**1. Planning**

Objective: Define the project scope, goals, and resources.

Activities:

- Identify stakeholders (e.g., developers, product owners, end-users such as developers using the API and administrators).
- Define project goals: Develop a secure SaaS platform for user account management (registration, login, email verification, password reset, admin access) and text case conversion (upper/lower case) with a credit-based system and API key authentication.
- Determine Technologies:
    - Backend: FastAPI, Python
    - Database: SQLite with SQLAlchemy 2.0 ORM (async) and aiosqlite
    - Authentication: JWT for access/refresh tokens, bcrypt for password hashing
    - Security: OAuth2, HTTP-only cookies for refresh tokens, API key authentication
    - Other: Pydantic for data validation, CORS middleware
- Establish requirements:
    - Functional: User registration/login, email verification, password reset, API key generation, text case conversion, credit management, admin functionality.
    - Non-functional: Security (secure tokens, password hashing), scalability, async database operations, user-friendly API responses.
- Define Agile sprints (e.g., 2-week iterations) with deliverables like user authentication in Sprint 1 and text conversion/credit system in Sprint 2.
- Allocate resources: Backend developers, database administrators, DevOps for deployment, and QA testers.
- Risk assessment: Address potential risks like SQLite scalability limitations, secret key exposure, and CORS misconfiguration in production.

**2. Requirements Analysis**

Objective: Gather and document detailed requirements.

Activities:

- User Stories:
    - As a user, I want to register and log in securely to access the text conversion API.
    - As a user, I want to verify my email to activate my account.
    - As a user, I want to generate an API key to use the text conversion service.
    - As a user, I want to manage credits to perform text conversions.
    - As an admin, I want to approve credit requests and access admin-only endpoints.
- Technical Requirements:
    - Database schema for users, refresh tokens, API keys, user credits, and credit requests
    - RESTful API endpoints for account management (/api/account/*) and text conversion (/api/convert/*).
    - Secure token handling (JWT with 15-minute access tokens, 7-day refresh tokens, 1-hour email/password reset tokens).
    - Credit-based system with default 10 credits per user, decremented per conversion.
    - Async database operations for performance.

- Non-functional Requirements:
  - Response time: API responses under 200ms for conversion requests.
  - Security: HTTPS, secure cookies, hashed passwords, unique API keys.
  - Scalability: Plan for potential database migration (e.g., PostgreSQL) for production.
- Tools: Use Jira or Trello for sprint planning, Git for version control.

**3. Design**

Objective: Create a detailed system architecture and design.

Activities:

- System Architecture:
  - Backend: FastAPI with modular routers (account/routers.py, converter/routers.py) for separation of concerns.
  - Database: SQLite with tables for user, refresh_token, user_credits, api_keys, and credit_requests.
  - Authentication: OAuth2 with JWT tokens, refresh tokens stored in the database, and API key authentication for conversions.
  - Middleware: CORS for frontend integration, configurable for production.
- API Design:
  - Account Endpoints (from account/routers.py):
    - POST /api/account/register: Create a user.
    - POST /api/account/login: Authenticate and return tokens.
    - POST /api/account/refresh: Refresh access token.
    - GET /api/account/me: Get user profile.
    - POST /api/account/verify-request: Send email verification link.
    - GET /api/account/verify: Verify email with token.
    - POST /api/account/change-password: Update password.
    - POST /api/account/forgot-password: Send password reset link.
    - POST /api/account/reset-password: Reset password with token.
    - GET /api/account/admin: Admin-only endpoint.
    - POST /api/account/logout: Revoke refresh token.
  - Converter Endpoints (from converter/routers.py):
    - POST /api/convert/generate-api-key: Generate API key.
    - GET /api/convert/me/api-key: Retrieve user's API key.
    - GET /api/convert/me/credits: Check credit balance.
    - POST /api/convert/buy-credits: Request additional credits.
    - GET /api/convert/credit-requests: List credit requests (admin-only).
    - POST /api/convert/approve-credit/{request_id}: Approve credit request (admin-only).
    - POST /api/convert/convert: Perform text case conversion.
  - Data Models: Pydantic schemas for request/response validation (e.g., UserCreate, ConvertRequest, CreditRequestOut).
- Database Design:
  - Tables: user (user data), refresh_token (JWT refresh tokens), user_credits (credit balance), api_keys (API keys), credit_requests (credit requests).
  - Relationships: One-to-many between User and RefreshToken, User and CreditRequest; one-to-one between User and UserCredits, User and APIKey

- Security Design:
  - Passwords hashed with bcrypt (passlib in utils.py).
  - JWT tokens with secret key and HS256 algorithm.
  - HTTP-only, secure cookies for refresh tokens with 7-day expiry.
  - API key authentication for conversion endpoint using X-API-Key header.
- Diagrams:
  - Create ERD (Entity-Relationship Diagram) for database tables.
  - Create sequence diagrams for authentication flow and text conversion process.
  - Tools: Use Draw.io for diagrams, Swagger (FastAPI's built-in) for API documentation.

**4. Development**

Objective: Implement the system based on the design.

Activities:

- Setup:
  - Initialize project with FastAPI, install dependencies (FastAPI, SQLAlchemy, aiosqlite, passlib, python-jose, pydantic).
  - Configure SQLite database with async support .
  - Set up Git repository and branching strategy (e.g., feature branches, main branch).
- Sprint Breakdown:
  - Sprint 1: Account Management (2 weeks):
    - Implement database models and configuration
    - Develop user authentication (register, login, refresh, logout)
    - Implement email verification and password reset functionality.
    - Add admin dependency
    - Write unit tests for authentication and token handling.
  - Sprint 2: Text Conversion and Credits (2 weeks):
    - Implement converter models for credits, API keys, and credit requests.
    - Develop text conversion
    - Implement API key generation and credit management
    - Add API key authentication
    - Write unit tests for conversion and credit deduction.
  - Sprint 3: Integration and Refinement (2 weeks):
    - Integrate routers in main.py with CORS middleware.
    - Implement database table creation on startup
    - Add input validation using Pydantic schemas
    - Write integration tests for API endpoints.
- Coding Standards:
  - Follow PEP 8 for Python code.
  - Use type hints and async/await for consistency
  - Modularize code into routers, services, and utilities.
- Version Control:
  - Commit changes frequently with clear messages (e.g., "Add user registration endpoint").
  - Use pull requests for code reviews.
- Tools: VS Code, Black for formatting, pytest for testing, GitHub for version control.

**5. Testing**

Objective: Ensure the system is functional, secure, and reliable.

Activities:

- Unit Testing:
  - Test convert text for upper/lower case conversions.
  - Test password hashing/verification
  - Test token creation/verification
  - Test database operations (e.g., user creation, credit deduction).
- Integration Testing:
  - Test API endpoints using FastAPI's TestClient.
  - Verify authentication flow (login → access token → refresh token).
  - Test credit deduction on conversion and API key validation.
- Security Testing:
  - Test for SQL injection in database queries.
  - Validate JWT token security (e.g., invalid/expired tokens).
  - Ensure refresh tokens are HTTP-only and secure.
  - Test CORS configuration for production safety.
- Performance Testing:
  - Measure API response times under load (e.g., using Locust).
  - Test async database performance with multiple concurrent requests.
- Test Cases:
  - Register user with existing email → Should fail with 400.
  - Convert text with insufficient credits → Should fail with 402.
  - Access admin endpoint without admin role → Should fail with 403.
  - Use invalid API key → Should failover to 401.
- Tools: pytest, FastAPI TestClient, Locust, OWASP ZAP for security testing.

**6. Deployment**

Objective: Deploy the application to a production environment.

Activities:

- Environment Setup:
  - Replace SQLite with a production-ready database (e.g., PostgreSQL) for scalability.
  - Configure environment variables for production
  - Use a WSGI/ASGI server like Uvicorn or Gunicorn with workers.
- Deployment Pipeline:
  - Set up CI/CD with GitHub Actions
  - Automate testing, building, and deployment to a cloud platform (e.g., AWS, Heroku, or DigitalOcean).
  - Use Docker to containerize the application for consistency.
- Production Configuration:
  - Enable HTTPS with SSL certificates (e.g., Let's Encrypt). Emulate email sending for verification/password reset
  - Configure CORS to allow only specific origins
- Rollout Strategy:

        - Deploy to a staging environment for final testing.
        - Use blue-green deployment to minimize downtime.
- Tools: Docker, AWS, Uvicorn, Let's Encrypt, Prometheus.

**7. Maintenance**

Objective: Monitor, update, and enhance the application.

Activities:

- Monitoring:
    - Monitor API uptime and performance using tools like New Relic.
    - Log errors and exceptions (add logging to FastAPI middleware).
    - Track credit usage and API key activity for analytics.
- Bug Fixes:
    - Address issues like token expiration errors or database connection failures.
    - Fix potential security vulnerabilities (e.g., update dependencies regularly).
- Updates:
    - Add new features (e.g., additional text operations like title case, analytics dashboard).
    - Optimize database queries for performance.
    - Upgrade to a more robust database if user growth demands it.
- User Feedback:
    - Collect feedback via user surveys or support tickets.
    - Implement requested features in future sprints (e.g., bulk conversion API).
- Security Maintenance:
    - Rotate SECRET_KEY periodically.
    - Monitor for unauthorized access attempts.
    - Update dependencies to patch vulnerabilities (e.g., using Dependabot).
- Tools: Sentry for error tracking, Dependabot for dependency updates, Jira for issue tracking.

**8. Agile Iterations**

- Sprint Reviews: At the end of each sprint, review deliverables with stakeholders, demo features (e.g., login flow, text conversion), and gather feedback.
- Retrospectives: Discuss what went well (e.g., modular code structure) and areas for improvement (e.g., email sending implementation).
- Backlog Refinement: Prioritize new features (e.g., payment integration for credits, rate limiting) for future sprints.