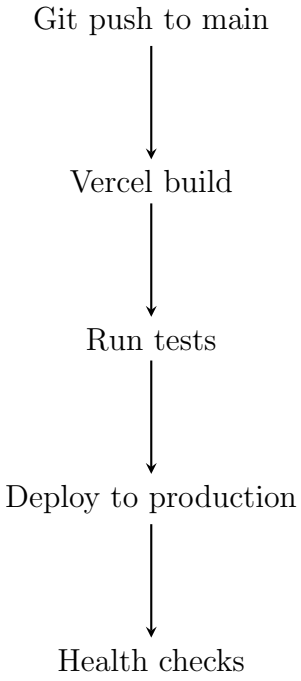
Listing 6.1: .env.local

```
# Database
DATABASE_URL=postgresql://...
DIRECT_URL=postgresql://...

# Auth
BETTER_AUTH_SECRET=...
BETTER_AUTH_URL=http://localhost:3000

# AI
GOOGLE_GEMINI_API_KEY=...

# Redis
UPSTASH_REDIS_REST_URL=...
UPSTASH_REDIS_REST_TOKEN=...

# Email
GMAIL_USER=...
GMAIL_APP_PASSWORD=...

# Storage
```

```
UPLOADTHING_SECRET=...
UPLOADTHING_APP_ID=...
```

6.3 CI/CD Pipeline

Vercel provides automatic deployments:



6.4 Monitoring & Observability

Monitoring Stack

- **Vercel Analytics** - Web vitals and performance
- **PostgreSQL Logs** - Database query performance
- **Redis Monitoring** - Cache hit rates
- **Error Tracking** - Runtime error aggregation
- **Uptime Monitoring** - Service availability checks

Chapter 7

Security & Privacy

7.1 Authentication Security

Better Auth provides enterprise-grade security:

- **Password Hashing** - Argon2id algorithm
- **Session Management** - Secure, httpOnly cookies
- **CSRF Protection** - Token-based validation
- **Rate Limiting** - Upstash Redis limiter
- **Email Verification** - Required for sensitive actions

7.2 Data Protection

Transport Layer - TLS 1.3 Encryption

Application Layer - Input Validation

Database Layer - Parameterized Queries

Storage Layer - Encrypted at Rest

7.3 Privacy Compliance

GDPR Compliance Features:

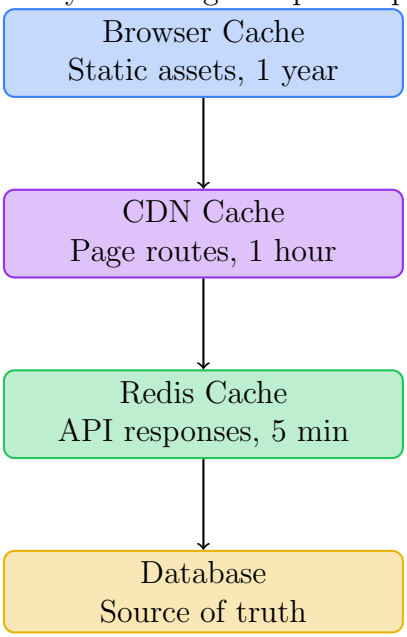
- User data export capability
- Account deletion with cascade
- Cookie consent management
- Privacy policy integration
- Data retention policies

Chapter 8

Performance Optimization

8.1 Caching Strategy

Multi-layer caching for optimal performance:



8.2 Database Optimization

8.2.1 Query Optimization

Listing 8.1: Optimized Query Example

```
-- Use indexes and select only needed fields
SELECT
    t.id, t.title, t.status, t.priority
FROM task t
WHERE
    t.user_id = $1
    AND t.status = 'todo'
ORDER BY t.priority DESC, t.due_date ASC
LIMIT 20;
```

8.2.2 Connection Pooling

Drizzle ORM with Neon serverless driver handles connection pooling automatically:

```
import { neon, neonConfig } from '@neondatabase/serverless';
import { drizzle } from 'drizzle-orm/neon-http';

neonConfig.fetchConnectionCache = true;
const sql = neon(process.env.DATABASE_URL!);
export const db = drizzle(sql);
```

8.3 Frontend Performance

8.3.1 Code Splitting

Next.js automatic code splitting:

Listing 8.2: Dynamic Imports

```
import dynamic from 'next/dynamic';

const HeavyComponent = dynamic(
  () => import('@/components/heavy-component'),
  { loading: () => <Skeleton /> }
);
```

8.3.2 Image Optimization

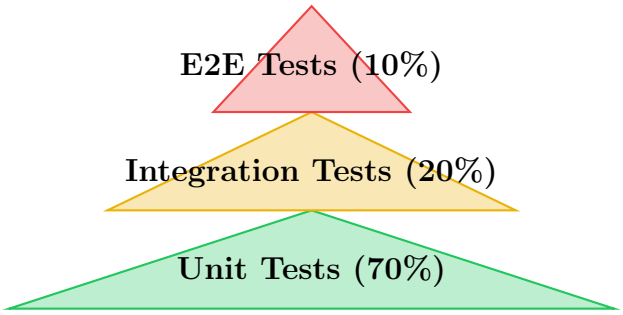
Next.js Image component with automatic optimization:

```
<Image
  src="/hero.png"
  alt="Momentum"
  width={1200}
  height={600}
  priority
  placeholder="blur"
/>
```

Chapter 9

Testing Strategy

9.1 Testing Pyramid



9.2 Unit Testing

Example unit test for gamification logic:

Listing 9.1: Jest Unit Test

```
import { calculateLevel, getPointsForNextLevel } from '@/lib/
   gamification';

describe('Gamification', () => {
  test('calculates correct level from points', () => {
    expect(calculateLevel(0)).toBe(1);
    expect(calculateLevel(100)).toBe(2);
    expect(calculateLevel(400)).toBe(3);
    expect(calculateLevel(900)).toBe(4);
  });

  test('calculates points needed for next level', () => {
    expect(getPointsForNextLevel(1)).toBe(400);
    expect(getPointsForNextLevel(2)).toBe(900);
  });
});
```

9.3 Integration Testing

API route testing with Vitest:

Listing 9.2: API Integration Test

```
import { POST } from '@/app/api/tasks/route';

describe('/api/tasks', () => {
  it('creates task with AI decomposition', async () => {
    const request = new Request('http://localhost:3000/api/tasks
       ', {
      method: 'POST',
      body: JSON.stringify({
        title: 'Build feature',
        description: 'Implement new dashboard',
        dueDate: '2024-12-31',
        priority: 'high',
        estimatedTime: 240,
        aiDecompose: true
      })
    });

    const response = await POST(request);
    expect(response.status).toBe(201);
  });
});
```

Chapter 10

Future Enhancements

10.1 Roadmap

Q1 2025 - Mobile app (React Native)

Q2 2025 - Team collaboration features

Q3 2025 - Advanced AI coaching

Q4 2025 - Integration marketplace

10.2 Planned Features

10.2.1 Mobile Applications

Native iOS and Android apps with:

- Push notifications for task reminders
- Offline mode with sync
- Widget support for quick task access
- Biometric authentication

10.2.2 Team Collaboration

Team Features

- Shared workspaces and task boards
- Team leaderboards and challenges
- Manager dashboard for team oversight
- Collaborative goal setting
- Team productivity analytics

10.2.3 Advanced AI

- **Predictive Scheduling** - ML-based optimal time suggestions
- **Personalized Coaching** - Adaptive AI mentor
- **Habit Recognition** - Automatic pattern detection
- **Smart Prioritization** - AI-driven task ordering

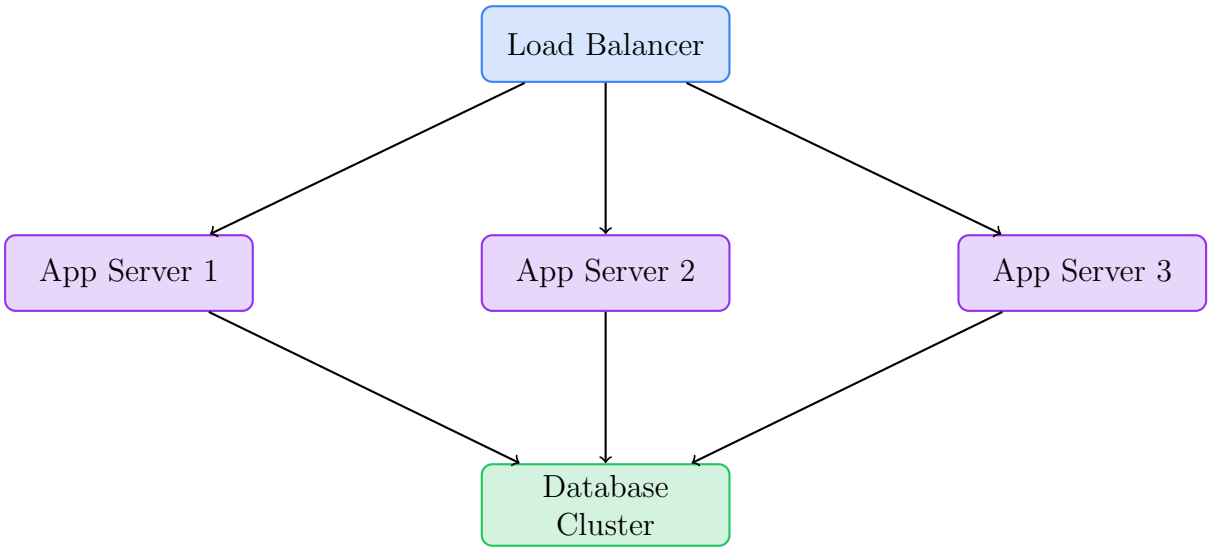
10.2.4 Integration Ecosystem

Planned integrations:

- Calendar: Google Calendar, Outlook, Apple Calendar
- LMS: Canvas, Blackboard, Moodle, Google Classroom
- Communication: Slack, Discord, Microsoft Teams
- Project Management: Jira, Trello, Asana
- Health: Apple Health, Google Fit, Fitbit

10.3 Scaling Considerations

10.3.1 Infrastructure Scaling



10.3.2 Database Sharding

For large-scale deployment:

- User-based sharding (shard by user_id)
- Read replicas for query distribution
- Eventual consistency for non-critical data

Appendix A

API Quick Reference

A.1 Authentication

- POST /api/auth/signup - Create account
- POST /api/auth/login - User login
- POST /api/auth/logout - End session
- POST /api/auth/reset-password - Password reset

A.2 Tasks

- GET /api/tasks - List tasks
- POST /api/tasks - Create task
- PATCH /api/tasks/[id] - Update task
- DELETE /api/tasks/[id] - Delete task

A.3 Contests

- GET /api/contest - List contests
- POST /api/contest - Create contest
- GET /api/contest/[id] - Contest details
- POST /api/contest/[id]/join - Join contest

Appendix B

Glossary

AI Decomposition Process of breaking complex tasks into subtasks using AI

Gamification Application of game mechanics to increase engagement

Pomodoro Time management technique with focused work intervals

Procrastination Alert System notification about task avoidance

Schedule Block Time-boxed calendar entry for tasks or breaks

Streak Consecutive days of task completion

Wellness Metric Tracked health and break activities

Appendix C

Contributing Guidelines

C.1 Development Setup

Listing C.1: Local Setup Commands

```
# Clone repository
git clone https://github.com/AyushPandey003/momentum-app.git
cd momentum-app

# Install dependencies
pnpm install

# Setup environment
cp .env.example .env.local
# Fill in environment variables

# Run database migrations
pnpm drizzle-kit push

# Start development server
pnpm dev
```

C.2 Code Style

- **TypeScript** - Strict mode enabled
- **ESLint** - Follow Next.js recommended config
- **Prettier** - Automatic code formatting
- **Commit Convention** - Conventional Commits standard

C.3 Pull Request Process

1. Fork the repository

2. Create a feature branch: `git checkout -b feature/amazing-feature`
3. Commit changes: `git commit -m 'feat: add amazing feature'`
4. Push to branch: `git push origin feature/amazing-feature`
5. Open a Pull Request with detailed description

Conclusion

Momentum represents a comprehensive approach to modern productivity management, combining cutting-edge AI technology with proven time management techniques and engaging gamification mechanics. The platform's architecture is designed for scalability, performance, and extensibility, making it suitable for both individual users and team deployments.

The integration of real-time competitive features, intelligent task decomposition, and wellness monitoring creates a holistic productivity ecosystem that addresses not just task completion, but overall well-being and sustained motivation.

As the platform continues to evolve, the roadmap focuses on expanding integration capabilities, enhancing AI intelligence, and building collaborative features that make productivity engaging and social.

Build Unstoppable Momentum

Resources & Contact

Project Links

- **GitHub Repository:** https://github.com/AyushPandey003/momentum-app
- **Live Demo:** https://momentum-app.vercel.app
- **Documentation:** https://momentum-docs.vercel.app

Author Contact

- **Name:** Ayush Pandey
- **GitHub:** AyushPandey003
- **Email:** ayush.pandey@example.com

License

Generated on November 14, 2025