# NotePlan Specifications

## 1. Front Matter

* **Title**: NotePlan
* **Author**: Andrei Bertescu
* **Team**: Individual Developer

## 2. Introduction

### a. Overview

* **One-phrase application definition**:  
  An app where you can centralize all your events and notes.
* **Summary of the Problem**:  
  Users often struggle to maintain coherent notes and tasks without getting lost in cluttered text files or overbuilt note-taking apps. NotePlan helps this by enabling efficient notes and to-dos through an intuitive GUI interface.

### b. Glossary

* **Spring Boot**: A framework for building stand-alone, production-grade Spring-based applications.
* **Spring Security**: A framework providing authentication and authorization capabilities.
* **Hibernate**: An object-relational mapping (ORM) tool for Java.
* **MySQL**: An open-source relational database management system.

### c. Context or Background

* **Why it matters**:  
  Modern digital workflows rely on flexible, fast, and portable note-taking solutions. Existing tools are often either too simple or overly complex.
* **Origin**:  
  Created as a personal productivity tool and as part of my apprenticeship to Generația Tech.
* **Impact**:  
  Increases productivity for developers and note-takers who prefer plain and simple tools.

### d. Goals

* **Product Requirements (User Stories)**:
  + As a user, I want to create structured notes easily by writing.
  + As a user, I want automated task organization.
  + As a user, I want a simple GUI to navigate my notes visually.
  + As a user, I want to assign different colors for my tasks.
* **Technical Requirements**:
  + Web application using Spring Boot.
  + Database integration using Hibernate.
  + Mobile-responsive UI.
  + Configurable settings.

### e. Assumptions

* User has a web browser installed.
* User has an email address to make an account with.
* Basic OS compatibility: Linux, Windows, macOS.

## 3. Solutions

### a. Proposed Solution

* **Components**:
  + MySQL database (for data storage).
  + HTML, CSS, and Thymeleaf for the front-end templating.
  + Spring Boot and Spring Security for the back-end.
* **Pros**:
  + Easy to use.
  + No installation needed.
* **Cons**:
  + GUI is minimal.
  + Requires internet access.
* **Business Logic**:
  + Controllers handle HTTP requests and responses.
  + Services contain business logic for processing data.
  + Repositories interact with the database using Hibernate.
* **Presentation Layer**:
  + Web pages designed for user interaction with events and notes.
  + Responsive design for accessibility across devices.
* **Scalability**:
  + Application can be scaled horizontally using container orchestration tools.
* **Limitations**:
  + Currently lacks features like notifications or integrations with external calendars.

### b. Test Plan

* **Unit tests**: Test individual components like services and repositories.
* **Integration Tests**: Test the interaction between components and the database.
* **End-to-End Tests:** Simulate user interactions with the application.

### c. Monitoring and Alerting

* **Logging** with Spring Boot’s built-in logging system.d. Deployment Plan
* **Deployment Architecture**: Containerized deployment using Docker.
* **Environments:** Development and production environments configured separately.
* **Roll-out**: Manual deployment from a local PC, followed by a cloud hosting service.
* **User Communication**: Documentation provided on the GitHub repo.

### g. Alternate Solutions

* **Alternatives**: Using a monolithic architecture without containerization.
* **Pros:** Easier to deploy.
* **Cons**: Less scalable and harder to maintain.

## 4. Further Considerations

### a. Third-party Services

* None integrated yet.
* Potential future integrations: GitHub issues, Google Calendar (via API).
* No current cost implications.

### b. Security Considerations

* **Threats**: Unauthorized access, data breaches.
* **Mitigation**: Robust authentication and authorization mechanisms.

### c. Operational Considerations

* Implement regular backups and disaster recovery plans.

### d. Risks

* Potential security vulnerabilities.

## 5. Success Evaluation

### a. Impact

* **Security**: Minimal exposure.
* **Performance**: Near-instant operations.
* **Cost**: Free and open-source.

### b. Metrics

* User engagement.
* System uptime.
* Response times.

## 6. Work

### a. Work Estimates

* **Tasks:** Develop Spring Boot app, implement security, design UI, set up database.
* **Resources Needed:** Development tools, hosting services, monitoring tools.
* **Time Estimates:** Approximately 3-4 months for initial development and deployment.

### b. Prioritization

### Security implementation and core functionalities.

### c. Milestones

* Registration method: Nov 5, 2024
* Front-end and back-end fully functional: Nov 20, 2024
* Unit testing and monitoring: Apr 10, 2025
* Deployment using docker: Apr 30, 2025

### d. Future Work

* Implement notifications
* Integrate with external calendars
* Enhance UI/UX.
* Make more accessible for mobile devices.