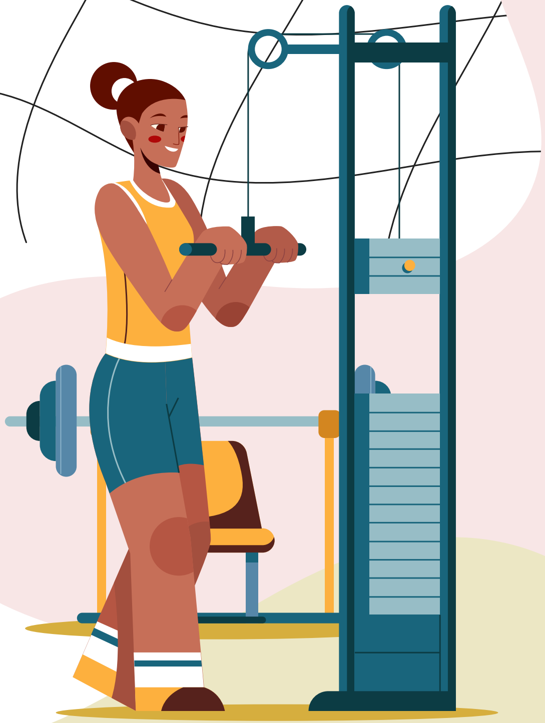
FIT TRACKER

Tatu Bogdan



Project Requirements

1

Project **Requirements**

An all-in-one gym app that tracks your workout progress, combined with a food tracker and calorie counter.

- Create and customize a profile
- Create a workout and add exercises to it (sets, reps, weight)
- Search vast amounts of foods
- Add/edit/remove foods from daily meals
- View progress history

Project Specifications

2

Project **Specifications**

01 Code Management

Git and GitHub
Each platform on different repo

02 Database

MySQL on PlanetScale
Scalability and analytics

03 API

TypeScript with tRPC
Developer experience and reduced boilerplate

04 Frontend

Web: Next.js
- SSR, SEO

Mobile: React Native
- Android and iOS

Tailwind CSS

05 Hosting

Web: Vercel
Mobile: Google Play Store and App Store

Task Breakdown

3

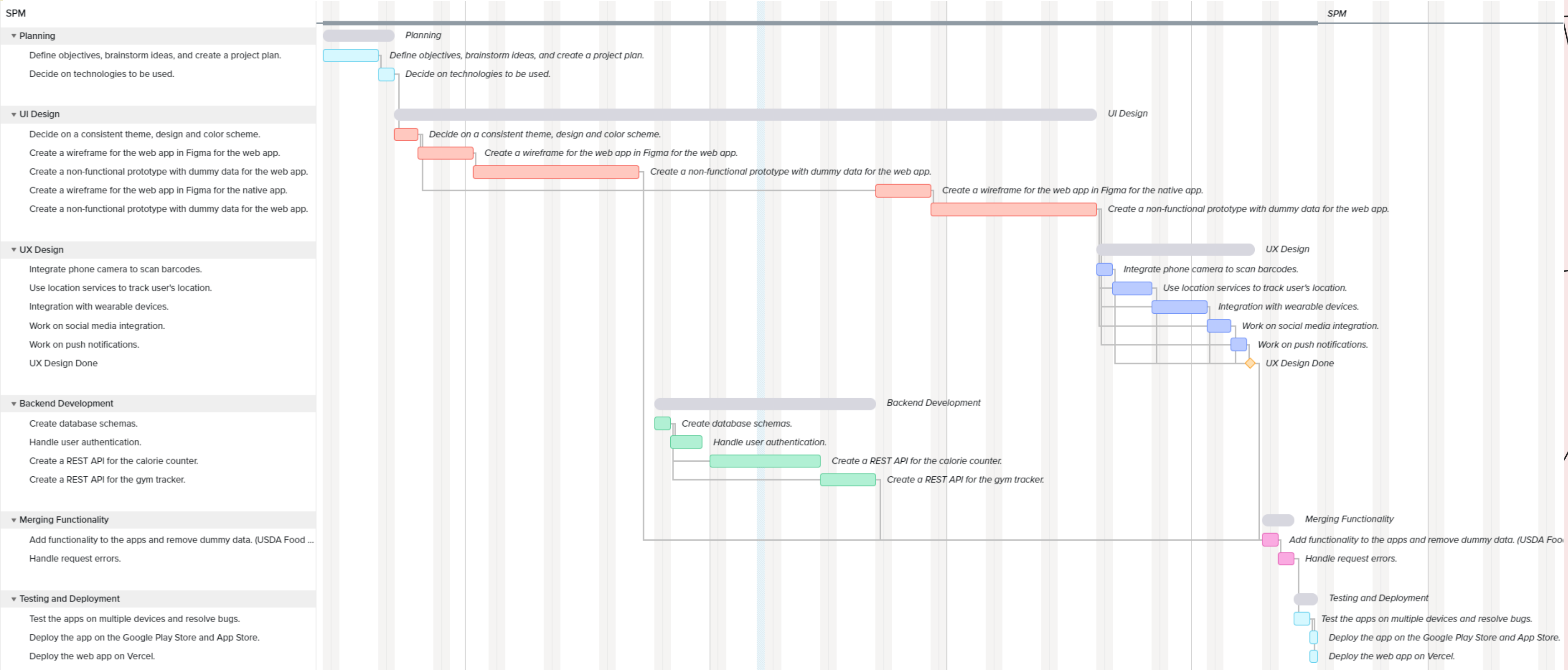
Task Breakdown

TASK	DURATION
Planning	1 week
UI Design	5 weeks (Web) & 5 weeks (Native)
UX Design and Integrations	3 weeks
Backend Development	4 weeks
Merging Functionality	1 week
Testing	3 days
Deployment	1 day

Gantt Chart

4

Gantt Chart



SPM

- Planning
  - Define objectives, brainstorm ideas, and create a project plan.
  - Decide on technologies to be used.
- UI Design
  - Decide on a consistent theme, design and color scheme.
  - Create a wireframe for the web app in Figma for the web app.
  - Create a non-functional prototype with dummy data for the web app.
  - Create a wireframe for the web app in Figma for the native app.
  - Create a non-functional prototype with dummy data for the web app.
- UX Design
  - Integrate phone camera to scan barcodes.
  - Use location services to track user's location.
  - Integration with wearable devices.
  - Work on social media integration.
  - Work on push notifications.
  - UX Design Done
- Backend Development
  - Create database schemas.
  - Handle user authentication.
  - Create a REST API for the calorie counter.
  - Create a REST API for the gym tracker.
- Merging Functionality
  - Add functionality to the apps and remove dummy data. (USDA Food ...
  - Handle request errors.
- Testing and Deployment
  - Test the apps on multiple devices and resolve bugs.
  - Deploy the app on the Google Play Store and App Store.
  - Deploy the web app on Vercel.

SPM
Planning
Define objectives, brainstorm ideas, and create a project plan.
Decide on technologies to be used.
UI Design
Decide on a consistent theme, design and color scheme.
Create a wireframe for the web app in Figma for the web app.
Create a non-functional prototype with dummy data for the web app.
Create a wireframe for the web app in Figma for the native app.
Create a non-functional prototype with dummy data for the web app.
UX Design
Integrate phone camera to scan barcodes.
Use location services to track user's location.
Integration with wearable devices.
Work on social media integration.
Work on push notifications.
UX Design Done
Backend Development
Create database schemas.
Handle user authentication.
Create a REST API for the calorie counter.
Create a REST API for the gym tracker.
Merging Functionality
Add functionality to the apps and remove dummy data. (USDA Foo
Handle request errors.
Testing and Deployment
Test the apps on multiple devices and resolve bugs.
Deploy the app on the Google Play Store and App Store.
Deploy the web app on Vercel.

SWOT Analysis

5

SWOT Analysis

S

- Integration with wearable tech
- Detailed analytics

W

- Significant development resources
- Competition
- Limited early adoption

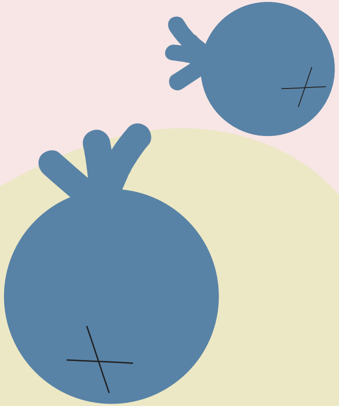
O

- Growing fitness industry and nutrition knowledge
- Subscription models
- Gamification features

T

- Disruption with AI and virtual reality
- Changing market

Assessing Object-Oriented Design Quality



TECHNIQUES

- **Code reviews**

Identify potential design issues.

- **Static code analysis**

Analyze source code without executing it, identify code smells, duplicate code and security vulnerabilities.

- **Automated testing**

Assuring that the design is testable, and it works.

- **Refactoring**

Improving the design of existing code without changing its behavior.

PRINCIPLES

S

Single Responsibility Principle

Classes should be responsible for a single task. Keeps classes small, focused and easier to maintain.

O

Open-Closed Principle

Classes, methods and software entities should be accept new functionality without requiring modification of the existing code.

L

Liskov Substitution

Objects of a superclass should be able to be replaced with objects of a subclass without breaking the system.

I

Interface Segregation Principle

Interfaces should be small and only contain the methods that a client requires, avoid monolithic interfaces that are difficult to understand.

D

Dependency Inversion Principle

High-level modules should not depend on low-level modules. Both should depend on abstractions, interfaces or abstract classes.

COMPOSITION vs INHERITANCE

- Classes should achieve polymorphic behavior and code reuse by their composition, rather than inheritance from a base or parent class.

- **Inheritance**

Sub-classes become dependent on parent classes.
Changes could create ripple effects.

- **Composition**

Composing smaller, more focused classes out of functionalities.
Leads to more reusable and extendable code.

* May not always be the best approach. Developers should make the decision