요리용 코딩 언어 개발

중간 보고서

첫눈에

202255661 박혜연

201924419 김도엽

201924577 정종현

목차

1. 요구조건 및 제약 사항 분석에 대한 수정사항

1.1 기존 요구 조건

a) 사용자 친화적 UI : 블록 기반 인터페이스를 활용하여 사용자가 드래그 앤 드롭 방식으로 요리과정을 직관적으로 설계할 수 있어야 한다.

b) 요리용 코딩언어 개발 : 블록 UI에서 입력된 레시피 정보를 JSON 형식으로 구조화하여 백엔드에서 파싱, 저장, 재사용 가능하도록 한다. 블록 조립 방식과 JSON 데이터의 규칙을 통해 요리 과정을 코드화하여 순차적으로 실행, 조건처리, 반복 처리 등이 가능하도록 설계한다.

c) 레시피 생성 및 저장 : 블록 기반으로 생성한 레시피를 저장, 수정, 불러오기 가능하도록 한다.

d) 레시피 개인화: 로그인 인증을 통해 유저마다 작성하는 레시피가 구분되어 레시피의 개인화를 보장한다

1.2 기존 제약 사항 분석

a) 재료의 이름은 한글로만, 시간과 양은 숫자로만, 단위는 블록에 지정되어 있는 걸 이용한다. 특수기호는 기입 불가하다.

b) 모든 동작을 다 표현할 수 없으므로, 잘 쓰이는 동작들로만 구성한다.

c) 재료를 사용자가 추가하므로, 재료에 관한 오류 검증이 제한적이다.

1.3 요구조건 및 제약 사항 분석에 대한 수정사항

요구 조건에 변경된 사항 없다. 하지만 제약사항 분석에 재료 블록 생성 시 개발자가 제공한 토글 목록 안에서 태그를 선택하도록 한다"라는 내용을 추가한다. 태그에 따라 재료가 들어갈 블록을 정하기 위해서이며, 사용자가 직접 생성하면 오류 검증 범위가 너무 넓어지므로 개발자가 제공하는 토글 목록에서 선택하게 만든다.

2. 관련 연구 조사

a) Scratch

Scratch는 MIT Media Lab에서 2003년에 개발한 블록 기반의 시각적 프로그래밍 언어

수학, 과학, 기술 등의 STEM 교육 분야에 활용되며, 학생들이 문제 해결 능력, 추상화, 절차적 사고력을 배울 수 있도록 돕는다.

50개 이상의 언어를 지원하고, 비주얼 언어로 구성되어 있어 프로그래밍에 익숙하지 않은 사용자도 쉽게 접근할 수 있다.

b) Entry

Entry 는 한국에서 개발된 교육용 프로그래밍 언어 및 플랫폼으로, 초등학생과 중학생을 대상으로 프로그래밍 교육 및 창의적 사고력 향상을 목적으로 설계되었다.

본 논문에서는 Entry 를 활용한 협업 기반 스토리 창작 활동이 학생들의 창의성, 문제 해결 능력, 협업 능력 향상에 긍정적인 영향을 미쳤음을 실험을 통해 보여주었다.

그림 1: 스크래치 예시

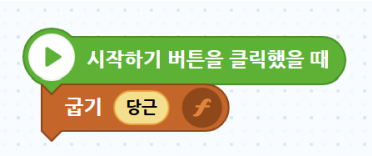

그림 2: 엔트리 예시

entry와 scratch는 코딩을 교육적으로 접근하여 누구나 쉽게 사용할 수 있도록 설계되었다. Blockly는 이에 기반하여 요리라는 주제를 더한다. 프로그래밍에 익숙하지 않은 사용자도 본인의 레시피를 블록 형태의 코드로 저장하여 쉽게 접근할 수 있도록 한다. scratch와 Entry가 토대로 한 Blockly를 사용하여 새로운 시스템을 만들려고 한다.
기존의 시스템은 요리를 위해서 만들어지지 않았다. 그러므로 우리는 요리를 위해 특화된 시스템을 만들려고 한다. scratch나 entry는 재료 블록을 새로 생성할 수 없었고, 직접 타이핑하여서 넣어야하는 점이 있었는데, 이런 점은 실수를 유발할 수 있다. 또한 동작블록도 사용자가 일일이 다 지정해야하는 점에서 레시피를 생성할 때마다 새로 생성해야하는 불편함이 있었고, 또한 새로 만들 때 정의된 동작이 달라질 수 있었다. 이러한 점을 생각하여 비슷하지만 새로운 시스템을 만들려 한다.

c) Blockly

Blockly는 Google에서 개발한 웹 기반의 오픈소스 시각적 프로그래밍 라이브러리로, 사용자가 블록 형태의 UI를 조합함으로써 코딩 로직을 작성할 수 있도록 돕는다. JavaScript, Python 등 주요 언어로 코드 변환이 가능하며, 드래그 앤 드롭 방식으로 블록을 조립하여 직관적인 프로그래밍 경험을 제공한다.
본 졸업과제는 요리용 코딩언어 개발에서 부터 현 블록기반 코딩으로 보다 사용자에게 친숙한 방식으로 레시피를 구성할 수 있도록 하는것이 주 목표가 되어왔다.

Blockly는 다음과 같은 측면에서 연관성을 가진다.

비전문가 대상 UI: 코딩 지식이 없는 사용자라 할지라도 블록모양을 보고 논리 구조를 이해하며 직관적인 레시피 제작을 할 수가 있다.
블록기반 시각화: 요리 레시피의 단계적 로직(ex: 물을 끓인다 → 라면 수프를 넣는다→ 라면 면을 넣는다)을 블록을 조합하여 시각화가 가능하다.
중첩 구조 표현력: 조건문과 반복문 같은 프로그래밍 구조를 블록화 하여 요리 순서의 분기나 반복을 표현 가능하게 한다.

현재 적용 가능한 범위

Blockly는 위에 명시 되어 있듯이 자바스크립트등의 주요언어로 코드 변환이 가능하기에 현 React환경내에 연동이 가능함을 확인하였다. 다음과 같이 실현 가능하다.
Blockly workspace를 React 내에 삽입 및 사용자 상호작용 구현
기본 블록 정의 및 커스텀 블록 생성
블록 조합에 따른 레이아웃 자동 조정 기능 (블록 중첩 시 크기 확장 등)
제한점 및 현실적 어려움
사용자 정의 블록 제한성: 많은 동작에 대한 사용자 정의 블록에 대한 커스텀 제작엔 로직이 동사에 따라 무궁무진하게 달라질 가능성이 많아 시간적 제한이 많이 들것으로 예상되어 해당 부분은 제한
UI/UX 한계: 블록 간 간격 조정, 직관적인 블록 색상 및 아이콘 설정 등에서 추가적인 사용자 경험 개선이 필요함
데이터 연동: 블록 조합 결과를 서버에 전송하고, 이를 기반으로 요리코드 변환을 수행하려면 백엔드와의 데이터 스키마 설계가 선행되어야 함

출처: https://developers.google.com

d) 레시피 블록 기반 시각화

제목 : A new recipe format for cooking (RecipeBlocks)
저자 : Clement Lo
출처 : [Introducing RecipeBlocks - A new visual recipe format for cooking | UX Collective]
주요 아이디어 : 재료 블록과 조리 단계 블록(Method Blocks)을 연결해 레시피의 흐름을 시각적으로 표현
졸업과제와의 연관성 :
RecipeBlocks는 블록을 이용한 시각적 레피시 포맷으로,
재료블럭과 동작블럭 등 구성요소를 명확히 분리하여 레시피를 구성하였다는 점
시작블럭과 종료블럭을 통해서 레피시 흐름(순차,병렬 처리)을 한눈에 보여둔다는 점
위와 같은 이유로 저희 졸업과제(요리용 코딩언어 개발)에서 연관성을 가지지만
문법이 정의되어 있지 않아 구문 오류나 모호성 등을 방지하지 못하는 점
시각화용 표현만 있을 뿐 이를 동작하기 위한 코드 생성기가 없다는 점
등이 졸업과제에 전체적으로 적용하기엔 아쉬운 점이다.

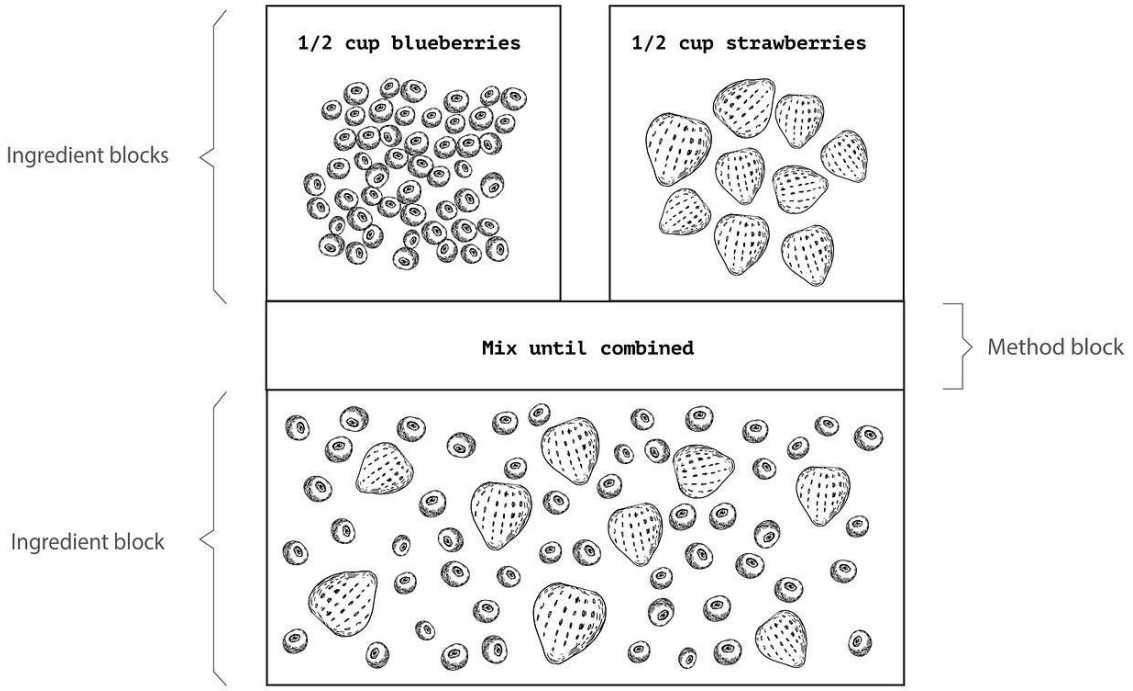

그림3: 레시피 블록 시각화 예시

3. 설계 상세화 및 변경 내역

3.1 변경 내역

a) 관리자 기능 구현 삭제

개인이 코드를 짜는 용으로 쓰기 때문에 관리자 기능이 필수적이지 않다고 판단했다.
정보를 많이 받지 않기 때문에 관리자가 할 일이 많이 없다.
관리자 기능 중 핵심인 블록 관리 기능(동작블록 생성, 수정, 삭제)등을 웹에서 구현하지 않는다.

b) 버전 관리 삭제

저장공간 사용을 줄이기 위해 삭제한다.
코드를 쓰기 시작한 지점부터 계속 저장하고 있어야 하므로 낭비가 심하다.
Blockly에서 기본적으로 undo, redo를 지원한다.

c) 자동 저장 삭제

서버 비용과 부담을 증가시키므로 삭제한다.
Blockly에서 기본으로 제공하는 undo, redo가 있으므로 삭제한다.

3.2 설계 상세화

3.2.1 피그마 화면 구성도

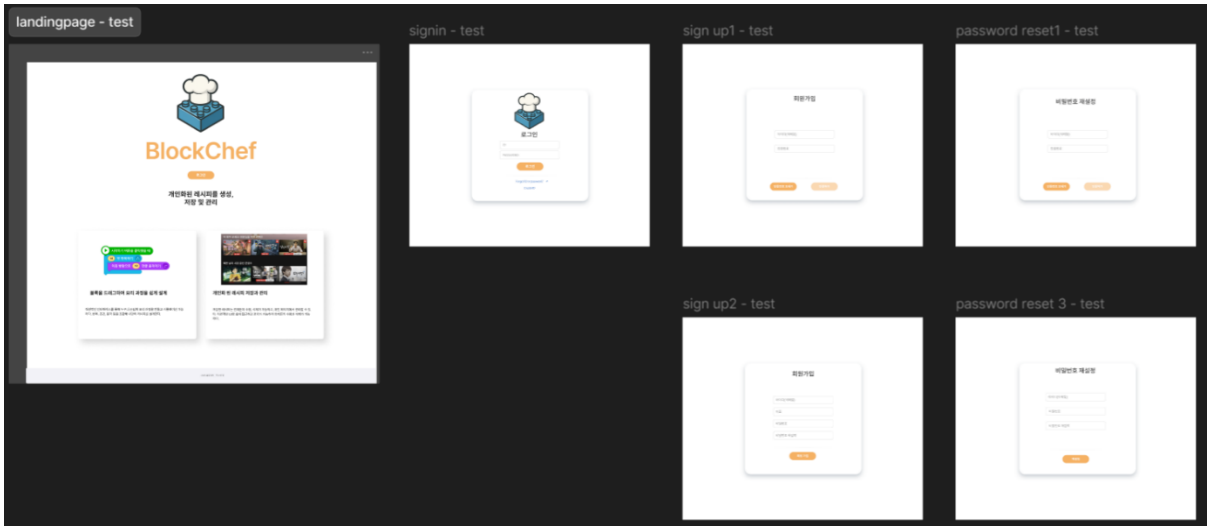

그림 4: 피그마 인증/인가 화면

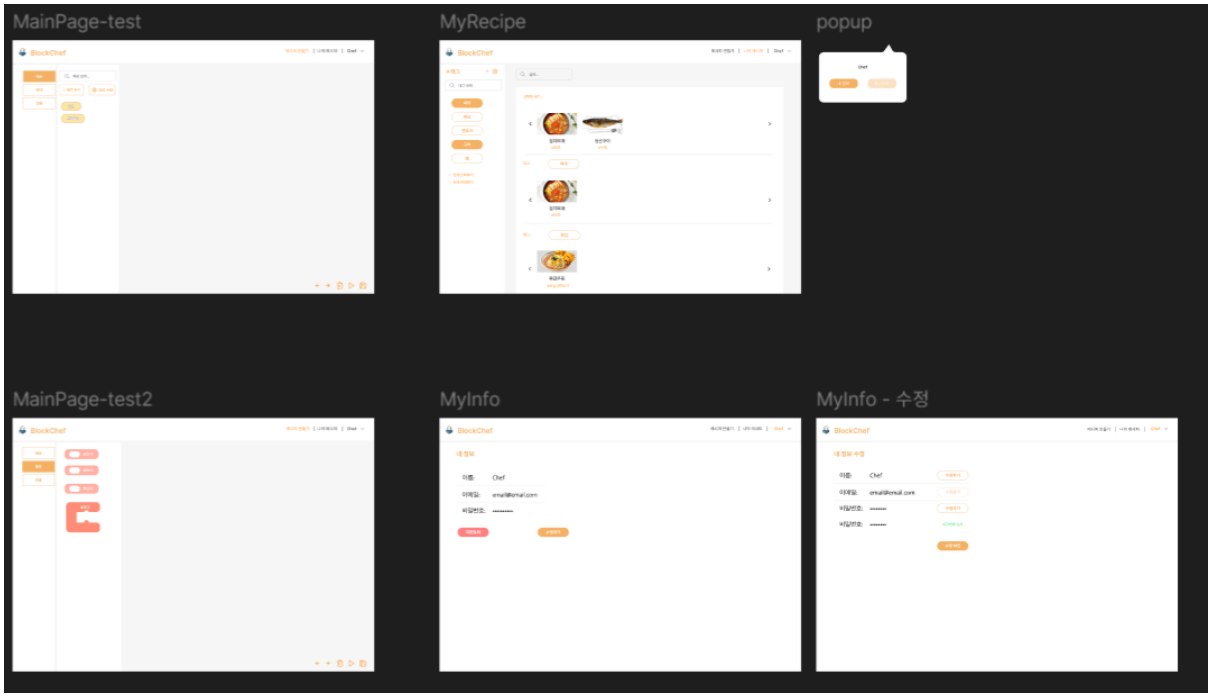

그림 5: 피그마 메인페이지/마이레시피 화면

3.2.2 API 명세서

기능	HTTP 매서드	도메인
로그인	POST	auth/login
회원가입	POST	auth/sign
회원가입 이메일 인증 코드 전송	POST	auth/sign/signup/send-code
비밀번호 재설정 이메일 인증 코드 전송	POST	auth/email/reset-password/send-code
이메일 확인	POST	auth/email/verify-code
비밀번호 재설정	POST	auth/reset-password
내 레시피 전부 보기	GET	/{userId}/myRecipe
내 레시피 하나 보기	GET	/{userId}/myRecipe/{recipeId}
내 레시피 만들기	POST	/{userId}/myRecipe
레시피 수정하기	PUT	/{userId}/myRecipe/{recipeId}
레시피 삭제하기	DELETE	/{userId}/myRecipe/{recipeId}
내 정보 조회하기	GET	/{userId}
내 정보 수정하기	PUT	/{userId}
회원 탈퇴 하기	DELETE	/{userId}

4. 갱신된 과제 추진 계획

업무	7월				8월				9월			
프론트 개발(메인페이지)												
백엔드 개발(메인페이지, 레시피 도메인)												
테스트												
블록 설계 및 구현												
프론트 개발(마이 레시피, 마이 페이지)												
백엔드 개발(마이 레시피, 마이 페이지)												
개선사항 분석												
최종테스트												
시스템 개선												
최종보고서 작성												
발표 준비												

5. 구성원별 진척도 및 어려움

5.1 구성원 별 친척도

학번/이름	구성원별 진척도
201924419 김도엽	User 도메인 구현 로그인, 회원가입 구현 이메일 인증 구현 배포
202255661 박혜연	DB 연결 Swagger 구현 및 정적 배포 API 명세서 작성 Figma 디자인
201924577 정종현	Figma 디자인 랜딩페이지 구현 로그인, 회원가입 페이지 구현 이메일 인증 페이지 구현

5.2 구성원 별 어려움

201924419 김도엽

a) 이메일 인증 로직의 예외 처리 구조 설계 어려움

문제 : 설계에서 이메일 인증을 진행하는 상황이 두 가지가 있습니다. 회원가입시에 DB에 아이디가 없어야 이메일 인증을 진행하는 경우, 비로그인 상태에서 비밀전호 재설정 시에 DB에 아이디가 있어야 이메일 인증을 진행하는 경우입니다. 처음에는 단순히 이메일을 인증한다는 것에만 신경써서 두가지 상황에 동일한 sendVerificationCode() 로직을 재사용하면서 에러가 났었습니다.

해결 : 인증 메서드를 기능별로 sendSignupVerificationCode, sendResetPasswordVerificationCode 등으로 분리하고 내부적으로 공통 로직은 sendVerificationCode로 분리하여 코드의 중복을 줄였고, 호출 전 유효성 검사 로직은 각 매서드에서 담당하도록 분리하였습니다.

b) GitHub Actions를 이용한 CI/CD 자동 배포 실패

문제 : GitHub Actions 에서 EC2에 Spring Boot 애플리케이션을 자동 배포하는 과정에서 application-secret.properties 파일이 누락되어 배포된 코드가 정상 동작하지 않는 문제가 발생했습니다.

해결 : .gitignore 에 의해 로컬에서는 존재하던 파일이 GitHub에는 반영되지 않아 생긴 문제였습니다. 이를 해결하기 위해 application-secret.properties 파일의 내용을 GitHub Secrets에 등록하고 deploy.yml에서 그 값을 사용하여 배포 시점에 생성하도록 하였습니다.

202255661 박혜연

swagger를 작성할 때 NoResourceFoundException 에러가 났다. 처음에는 이유를 몰라서 계속 찾다가, @RestControllerAdvice랑 부딪혀서 swagger와 open-api가 작동하지 않았다는 것을 알게 되었다. RestControllerAdvice의 코드 내에 swagger관련된 건 무시하도록 코드를 변경했지만, 그럼에도 오류는 계속 되었다.

코드를 변경 전에는

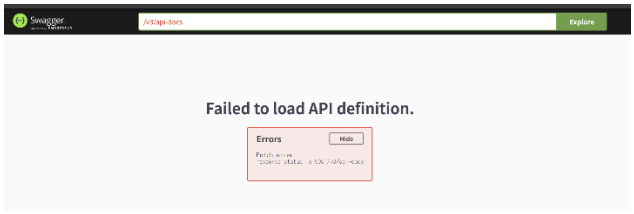

그림 6: 스웨거 로딩 오류 화면

{"code":"UNKNOWN_ERROR","message":"예상치 못한 오류가 발생했습니다."}

그림 7: 스웨거 오류 메시지

```
@ExceptionHandler(Exception.class)  0개의 사용위치  doyeop *
public ResponseEntity<ErrorResponse> handleGeneralException(Exception ex, HttpServletRequest reque
    String path = request.getRequestURI();
    if (path.startsWith("/v3/api-docs") || path.startsWith("/swagger-ui")) {
        throw new RuntimeException(ex);  // Spring 기본 처리로 넘김
    }

    ex.printStackTrace(); // 콘솔에 전체 스택 트레이스 출력 (개발용)

    return ResponseEntity
            .status(HttpStatus.INTERNAL_SERVER_ERROR)
            .body(new ErrorResponse( code: "UNKNOWN_ERROR",  message: "예상치 못한 오류가 발생했습니다."));
}
```

그림 8: 예외 처리 코드 변경 화면

이 코드로 변경하니

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Jul 15 22:29:53 KST 2025
There was an unexpected error (type=Internal Server Error, status=500).

그림 9: 에러 페이지 화면

이렇게 뜨기 시작했다. 하지만 여전히 안되었다. 그래서 찾은 방법이 @Hidden 어노테이션으로, swagger와 같은 곳에서 무시하는 동작을 하는 어노테이션이었다. 이걸 붙이니 평소에는 정상 동작하고, swagger에서만 무시된다.

201924577 정종현

a) React 개발 환경 구축의 어려움

문제 상황: HTML, CSS, JavaScript는 다룰 수 있었으나 React는 처음 접하는 기술로, 초기 개발 환경을 구성하는 데 많은 어려움이 있었다.

원인 분석: React의 구조와 설정 방식, 컴포넌트 기반 아키텍처에 대한 이해 부족으로 인해 생긴 문제였다. 특히 npx create-react-app, npm, node_modules 등 관련 도구의 사용과정에서 에러가 자주 발생하였다.

해결 방법: 공식 문서와 커뮤니티 자료를 참고하면서 버전 충돌 문제를 인식하게 되었고, Node.js 및 패키지의 버전을 다운그레이드 또는 업그레이드하여 안정적인 환경을 구축하였다.

b) TailwindCSS 도입과 스타일 적용 혼란

문제 상황: 기존에는 인라인 CSS나 외부 스타일시트를 사용했기 때문에, TailwindCSS의 유틸리티 클래스 기반 스타일링 방식이 낯설었다.

원인 분석: 스타일 속성을 클래스명으로 일일이 지정하는 방식이 초반에는 비효율적으로 느껴졌고, 구조 이해 없이 적용하려다 보니 CSS가 의도대로 작동하지 않는 경우도 많았다.

해결 방법: TailwindCSS의 공식 문서를 참고하고, 작은 단위의 UI부터 적용해보며 실습을 병행함으로써 점차 익숙해졌고, 이후에는 작업 효율이 크게 향상되었다.

c) 컴포넌트화 과정에서의 구조 설계 문제

문제 상황: 각 페이지에 들어가는 요소를 공통 컴포넌트로 구성하려 했지만, 디자인 요소(간격, 크기, 색상 등)가 페이지마다 미묘하게 달라 재사용성이 떨어졌다.

원인 분석: 디자인 기준이 픽셀 단위로 다르게 적용된 점과, 컴포넌트 설계 시 커스터마이징 가능성에 대한 고려가 부족했던 것이 주요 원인이었다.

해결 방법: 공통 레이아웃은 ShadowBox, InputField 등의 컴포넌트로 분리하고, 각 컴포넌트가 props를 통해 유연하게 스타일을 바꿀 수 있도록 구조를 개선하였다. 반복 수정이 필요했던 부분은 별도 유틸 함수를 만들어 적용 중이다.

d) 유틸 컴포넌트 적용 시 중복 렌더링 및 기능 오류

문제 상황: 유틸성 컴포넌트를 여러 페이지에서 재사용하면서 의도치 않게 중복 렌더링되거나 로직이 충돌하는 문제가 발생하였다.

원인 분석: 컴포넌트 내부 상태 관리 및 렌더링 타이밍에 대한 이해가 부족했고, 공통 컴포넌트를 설계하면서 전역 상태 또는 props 전달 구조가 명확하지 않았다.

해결 방법: 컴포넌트 설계를 단순화하고, 불필요한 상태 값을 제거하였다. 또한 useEffect, useState 훅의 의존성 관리와 조건부 렌더링을 통해 의도한 로직대로 작동하도록 수정하였다.

6. 보고 시점까지의 과제 수행 내용 및 중간 결과

a) GitHub: https://github.com/orgs/firstsnow25/repositories

b) Swagger : https://blockchef-swagger.netlify.app/

c) Figma : https://www.figma.com/design/ATVy8haEJiOXVJb0Rc4yea/BlockChef?node-id=0-1&t=ob6lAM7JZIhqvF5N-1

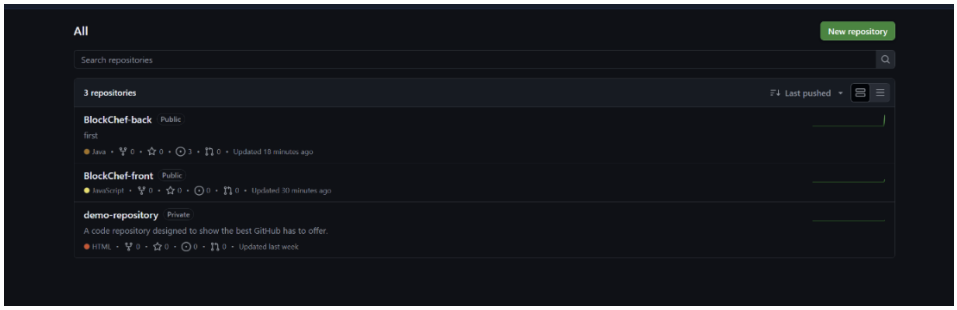

그림 10: 깃허브 레포지토리 화면

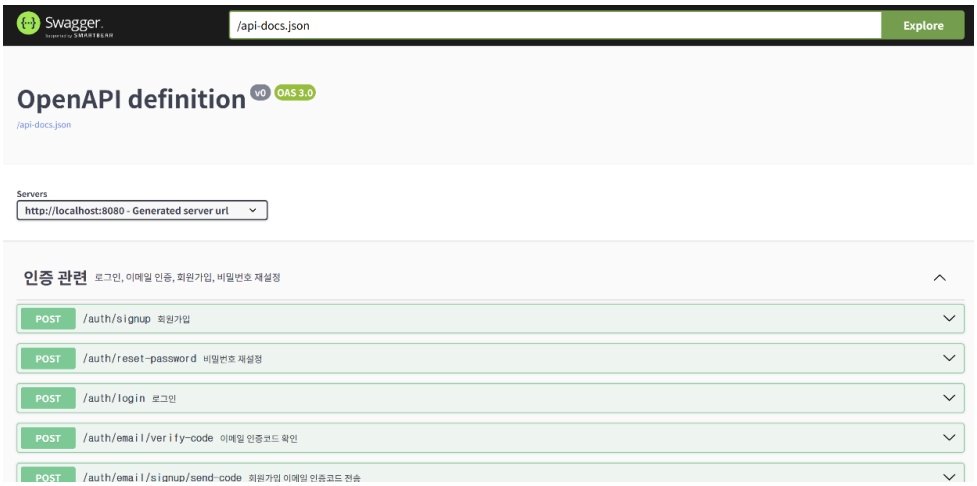

그림 11: swagger api 명세서 화면

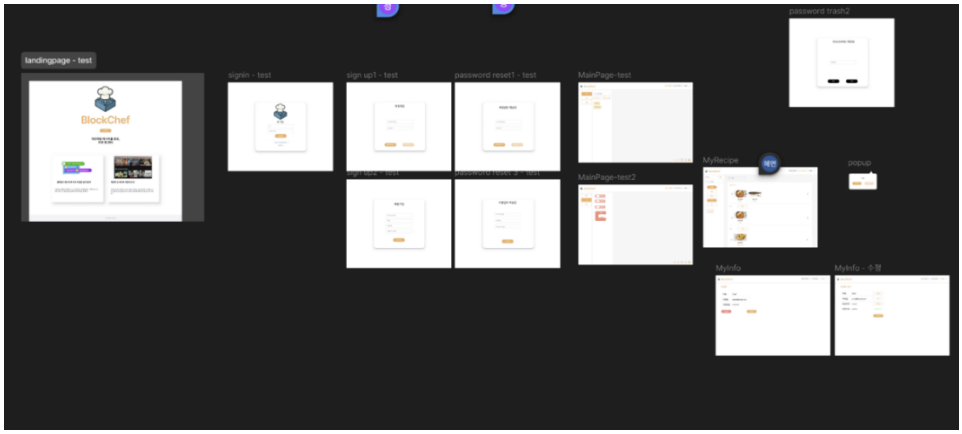

그림 12: 피그마 화면

d) 구현된 페이지

랜딩페이지

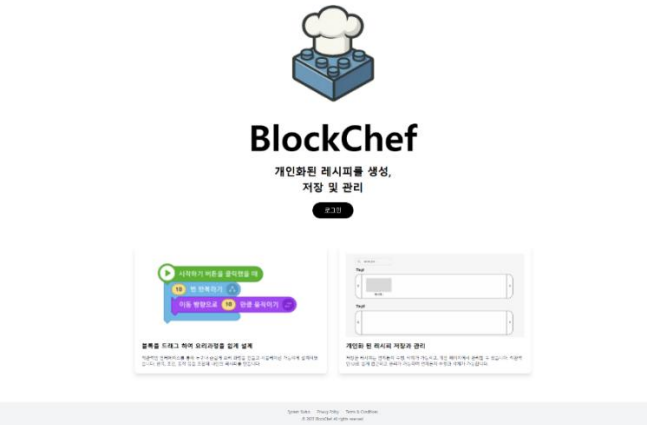

그림 13: 랜딩페이지 화면

인증/인가 (로그인, 회원가입) 페이지 + 이메일 인증

그림 14: 로그인 화면

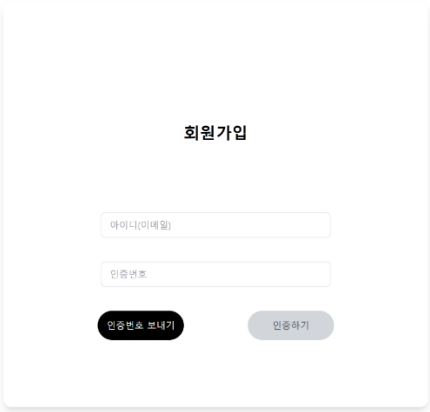

그림 15: 회원가입 화면

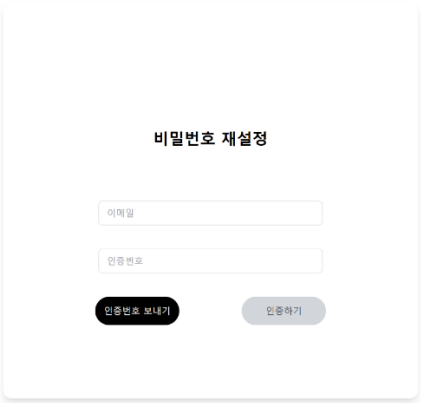

그림 16: 비밀번호 재설정 화면