해커톤 프로젝트 결과 보고서

목차

1. 우리가 사용한 AI 기술

1.1 ChatGPT

- **문서 분석 및 요구사항 도출**: 제안요청서를 분석하여 핵심 요구사항을 도출하는 데 활용
- **MDC 문서 생성**: 요구사항을 바탕으로 MDC 문서 생성

1.2 Readdy

- **UI/UX 디자인 생성**: 화면 디자인 및 코드 생성
- **시각적 컴포넌트 설계**: MDC 문서를 바탕으로 시각적 컴포넌트와 디자인 요소 생성

1.3 Cursor AI

- **코드 생성**: Vue 기반의 FE 코드, Spring-Boot 기반 BE 코드 자동 생성
- **오류 해결 지원**: 코드 작성 과정에서 발생한 오류 해결을 위한 제안 제공

2. 구현 순서 및 내용

이 섹션에서는 프로젝트의 구현 과정을 시간 순서대로 정리하고, 각 단계에서 수행한 작업의 상세 내용을 설명합니다.

2.1 요구사항 분석 및 MDC 문서 생성

- **제안요청서 분석**: ChatGPT를 활용하여 제안요청서의 내용을 분석하고 핵심 요구사항 추출
- **요구사항 도출**: 분석된 내용을 바탕으로 시스템이 갖추어야 할 기능 요구사항 정리
- **MDC 문서 생성**: 요구사항을 구조화하여 MDC 문서 자동 생성
- **문서 구조화**: 개발 방향성과 구현 상세사항을 포함한 체계적인 문서 완성

2.2 UI/UX 디자인 개발

- **Readdy 활용**: MDC 문서를 기반으로 Readdy를 통해 화면 디자인 생성
- **디자인 요소 정의**: 사용자 인터페이스 컴포넌트 및 레이아웃 설계
- **시각적 디자인**: 사용자 경험을 고려한 UI 요소의 디자인 및 인터랙션 설계
- **디자인 코드 추출**: 시각적 요소에 대한 구현 코드 자동 생성

2.3 프론트엔드 코드 생성 및 초기 구현

- **Cursor AI 활용**: Vue.js 기반 FE 코드, Spring-Boot 기반 BE 코드 자동 생성
- **초기 통합 시도**: Readdy에서 생성한 디자인을 Vue 프로젝트에 통합 시도

- **문제 발견**: 디자인 요소가 코드에 제대로 반영되지 않는 문제 확인
- **여러 번의 재시도**: 디자인-코드 통합을 위한 반복적인 시도 수행

2.4 설정 문제 해결 및 코드 최적화

- **원인 파악**: .cursor/rules에 Vue 파일을 추가하지 않은 것이 원인임을 발견
- **설정 수정**: .cursor/rules에 Vue 파일 관련 설정 추가
- **성공적 통합**: 디자인 요소가 Vue 코드에 올바르게 반영되는 것을 확인
- **코드 최적화**: 생성된 코드를 검토하고 필요한 부분 수정

2.5 로그인 기능 구현 및 오류 해결

- **핵심 기능 개발**: 사용자 인증 및 로그인 기능 구현
- **AI 기반 오류 해결**: 발생한 오류에 대해 AI 도구를 활용한 해결책 적용
- **한계 발견**: 일부 오류는 AI가 같은 코드를 반복 제안하는 문제 발생
- **하이브리드 접근**: 필요한 경우 직접 소스 코드를 검토하고 수동 수정 수행

2.6 AI 중심 개발 방법론 적용

- **최소 인간 개입**: 최대한 AI 도구를 활용하고 사람의 개입을 최소화하는 원칙 적용
- **도구 간 연계**: ChatGPT, Readdy, Cursor AI 등 각 도구의 산출물을 효과적으로 연계
- **자동화 중심**: 문서 생성부터 코드 구현, 오류 해결까지 AI 기반 자동화 프로세스 구축 시도
- **효율성 검증**: AI 중심 개발 방법론의 효율성과 한계점 검증

3. Trouble Shooting 내용

3.1 Readdy와 Cursor AI 연계 문제

- **디자인 반영 실패**: Readdy에서 생성한 디자인을 Cursor AI가 무시하는 문제 발생
- **여러 번의 재시도**: 디자인 요소를 제대로 반영하기 위해 여러 차례 코드 생성 시도
- **해결 방법**: .cursor/rules에 Vue 파일을 추가하여 문제 일부 해결

3.2 AI 기반 오류 해결의 한계

- **반복적인 오류 해결 제안**: AI가 동일한 해결책을 반복적으로 제시하는 문제
- **무한 반복 코드**: 오류 해결 과정에서 동일한 코드를 반복적으로 생성하는 현상
- **해결 방법**: 일부 오류는 직접 소스 코드를 검토하고 수동으로 수정

3.3 도구 간 일관성 유지

- **산출물 간 불일치**: 각 AI 도구에서 생성된 산출물 간의 일관성 부족
- **통합 과정의 어려움**: 서로 다른 AI 도구의 결과물을 하나의 프로젝트로 통합하는 과정에서 발생한 충돌
- **해결 방법**: 통합 지점을 명확히 정의하고 단계별 검증 과정 추가

4. 배운 점

4.1 AI 도구의 가능성과 한계

- **AI 기반 개발의 효율성**: AI 도구를 활용하면 개발 초기 단계의 작업 속도를 크게 향상시킬 수 있음
- **맥락 이해의 중요성**: AI가 전체 프로젝트의 맥락을 이해하도록 명확한 지시와 정보 제공이 필요함

- **오류 해결의 한계**: 복잡한 오류 상황에서는 여전히 인간의 직관과 경험이 중요한 역할을 함

4.2 AI 도구 간 통합의 교훈

- **도구 선택의 중요성**: 각 AI 도구의 특성과 장단점을 이해하고 적절한 단계에 활용하는 것이 중요함
- **인터페이스 표준화**: 도구 간 산출물을 효과적으로 연계하기 위한 표준 인터페이스 정의가 필요함
- **검증 단계 필요**: AI 생성 결과물에 대한 중간 검증 단계가 최종 품질에 큰 영향을 미침

4.3 향후 AI 기반 개발 개선점

- **규칙 파일 최적화**: .cursor/rules와 같은 AI 도구의 동작을 제어하는 설정 파일의 중요성 인식
- **프롬프트 엔지니어링**: 더 효과적인 결과를 얻기 위한 프롬프트 작성 기술 개발 필요
- **하이브리드 접근법**: AI 도구와 인간 개발자의 강점을 결합한 하이브리드 개발 방법론 구축
- **프로젝트 수행 방법론 개선**: 요구사항 분석 후 API 문서를 먼저 작성하고, 이를 기반으로 프론트엔드와 백엔드 개발을 분리하여 진행했다면 더 효율적인 개발이 가능했을 것임