HW4. Software Architecture Term Project Proposal

20171677 소프트웨어학부 이정하

시스템 목표

Vision

도서관에 항상 관리자가 있지 않지만 관리자가 없는 시간에도 도서관을 이용하고 싶은 사람이 많다. 그리고 이용자가 많아지면서 이를 수기로 작성하여 관리하는 것이 어려워졌다. 도서관의 많은 도서 정보와 사용자가 대출, 반납하는 기록 등 여러 데이터를 전산으로 관리하게 되었다. **이 시스템은 도서관 이용자가 도서관 관리자가 근무하지 않는 때에도 언제든지 도서를 대출하고 반납할 수 있도록 한다.**

Scope

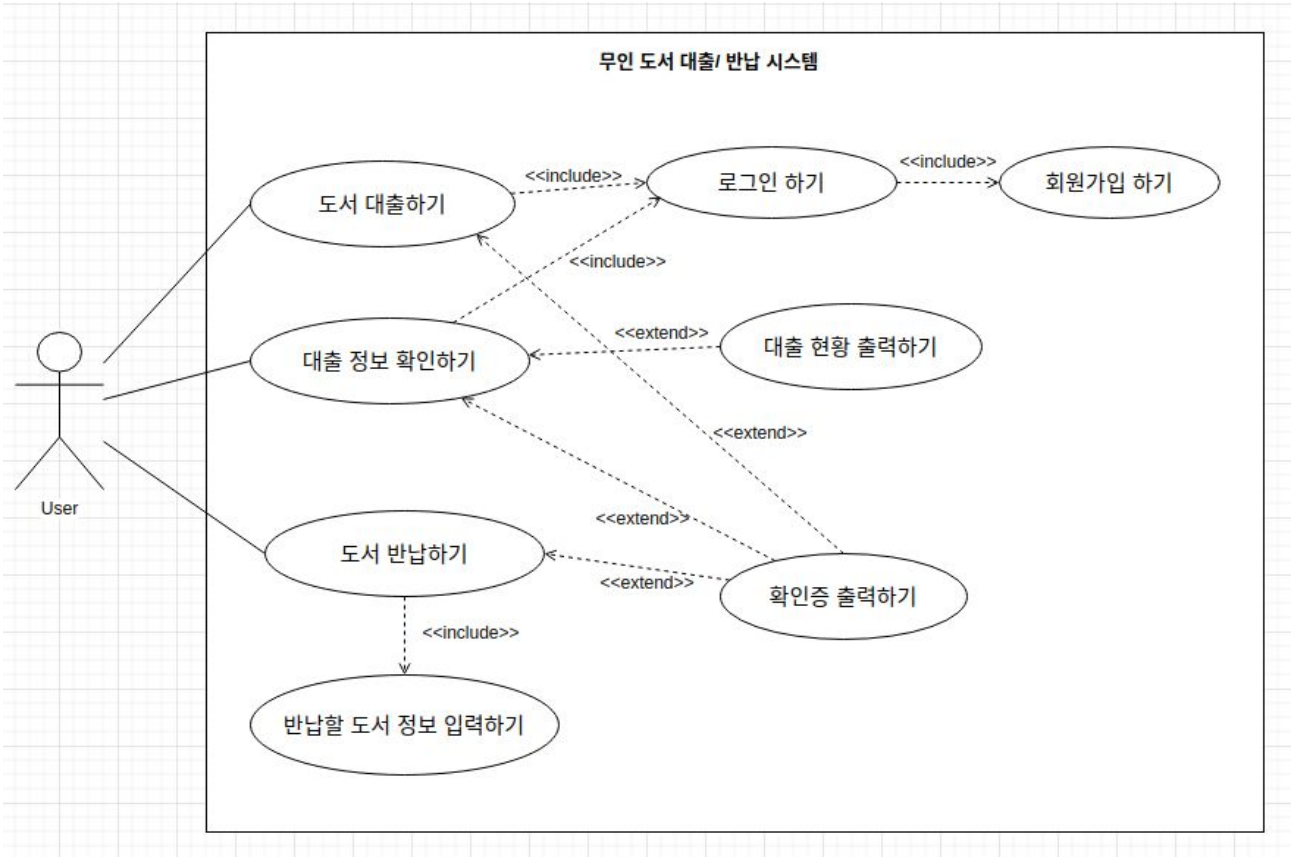
우리 주변의 도서관을 가면 어디서든지 **"무인도서 대출/반납기"**를 쉽게 찾아볼 수 있다. 이 시스템에서는 대학교에 있는 도서관으로 학생들만 이용할 수 있는 도서관의 "무인도서 대출/반납기"(이하 무인기)로 범위를 좁혀서 설계한다. 그리고 사용자 입장에서 무인기의 전산시스템을 이용하여 도서 대출, 반납, 자신의 대출 기록 확인할 수 있도록 설계한다. 관리자는 어드민 권한으로 전산시스템에서 전반적인 도서관 상황을 확인할 수 있도록 한다.

시스템 설명

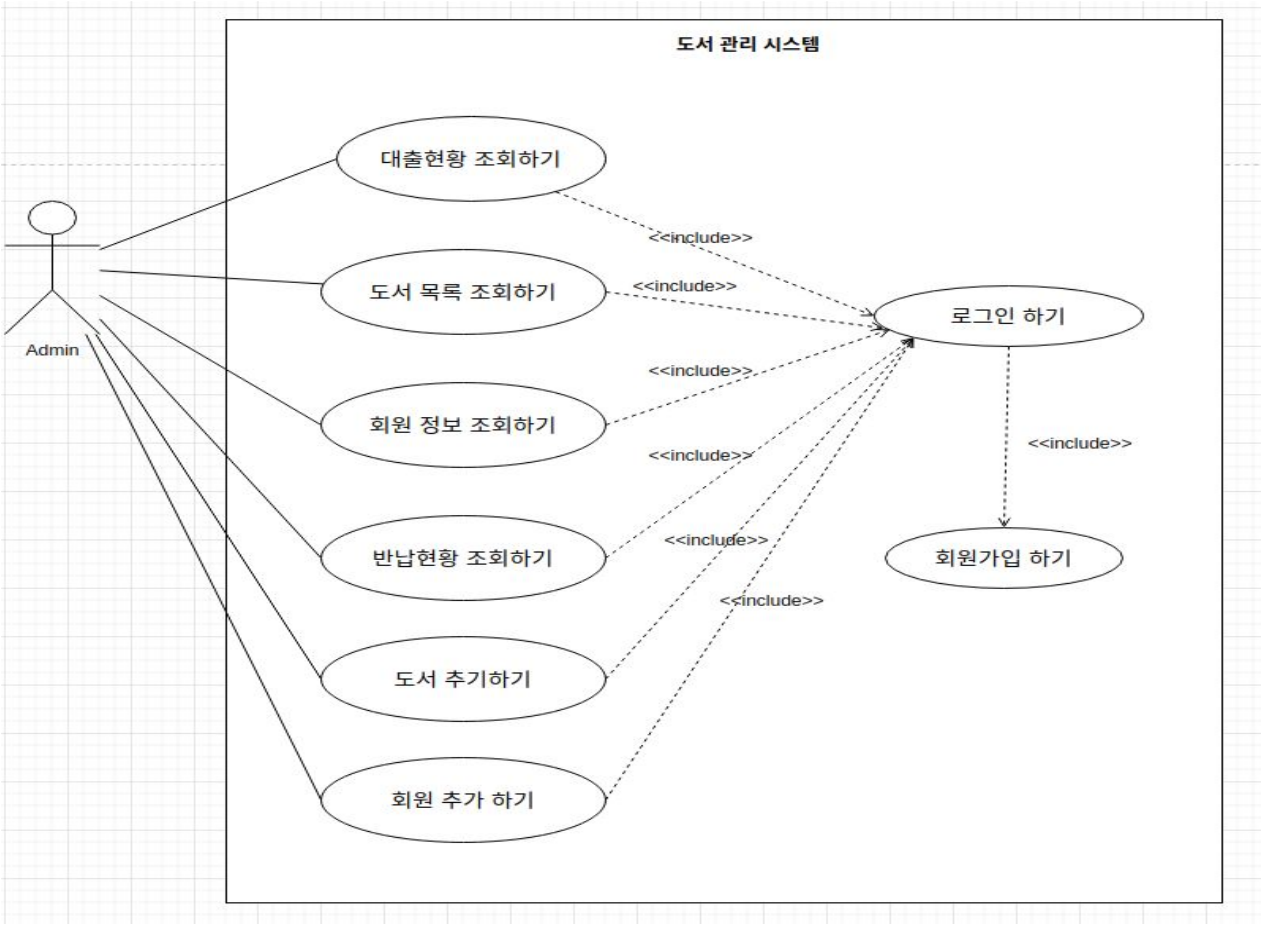
지난 객체지향분석 및 설계수업때 개인 프로젝트로 "무인 도서 대출/반납 시스템"을 Python으로만 구현하였으며 DB, 사용자가 사용하기 편리한 UX/UI, 서버를 따로 구축하지 않았다. 그리고 관리자 시스템은 구현하지 않았었다. 그래서 이번 프로젝트에서는 이를 소프트웨어아키텍처 스타일로 더 탄탄하게 설계하고 관리자와 일반 사용자의 기능을 모두 구현하여 이전 프로젝트의 부족한 점을 보완할 것이다.

	이전 프로젝트	이번 프로젝트
UX/UI	UX/UI 없이 콘솔창으로 출력	React.js를 이용하여 웹 프론트엔드 개발예정
DB	파일 입출력으로 대체	Firebase DB를 이용하여 Serverless Web service개발 예정
Server	없음	
Architecture Style	R&R을 확실하게 구분하려고 노력함.	Serverless Architecture(BaaS), MVC Architecture

시스템 개요 및 특성



[그림 1] User usecase



[그림 2] Admin usecase

Functional Requirement 설명

- **User**

	내용	중요도 (상/중/하)
FR1	화면에 “대출”, “반납”, “대출정보” 버튼을 출력하여 보여준다.	하
FR2	로그인하라는 메시지를 화면에 출력한다.	하
FR3	입력받은 아이디, 비밀번호가 일치하지 않으면 재입력을 요청한다.	중
FR4	입력받은 아이디, 비밀번호의 번호를 서버로 전송하여 데이터 일치여부를 확인한다.	상
FR5	도서 바코드와 스캐너의 위치가 일치하도록 안내 메세지를 화면에 보여준다.	하
FR6	도서 바코드와 스캐너의 위치가 일치하지 않는다면 위치조정을 위한 메시지를 화면에 출력한다.	하
FR7	반납 시, 해당 도서가 대출된 기록이 없다면 도서 정보에 대한 에러 메세지를 출력한다.	상

FR8	현재 몇 권을 대출했는지, 그리고 현재 대출 가능한 도서가 몇 권인지, 대출한 도서명과 반납 예정일을 화면에 보여준다.	상
FR9	추가 대출을 원하는 지 확인하는 메시지와 함께 “계속”, “완료” 버튼을 화면에 보여준다.	하
FR10	확인증을 받을 것인지 물어보는 메시지와 함께 “예”, “아니오” 버튼을 출력한다.	하
FR11	“예”를 눌렀다면 대출확인증을 출력한다.	하
FR12	도서를 꺼내달라는 안내 메시지를 보여준 후 default page로 돌아간다.	하
FR13	추가 반납을 원하는 지 확인하는 메시지와 함께 “계속”, “완료” 버튼을 화면에 보여준다.	하
FR14	“예”를 눌렀다면 반납확인증을 출력한다.	하
FR15	사용자 이름, 현 대출권수, 도서명, 대출일, 반납예정일을 화면에 표로 출력한다.	상
FR16	“종료” 버튼을 누르면 default page로 돌아간다.	하
FR17	확인증 용지가 부족할 경우 용지 부족 에러 메세지를 출력한다.	하

- **Admin**

	내용	**중요도 (상/중/하)**
FR1	어드민으로 로그인한다.	상
FR2	아이디, 패스워드 정보를 서버로 보내 일치 여부를 확인한다.	상
FR3	어드민 로그인 정보가 없다면 회원가입 페이지로 보낸다.	하
FR4	조회하고 싶은 날짜, 도서 번호, 도서명을 입력받아 대출현황을 검색한다.	중
FR5	조회하고 싶은 날짜, 도서 번호, 도서명을 입력받아 도서정보를 검색한다.	중
FR6	조회하고 싶은 날짜, 도서 번호, 도서명을 입력받아 반납현황을 검색한다.	중
FR7	조회하고 싶은 학번, 이름을 입력받아 회원 정보를 검색한다.	중
FR8	검색하고 싶은 정보를 서버로 해당 데이터를 요청한다.	상
FR9	검색결과가 없는 경우, "일치하는 검색 결과가 없습니다."를 출력한다.	중
FR10	추가하고 싶은 도서 정보를 입력하여 도서 정보를 추가한다.	중
FR11	추가하고 싶은 회원 정보를 입력하여 회원 정보를 추가한다.	중

FR12	기존의 도서 정보가 있다면, "해당 도서는 이미 존재합니다."를 출력한다.	중
FR13	기존의 회원 정보가 있다면, "해당 회원은 이미 존재합니다."를 출력한다.	중

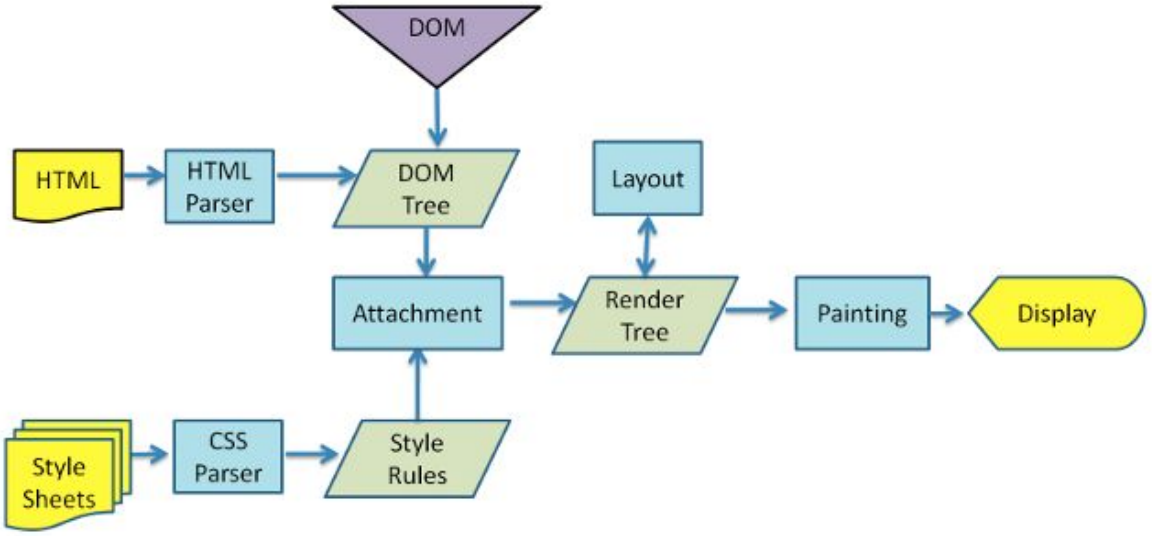
*노란색으로 색칠된 칸은 Alternative Scenario입니다.

Non-Functional Requirement

	내용	품질 속성 내용	품질 속성	중요도 (상/중/하)
NFR1	시스템은 사용자의 별도 조작 없이 갑작스레 종료되지 않아야 한다.	시스템은 사용자의 조작없이는 종료되서는 안 된다.	Stability	상
NFR2	시스템은 로그인 정보를 신속하게 정확하게 전달해야한다.	시스템은 3초 이내에 로그인 정보를 확인하고 결과를 보여야한다.	Accurateness, Security	상
NFR3	시스템은 이용자의 클릭에 즉각적으로 반응해야한다.	사용자가 버튼을 클릭하면 0.3초 이내에 해당하는 클릭 영역에 대한 정보를 인식하고 다음을 진행한다.	Time Behavior	하
NFR4	시스템은 오류가 났을 시 서버에 무조건 보고한다.	오류 발생시, 서버에 자동으로 전송한다.	Analyzability	상
NFR5	시스템 부팅시 오류가 없는지 확인해야한다.	전원이 들어옴과 동시에 오류를 확인한다.	Analyzability	중
NFR6	추가 대출을 위해 이전의 과정으로 신속하게 돌아간다.	추가 대출을 위해 이전의 과정으로 돌아가는 것을 3초 이내에 실행해야한다.	Time Behavior	하
NFR7	추가 반납을 위해 이전의 과정으로 신속하게 돌아간다.	추가 반납을 위해 이전의 과정으로 돌아가는 것을 3초 이내에 실행해야한다.	Time Behavior	하
NFR8	모든 일을 완료한 후 main page로 빠르게 돌아간다.	모든 일을 완료한 후 main page로 3초 이내에 돌아간다.	Time Behavior	하

사용하고 하는 기술 및 Framework

HW3에서 조사한 Chorimium multiprocessing architecture를 조사하였다. Chromium에서 실행시킬 수 있는 웹 브라우저를 개발하고 싶어 이 프로젝트를 고안하게 되었다. 그래서 지난 과제에서 조사하였던 대상 시스템에 추가적으로 브라우저의 작동 원리를 조사하였다.



[그림 3] 브라우저의 workflow

- **DOM Tree 생성**
  브라우저가 HTML을 전달받으면 브라우저의 렌더 엔진이 이를 파싱하고 DOM 노드로 이뤄진 트리를 만든다. 각 노드는 각 HTML 엘리먼트들과 연관되어있다.
- **Render Tree 생성**
  그리고 외부 CSS 파일과 각 엘리먼트의 inline 스타일을 파싱한다. 스타일 정보를 사용하여 DOM 트리에 따라 새로운 트리, 렌더 트리를 만든다.
  Webkit에서는 노드의 스타일을 처리하는 과정을 'attachment'라고 부른다. DOM트리의 모든 노드를 'attach' 메소드가 있고 이 메소드는 스타일 정보를 계산해서 객체형태로 반환한다. 이 과정은 동기적인 작업이며 DOM트리에 새로운 노드가 추가되면 그 노드의 attach 메소드가 실행된다. 렌더 트리를 만드는 과정에서 각 요소들의 스타일이 계산된다. 그리고 이 계산되는 과정에서 다른 요소들의 스타일 속성들을 참조한다.
- **Layout (reflow)**
  렌더 트리가 다 만들어지고 나면, 레이아웃 과정을 거치게 된다. 각 노드들은 스크린의 좌표가 주어지고, 정확히 어디에 나타나야 할 지 위치가 주어진다.
- **Painting**
  렌더링된 요소들에 색을 입히는 과정이다. 트리의 각 노드들을 거쳐가면서 paint() 메소드를 호출하고 스크린에 원하는 정보가 나타난다.

Web		**Framework, Library**
Frontend		React.js
Backend	DB	Firebase DB
	Server	

DOM에 변화가 생기면, 렌더 트리를 재생성하고 레이아웃을 만들고 페인팅하는 과정이 다시 반복된다. 복잡한 SPA(Single Page Application)에서는 DOM조작이 많이 발생한다. 즉, 변화를 적용하기 위해 브라우저가 많이 연산을 해야한다는 의미이며 전체적인 프로세스를 비효율적으로 만든다.

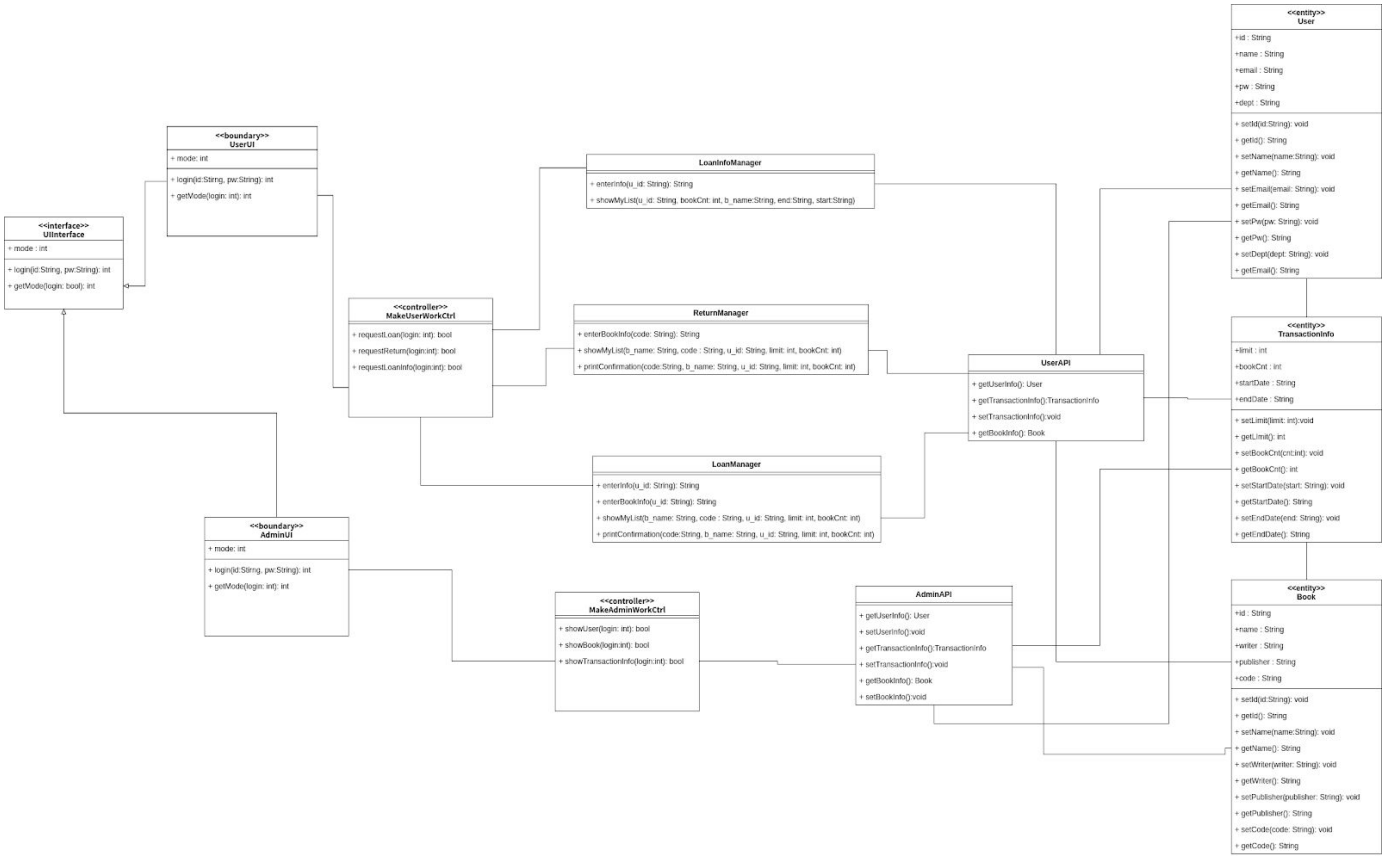
이 문제를 해결하기 위해 웹프론트엔드는 React,js를 사용할 것이다. 뷰가 변화가 있다면, 그 변화는 실제 DOM에 적용되기 전에 가상의 DOM에 먼저 적용시키고 그 최종적인 결과를 실제 DOM으로 전달한다. 이로써, 브라우저 내에서 발생하는 연산의 양을 줄이면서 성능을 개선하게 된다.

웹 백엔드는 Google Firebase DB를 이용하여 Serverless Architecture로 설계하였다. Google Firebase DB는 대표적인 BaaS(Backend as a Service)이다. BaaS는 앱 개발에 있어 필요한 다양한 기능들을 API로 제공해줌으로써 서버를 개발하지 않고서도 필요한 기능을 쉽고 빠르게 구현할 수 있도록 해주고, 비용은 사용한만큼 나가게 된다. 그리고 서버의 이용자가 늘어나게 되어도 알아서 확장이 된다.

BaaS를 사용함에 있어서 개발 시간의 단축, 서버 확장 작업의 불필요하다는 장점을 가지고 있다. 특히, Firebase에서는 실시간 데이터베이스를 사용하여 데이터가 새로 생성되거나, 수정되었을 때 소켓을 사용하여 클라이언트에게 바로 반영시켜주는 기능이 있어 짧은 기간 내에 개발해야하는 지금과 같은 상황에 아주 적합하다고 판단하였다.

소프트웨어 시스템 구성도 및 설명

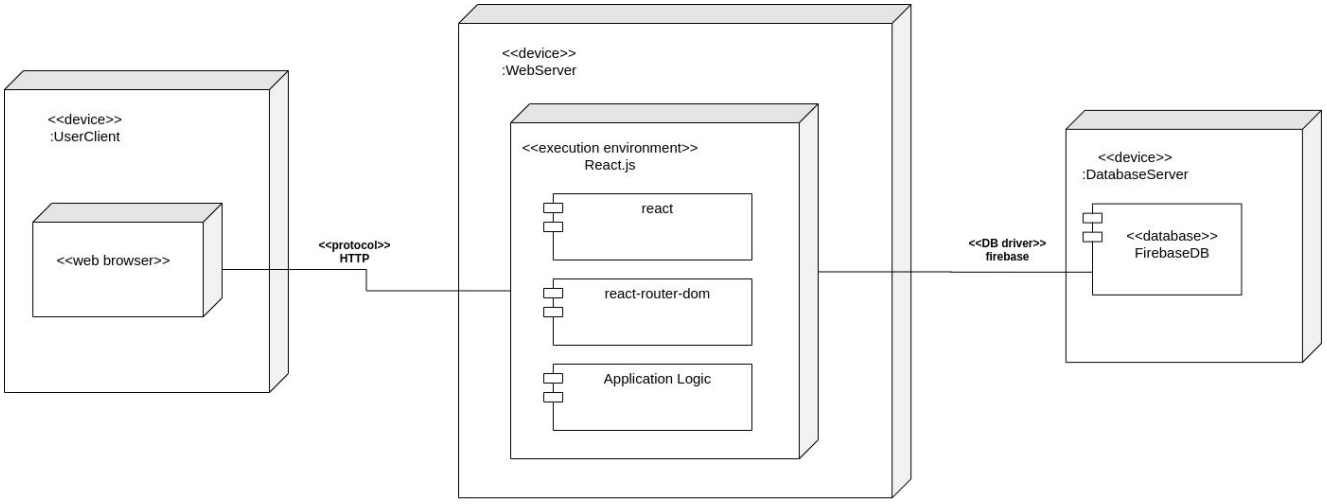
- Class Diagram



[그림 4] 무인 도서 대출,반납시스템 Class Diagram

클래스 다이어그램에서는 구체적인 flow를 보이기 위해 구체적으로 작성하였다. 크게 일반 사용자(User)와 관리자(Admin)으로 나누었다. 일반사용자는 UI를 이용하여 로그인하고 이후 대출, 반납, 대출현황 확인 서비스를 사용할 수 있다. 그리고 이 작업들의 요청을 기반으로 controller가 작업에 맞게 manager class를 호출한다. manager 클래스는 해당하는 정보를 얻기 위해 API를 호출하여 데이터에 접근한다. 대략적인 flow는 관리자도 동일하나 관리자는 controller 클래스에서 요청하는 작업에 따라서 바로 관리자 API를 호출하여 데이터에 접근하여 추가, 수정하는 기능까지 작업할 수 있다.

- Deploy view



[그림 5] 무인 도서 대출,반납 시스템 Deploy Diagram

deploy view를 보이기 위해 deploy diagram을 이용하여 표현하였다. deploy diagram으로 대략적인 프로그램의 흐름을 설명한다. 클라이언트는 웹 브라우저를 이용하여 서비스를 이용할 수 있다. 그리고 웹 브라우저는 프론트엔드는 React.js로 구현되며 백엔드에서 서버와 DB는 Firebase를 이용함으로써 Serverless로 구현되어있다. React.js에서 firebase모듈을 이용하여 Firebase DB에 접근하여 데이터를 사용할 것이다. 이 다이어그램에서는 MVC Architecture, Serverless Architecture에 초점을 두어 설계하였다.

사용하는 Software architecture

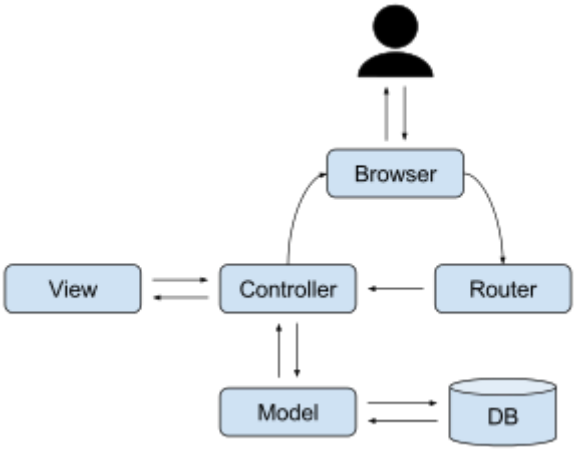
- **Serverless architecture**

  Serverless Architecture는 타사 서비스에서 애플리케이션을 호스팅하는 소프트웨어 디자인 패턴으로, 개발자가 서버 소프트웨어 및 하드웨어 관리를 할 필요가 없다. 응용프로그램은 개별적으로 호출하고 확장할 수 있는 개별 기능으로 나뉜다.

  인터넷에서 소프트웨어 응용 프로그램을 호스팅하려면 일반적으로 일종의 서버 인프라를 관리해야한다. 일반적으로 이는 관리해야하는 가상 또는 물리적 서버뿐만 아니라 응용 프로그램을 실행하는 데 필요한 운영체제 및 기타 웹 서버 호스팅 프로세스를 의미한다.

클라우드 제공 업체의 가상 서버를 사용한다는 것은 물리적 하드웨어 문제를 제거하는 것을 의미하지만 운영체제 및 웹 서버 소프트웨어 프로세스의 일부 수준의 관리가 여전히 필요하다.

- **MVC Architecture**



[그림 4] 웹 어플리케이션에서 일반적인 MVC구성요소 다이어그램

모델-뷰-컨트롤러는 응용프로그램을 세 가지의 구성요소로 나눈다. 각각의 구성요소들 사이에는 다음과 같은 관계가 있다.
컨트롤러는 모델에 명령을 보냄으로써 모델의 상태를 변경할 수 있다. 또, 컨트롤러가 관련된 뷰에 명령을 보냄으로써 모델의 표시 방법을 바꿀 수 있다.
모델은 모델의 상태에 변화가 있을 때 컨트롤러의 뷰에 이를 통보한다. 이와 같은 통보를 통해서 뷰는 최신의 결과를 보여줄 수 있고, 컨트롤러는 모델의 변화에 따른 적용 가능한 명령을 추가, 제거, 수정할 수 있다. 어떤 MVC 구현에서는 통보 대신 뷰나 컨트롤러가 직접 모델의 상태를 읽어오기도 한다.
뷰는 사용자가 볼 결과물을 생성하기 위해 모델로부터 정보를 얻어 온다.

시스템 아키텍트로써 고려하고 있는 사항

웹 프론트엔드의 경우, React.js를 이용하여 페이지를 컴포넌트 단위로 관리할 것이다. 그리고 각 컴포넌트는 R&R을 기준으로 나눈다. 이를 통해 Row coupling, High Cohesion한 코드가 될 것이다. 결과적으로, 재사용성을 높이고 유지보수하기 쉬운 코드를 될 것이다.
웹 백엔드의 경우, Firebase만을 이용하여 DB와 Server를 동시에 해결하는 Serverless Architecture로 설계하였다. Firebase DB는 json구조로 데이터를 저장한다. 그렇기 때문에 일반적인 DB설계와 다르게 DB 기능과 동시에 API역할도 하므로 스키마를 설계하는 것이 어렵게 느껴졌다. 그러므로 DB설계에 좀 더 시간을 써야할 것으로 예상하고 있다. Architecture측면에서는 서버와 DB의 경계가 모호하지만 MVC Pattern에서 Model, Controller를 Firebase가 하고 있다고 생각하였다.

참고문헌

https://velopert.com/3543
http://i-bada.blogspot.com/2015/10/backend-as-service-baas.html
https://www.twilio.com/docs/glossary/what-is-serverless-architecture
https://d2.naver.com/helloworld/59361
https://ko.wikipedia.org/wiki/%EB%AA%A8%EB%8D%B8-%EB%B7%B0-%EC%BB%A8%ED%8A%B8%EB%A1%A4%EB%9F%AC
https://raptor-hw.net/xe/know/17236