МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна система каталогізації бібліографічних описів з інтегрованим репозиторієм»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Волк Ю.Ю.**

Студента групи ІН.мз-91с — **Семенової А.В.**

СУМИ 2020

Сумський державний університет
(назва вузу)

Факультет *ЦЗДВФН* Кафедра *Комп'ютерних наук*
Спеціальність *«Інформатика»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Семеновій Анастасії Володимирівні
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна система каталогізації бібліографічних описів з інтегрованим репозиторієм*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи)

_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз предметної області. 2) Моделювання структури WEB-додатку 3) Конфігурація серверного оточення та налаштування засобів віртуалізації 4) Розробка інформаційної системи каталогізації бібліографіч-них описів з інтег-рованим репозиторієм.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

_____
_____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання
_____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.			
2.			
3.			
4.			
5.			

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 74 стор., 35 рис., 10 табл., 2 додатки, 20 джерел.

Об’єкт дослідження – інформаційні системи каталогізації та обліку.

Предмет дослідження – автоматизована бібліотечна інформаційна система з інтегрованим репозиторієм для приватного користування.

Мета роботи – розробка інформаційної системи каталогізації бібліографіч-них описів з інтегрованим репозиторієм.

Методи дослідження – технології віртуалізації в UNIX-системах.

Результати – розроблено автоматизовану бібліотечну інформаційну систему з інтегрованим репозиторієм для приватного користування. Систему розроблено у формі веб-додатку на базі PHP фреймворку Laravel.

АВТОМАТИЗОВАНА БІБЛІОТЕЧНА ІНФОРМАЦІЙНА СИСТЕМА, DOCKER, PHP, NGINX, MYSQL, LARAVEL.

ЗМІСТ

ВСТУП **6**

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ **8**

1.1 Огляд freeware та enterprise систем каталогізації бібліографічної інформації 8

1.2 Огляд фреймворків для реалізації веб-застосунків 16

1.3 Огляд засобів віртуалізації та оркестрації веб-процесів 20

2 ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ПРОГРАМНОГО ПРОДУКТУ **23**

2.1 Моделювання структури web-додатку 23

2.2 Проектування бази даних 26

2.3 Проектування UI 28

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ **31**

3.1 Конфігурація серверного оточення 31

3.2 Налаштування засобів віртуалізації 32

3.3 Процес розробки 36

3.4 Конфігурація інтерфейсу системи засобами HTML/CSS фреймворку BootStrap 39

3.5 Мануальне тестування програмного продукту 48

ВИСНОВКИ **56**

СПИСОК ДЖЕРЕЛ ПОСИЛАННЯ **57**

ДОДАТОК А **59**

ДОДАТОК Б **60**

ВСТУП

Стрімкий розвиток інформаційних технологій за останні десятиліття дозволив максимально полегшити побут людей, автоматизувавши більшість рутинних процесів. А можливість зберігання великих масивів інформації та їх швидкої обробки виключають необхідність її запам’ятовування та записування на матеріальних носіях. Ці зміни дали значний поштовх у еволюції бібліотечної справи. З кінця минулого століття почали з’являтись перші публічні інтернет-каталоги та автоматизовані бібліотечні інформаційні системи. У багатьох країнах бібліотеки вже протягом декількох десятиліть користуються автоматизованими системами каталогізації, циркуляції та управління бібліотечними фондами. На сьогоднішній день, вже існують ресурси, що пропонують свої рішення для автоматизації бібліотечних систем, зокрема, для приватного використання. Проте, всі вони мають свої переваги та недоліки, а також, в більшості випадків є платними. Розробка закритої автоматизованої інформаційної бібліотечної системи для приватного використання дозволяє інвентаризувати домашній бібліотечний фонд, полегшує пошук наявних книг та дозволяє відстежувати їх місцезнаходження.

Метою роботи є розробка інформаційної системи каталогізації бібліографічних описів з інтегрованим репозиторієм. Для досягнення поставленої мети були сформульовані і вирішені наступні завдання:

- аналіз предметної області;
- проектування та моделювання програмного продукту;
- розробка бази даних;
- реалізація інтерфейсної частини та структури програмного продукту
- тестування готового продукту.

Для досягнення зазначених вище завдань було використано методи як технології віртуалізації в UNIX-системах, методи мануального тестування програмних продуктів, скриптова мова php.

Практична значущість роботи полягає в тому, що було розроблено автоматизовану бібліотечну інформаційну систему з інтегрованим репозиторієм для приватного користування. Систему розроблено у формі веб-додатку на базі PHP фреймворку Laravel. В системі передбачено наступний функціонал:

- додавання книг;
- пошук серед наявних книг;
- можливість перегляду електронної версії (в разі її наявності);
- можливість додавання читачів;
- можливість відслідковувати місцезнаходження книг (включаючи їх видачу зареєстрованим читачам).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Автоматизовані бібліотечні інформаційні системи (БІС) беруть свій початок з кінця 1970-х років; їх було розроблено на базі систем, створених для роботи з однією або кількома дискретними функціями в бібліотеках. Зокрема, в Ріса в Нідерландах та BLCMP у Великій Британії були створені як системи каталогізації. Це були, по суті, кооперативні системи, покликані полегшити тягар каталогізації шляхом спільного використання записів та ресурсів. ALS та система LIBERTAS виникли як прості системи видачі бібліотек, але з часом розширились, включивши інші функції. До цих ранніх систем незабаром приєдналися за призначенням специфічні інтегровані системи управління бібліотеками, іноді спрямовані на конкретні типи таких бібліотек, як школи, коледжі чи університети [1].

1.1 Огляд freeware та enterprise систем каталогізації бібліографічної інформації

Перші автоматизовані БІС в основному використовувалася для управління книгообігом, адже допомагали зменшити випадки зникнення книг та відслідковували прострочення, допущені читачами бібліотек. У наступні два десятиліття функціональність автоматизованих БІС зростала одночасно з комп’ютерними технологіями. Вони стали спроможні обробляти не лише операції з управління книгообігом, але й комплектування, каталогізації та серіалізації. Наприкінці 1990-х популярність Інтернету призвела до розробки Інтернет-каталогів загального доступу (OPAC). Інтернет-каталоги дозволили користувачам здійснювати пошук у фондах бібліотек, переглядати історію своїх видач та здійснювати резервацію книг. На початку XXI століття системи автоматизації бібліотек значно покращили продуктивність роботи бібліотек [2].

Електронний бібліотечний каталог OPAC

Публічний каталог інтернет доступу (OPAC) є однією з основних технологій автоматизованих БІС, яка забезпечує доступ до усієї інформації, що міститься в записі елемента в бібліотеці. OPAC можна визначити як базу даних бібліографічних записів, що описують фонди бібліотеки. Це дозволяє користувачам шукати документ за авторами, заголовками, темою та ключовими словами з терміналу, а також дозволяє друкувати, завантажувати або експортувати записи за допомогою різних електронних засобів. Таким чином, OPAC надає користувачам засоби пошуку та доступу до інформації. Користувачі можуть бачити колекції та статус видачі кожного документа бібліотеки, а також можуть зарезервувати та поновити документ, що їх цікавить, за потреби. Кілька користувачів можуть одночасно здійснювати запити до бази даних, на відміну від традиційного каталогу карток. Бібліографічні записи зберігаються в базі даних і можуть бути швидко отримані для відображення на комп'ютерних терміналах. Розвиток OPAC дозволив користувачам, а також бібліотечним працівникам легко знаходити та отримувати доступ до бібліотечних ресурсів, не витрачаючи зусиль та часу [3].

Більшість ранніх онлайн-каталогів були доволі примітивними і могли здійснювати пошук на основі записів книгообігу. Вони часто представляли користувачеві простий алфавітний список, в якому міг здійснюватися пошук за ключовими словами або в деякій мірі реляційний пошук. З часом вони перетворились на складніші системи, що включають широкий спектр методів пошуку тексту, зокрема пошук з сортуванням на основі частоти зустрічальності у словниках даних, пошук за словосполученнями та скороченнями, ключовими словами та навіть змістом. Подальшим розвитком OPAC став доступ до різноманітних послуг, окрім основних бібліографічних даних. Зокрема, доступ до інформації про спільноту, години роботи, місце розташування тощо. Наступним кроком розвитку стало створення веб-версій, які зазвичай мають більшість функціональних можливостей декстопної версії [1].

Автоматизована БІС «Librarika»

Librarika – це хмарна безкоштовна автоматизована БІС,. Завданням розробників було створення максимально простого сервісу, яким зможуть користуватися школи, університети, бібліотеки, підприємства, організації та будь-який користувач інтернету для створення власної бібліотеки. Ресурс автоматично збирає обкладинки та бібліографічну інформацію з доступних джерел по всьому світу та відкритий для пропозицій користувачів щодо покращення. Слід зазначити, що безкоштовним є лише базовий план ресурсу, що містить рекламу, не дає можливості редагувати інтерфейс та обмежений у кількості записів. Платні плани відрізняються лише кількістю додавання записів. Слід зазначити, що благодійним та неприбутковим організаціям може безкоштовно надаватись більший спектр послуг.

На офіційному сайті зазначено наступні особливості Librarika:

- вбудований публічний каталог інтернет доступу (OPAC);
- можливість миттєвого додавання книжок за допомогою введення ISBN;
- необмежена кількість адміністраторів;
- необмежена кількість користувачів;
- можливість створення бібліотек самообслуговування для невеликих офісів;
- додаток для iPhone, що дозволяє сканувати книги;
- логін для користувачів;
- історія відвідувань;
- можливість відслідковувати видачу та здачу книжок;
- онлайн резервація; автоприєднання користувачів;
- можливість проведення платежів та зборів;
- створення рецензій та рейтингів;
- зчитування штрих-кодів;
- інтеграція бібліографічних даних з OpenLibrary;

- простий та User-friendly інтерфейс;
- можливість масового імпорту предметів каталогу та користувачів;
- Email нагадування про терміни здачі книжок чи прострочення цього терміну;
- інтеграція віджетів для вебсайтів та підтримка різних мов (beta-версія) [4].

Автоматизована БІС «Evergreen»

Evergreen – це автоматизована бібліотечна інформаційна система з відкритим вихідним кодом, ліцензована за GNU General Public License. Таким чином, система може безкоштовно завантажуватись, використовуватись, переглядатись, модифікуватись та розповсюджуватись. Програмне забезпечення використовується для забезпечення бібліотечних каталогів загальнодоступним інтерфейсом, для управління внутрішніми процесами (видача та здача книжок), придбання нових бібліотечних матеріалів та для обміну ресурсами між бібліотеками. Проект було ініційовано Службою публічних бібліотек штату Джорджія в США у 2006 році, що потребувала створення масштабного каталогу для спільного користування більше ніж 275 бібліотеками. Зважаючи на природу автоматизованих БІС, Evergreen має цікаву суміш функціональних можливостей, адже вона одночасно є пошуковою системою метаданих; механізмом обробки транзакцій; веб-додатком та базується на надійному, масштабованому фреймворку, що передає повідомлення – OpenSRF.

Особливістю ресурсу є те, що вся технічна документація розміщена на сайті у вільному доступі та постійно оновлюється. Крім того, Evergreen має власну bug-трекінгову систему, де будь-який користувач може повідомити про недоліки в роботі ресурсу [5].

Автоматизована БІС LibLime Koha

LibLime Koha – це веб-базована автоматизована бібліотечна інформаційна система з відкритим кодом, що не потребує встановлення програмного забезпе-

чення на комп’ютер чи наявності у бібліотек власних серверів. Ресурс має наступні особливості:

- просту у використанні політику обігу, інтуїтивну навігацію та розширені дозволи для облікових записів персоналу;
- функцію «Клуби та послуги», що дозволяє бібліотекам керувати читацькими групами, книжковими клубами та іншими спільнотами;
- широка підтримка резервування, включаючи опцію призупинення та повторної активації резервування, можливість дозволити співробітникам реорганізувати чергу резервувань;
- вдосконалені правила політики відповідності для тегів 001 та 035, що дозволяє бібліотекам оновлювати старіші записи новою версією;
- можливість масового скасування бібліотеками імпорту з каталогу одним клацанням миші, замість того, щоб видаляти їх поодинці;
- конфігурація SIP2 для широкого кола постачальників та їх продуктів та використання EzProxy як подвійного джерела автентифікації для віддаленого доступу до бази даних;
- публічний каталог інтернет доступу, персонал, адміністративні функції та інтерфейси ссамостійного оформлення замовлень базуються на стандартах, що відповідають стандартам технологій World Wide Web - XHTML, CSS та Javascript [6].

Автоматизована БІС Biblionix

Biblionix – платний ресурс, що використовує автоматизовану БІС Apollo, розроблену практикуючими бібліотекарями для публічних бібліотек. Публічний каталог є адаптивним та однаково виглядає на різних пристроях (комп’ютері, телефоні, планшеті, Mac тощо). Користувачі мають можливість інтегрувати вміст каталогу на власний веб-сайт та імпортувати дані своєї бібліотеки в обліковий запис Apollo. Ресурс є веб-базованим, не потребує наявності сервера у

користувачів або встановлення програмного забезпечення та має наступні функції:

- Управління обігом: впорядкований інтерфейс реєстрації; текстові електронні та телефонні сповіщення; загальний каталог та можливості віртуального консорціуму за допомогою VersaCat Sharing та VersaCard Ad Hoc Consortia; автоматичні дзвінки для отримання прострочених та резервних сповіщень (що не потребують будь-якого обладнання); самостійне оформлення замовлення, що потребує лише наявності комп'ютера.
- Управління колекціями: придбання; видалення елементів без втрати інформації; масове видалення.
- Інші переваги: «Запитайте бібліотекаря»/Gabbie – двосторонні текстові повідомлення, з наявною консоллю Curbside Console; посилання для членів сім'ї; звіти для відстеження успіху/невдачі повідомлень/електронної пошти/телефонних дзвінків; онлайн реєстрація; інтеграція зі сторонніми продуктами/послугами з інформацією про доступність [7].

Автоматизована БІС Follett

Follett – освітній ресурс, складовою якого є Destiny Library Manager, що є веб-базованим рішенням для управління бібліотеками. Підтримується усіма пристроями (ПК, планшетами, телефонами), а основними функціями є пошук, читання, книгообіг, каталогізація та інше. Destiny Library Manager забезпечує:

- єдиний вхід за допомогою Baker & Taylor Axis 360, MackinVIA та OverDrive, що надає можливість відкривати сторонній вміст із Destiny, не вимагаючи додаткової авторизації;
- вбудовану підтримку електронних книг Follett, включаючи функцію створення записів, виділення та обміну повідомленнями на уроках;
- пошук серед як безкоштовних, так і платних баз даних за допомогою One Search;

- можливість додавати затверджені викладачами цифрові веб-сайти за допомогою WebPath Express;
- інтеграцію з Follett Titlewave для зручності придбання та аналізу колекцій [8].

Як видно з тенденції, сучасні автоматизовані БІС здебільшого мають два рівні: веб-базовану систему автоматизації, яка повністю підтримує роботу бібліотеки, та службу виявлення, яка замінює традиційний OPAC. До останніх розробок в цій сфері можна віднести наступні здобутки.

Інтеграція та сумісність: Намагаючись вирішити проблеми інтеграції, спричинені наявністю декількох систем, які керують бібліотечними ресурсами та робочими процесами, автоматизовані бібліотечні інформаційні системи нового покоління візуалізуються через веб-базовані системи, які роблять різні інформаційні системи сумісними через метадані. Наприклад, автоматизована БІС Лібріс Альман має здатність «консолідувати різні системи» та забезпечує функціональність обміну метаданими у високозахищеному середовищі. Автоматизована БІС Сьєрра III відмовилася від своєї старої модульної моделі та інтегрувала всі функції у веб-програму. Використовуючи у більшості випадків API, Сьєрра робить обмін ресурсами більш ефективним для досягнення мети сумісності. Інші продукти, такі як OCLC WorldShare Management (WSM), Serials Solution ’Summon, Open Source Kuali OLE (Відкрите бібліотечне середовище) та Intota теж мають подібні функції та функціональні можливості.

Відкриті архітектури та масштабованість: Під впливом глобалізації даних доступність даних та масштабованість системи є важливими особливостями, що входять до систем наступного покоління. Відкрита архітектура робить систему більш масштабованою: модель сервісно-орієнтованої архітектури (SOA) дозволяє користувачеві приймати розумне рішення на основі своїх можливостей та доступності. Автоматизована БІС Kuali OLE служить повноцінною системою, яка, крім своєї здатності керувати бібліотечними та небібліотечними зу-

силлями, може обслуговувати бібліотеки різного розміру та задовольняти індивідуальні потреби шляхом інтеграції та взаємодії з іншими системами.

Дизайн, орієнтований на користувача: Автоматизовані бібліотечні інформаційні системи наступного покоління запозичили деякі функції від Google та Amazon. Зокрема, Summa – це пошукова система на базі ОС, розроблена бібліотекарами та IT-спеціалістами Данської державної та університетської бібліотеки. Його мета – полегшити процес пошуку ресурсів користувачами. Summa пропонує «інтегрований пошук та сортування за релевантністю» в різних джерелах даних. У ньому використовуються методи пропозицій Google, які допомагають користувачам знаходити відповідні пошукові терміни. По суті, більшість автоматизованих бібліотечних систем включають такі функції, як рейтинг відповідності, фільтрація фасет та групування типів ресурсів. Ще однією серйозною зміною, яку несуть ці нові системи, є дизайн інтерфейсу. Більшість інтерфейсів цих продуктів імітують простоту Google, пропонуючи єдине вікно пошуку.

Хмарні обчислення: з урахуванням різноманітних ресурсів, що співіснують у автоматизованих бібліотечних інформаційних системах, зростає попит на управління різними інформаційними системами. Розвиток технологій зумовлює попит на технічно підготований персонал, який володіє як апаратним, так і програмним забезпеченням. Хмарні обчислення дещо вирішили цю проблему. Завдяки доступним мережам великої ємності, недорогій апаратній частині та просторі для зберігання даних та архітектурі, орієнтованій на сервіс, більшість постачальників автоматизованих бібліотечних інформаційних систем пропонують хмарні послуги, що дозволяють бібліотекам користуватися перевагами програмного забезпечення як послуги (SaaS). SaaS не тільки допомагає бібліотекам зменшити вартість володіння технологічною інфраструктурою, але також дозволяє бібліотекам сконцентруватись на користувацьких послугах, не турбуючись про обмеження, викликані нестачею кваліфікованого персоналу. Наразі бі-

льшість продуктів наступного покоління, таких як Alma, Sierra, EDS EBSCO Discovery Service) та WSM, пропонують хмарні послуги [2].

1.2 Огляд фреймворків для реалізації веб-застосунків

На ранніх етапах ери веб-розробки всі програми кодувались вручну, що призводило до великої кількості помилок та затрат трудових ресурсів. Для того, щоб покращити цю ситуацію, на початку 2000-х були запроваджені веб-фреймворки. Веб-фреймворк – це в основному інструмент, який допомагає створювати веб-сайти, зменшуючи при цьому вірогідність помилок та затрати часу. Як статичні, так і динамічні веб-сторінки можуть використовувати фреймворки. Обирати фреймворки можна залежно від завдання. Їх можна класифікувати на дві категорії, а саме – на стороні клієнта та на стороні сервера. Клієнтські фреймворки відповідають за впровадження та вдосконалення користувальницьких інтерфейсів у вигляді анімованих функцій, вигадливих макетів тощо. Прикладами фреймворків на стороні клієнта є vue.JS, angular.JS та ember.JS. З іншого боку, серверні фреймворки мають правила та архітектуру і дозволяють створювати багато різних типів сторінок та можуть забезпечувати фактори безпеки веб-сторінок. Прикладами фреймворків на стороні сервера є Django, Zend, Ruby on rails тощо. Спільні особливості обох типів веб-фреймворків охоплюють безпеку, відображення URL-адрес, систему веб-шаблонів, кешування веб-сторінок тощо. Однією з основних проблем у фреймворках є масштабованість та продуктивність. Поведінку ядра неможливо модифікувати чи змінити. Деякі фреймворки не забезпечують достатньої підтримки. Деякі можуть навіть спричинити труднощі для розробників при спробах внести будь-які зміни в основну поведінку фреймворку. Деякі функції фреймворків можуть також впливати на продуктивність та швидкість веб-сайтів. Коли користувач вибирає фреймворк, він повинен бути обережним щодо обмежень. Деякі фреймворки дозволяють веб-серверу ідентифікувати користувачів, які використовують додаток, а також

обмежують доступ до деяких функцій на основі певних критеріїв. Одним із прикладів фреймворку, який забезпечує інтерфейс для створення користувачів та призначення їм ролей, є Drupa.

Базова структура веб-фреймворку складається з моделі, інтерфейсу та контролера. Модель в основному підтримує серверну систему і містить усі рівні логіки даних. Інтерфейс дбає про те, як візуально виглядає сторінка (front end). Контролер перетворює введені дані в команди. Інший вид архітектури – це трирівнева організація. Він має три фізичні рівні, тобто клієнт, застосунок та базу даних. База даних, як правило, RDBMS. Застосунок взаємодіє з клієнтом за допомогою HTTP, а також складається з бізнес-логіки, що працює на сервері. HTML, згенерований прикладним рівнем, запускається веб-браузером, над яким працює клієнт [9].

Laravel – це безкоштовний PHP веб-фреймворк з відкритим кодом, створений Тейлором Отуеллом для розробки веб-додатків за архітектурним зразком модель-вигляд-контролер (MVC). Серед особливостей фреймворку Laravel – модульна система упаковки із спеціальним менеджером залежностей. Структура фреймворку проста для розуміння та потужна; сама структура забезпечує автентифікацію, маршрутизацію, менеджер сеансів, кешування, IoC-контейнер і тонни найбільш часто використовуваних компонентів, а також інструменти міграції баз даних та інтегровану підтримку модульного тестування, всі ці інструменти надають розробникам можливість побудови складних додатків [10].

Ключовими особливостями Laravel є наступні функції.

Бандли (Bandles) – модульна пакувальна система з відповідними функціями, що робить процес додавання до програм зручнішим.

Eloquent ORM (об’єктно-реляційне відображення) – вдосконалена PHP-реалізація шаблону активних записів, що забезпечує водночас внутрішні методи, що забезпечують обмеження на взаємозв’язки між об'єктами бази даних. У Eloquent ORM таблиці баз даних представлені як класи, примірники об’єктів яких прив’язані до окремих рядків таблиці.

Конструктор запитів, що забезпечує більш прямий доступ до бази даних альтернативою Eloquent ORM. Замість безпосереднього написання запитів SQL, конструктором запитів Laravel надано набір класів і методів, здатних будувати запити програмно, чим дозволяється вибір кешування результатів виконаних запитів.

Логіка застосунків, що може бути реалізована за допомогою контролерів або як частина оголошень маршруту. Синтаксис, що використовується для визначення логіки програми, є подібним до до синтаксису, який використовується у фрейморку Sinatra.

Зворотною маршрутизацією визначається взаємозв'язок між посиланнями та маршрутами, що дає можливість подальшим змінам маршрутів автоматично розповсюджуватися у відповідні посилання. У разі якщо посилання створюються за допомогою імен існуючих маршрутів, відповідні єдині ідентифікатори ресурсів (URI) автоматично створюються Laravel.

Restful контролери, що надають необов'язковий спосіб відокремлення логіки обслуговування HTTP-запитів GET та POST.

Автозавантаженням класів забезпечується автоматичне завантаження класів PHP, що не потребує ручного обслуговування шляхів включення. Завантаження за вимогою запобігає включенню зайвих компонентів, тому відбувається завантаження лише фактично використаних компонентів.

Композери шаблонів слугують налаштовуваними логічними кодовими одиницями, які можна виконати при завантаженні шаблонів.

Blade-механізм шаблонування поєднує один чи декілька шаблонів з моделлю даних для створення результатів шаблонів, шляхом переносу шаблонів у кешований PHP-код для підвищення продуктивності. Blade також містить набір власних структур управління, зокрема, умовні оператори та цикли, які внутрішньо зіставляються зі своїми аналогами PHP. Крім того, сервіси Laravel можна викликати із шаблонів Blade, а сам механізм шаблонування можна розширити за допомогою спеціальних директив.

IoC-контейнери дозволяють створювати нові об’єкти, дотримуючись принципу інверсії управління (IoC), в якому фреймворк звертається до специфічного коду програми або завдання, з необов’язковим створенням екземплярів та посиланням на нові відокремлені об’єкти.

Міграціями забезпечуються система контролю версій для схем баз даних, що надає можливість пов’язувати зміни в кодовій базі програми та зміни в макеті бази даних. Як результат, цією функцією спрощується розгортання та оновлення програм на базі Laravel.

Посів баз даних забезпечує спосіб заповнення таблиць бази даних вибраними даними за замовчуванням, які можуть бути використані для тестування додатків або виконані як частина початкового налаштування програми.

Модульне тестування подається як невід’ємна частина Laravel, модульні тести виявляють і запобігають регресії у фреймворці. Модульні тести можна запускати через надану утиліту командного рядка.

Автоматична пагінація спрощує завдання реалізації пагінації, замінюючи звичні підходи до ручного впровадження автоматизованими методами, інтегрованими в Laravel.

Запит форми служить базою для перевірки введення форми внутрішньо прив’язуючими прослуховувачами подій, наслідком чого є автоматизований виклик методів перевірки форми та генерація фактичної форми.

Homestead є віртуальною машиною Vagrant, яка довзволяє розробникам Laravel всіма інструментами, необхідними для прямої розробки Laravel, включаючи Ubuntu, Gulp, Bower та інші засоби розробки, корисні для розробки повномасштабних веб-додатків.

Canvas – видавнича платформа, що працює на Laravel, яка допомагає візуалізувати щомісячні тенденції, бачити, звідки беруться читачі та в який час доби вони вважають за краще читати вміст. Має такі функції, як: Статистика публікацій, Написання без відволікання, Інтеграція без виклику, Спеціальні соціальні дані.

LazyCollection – функція PHP фреймворку Laravel 6, що дозволяє вам мати справу з великими навантаженнями даних, зберігаючи при цьому низьку кількість пам’яті [11].

1.3 Огляд засобів віртуалізації та оркестрації веб-процесів

Docker – це відкрита платформа для розробки, доставки та запуску додатків, що дозволяє відокремлювати програми від інфраструктури, пришвидшуючи видачу програмного забезпечення. За допомогою Docker можна керувати своєю інфраструктурою так само, як і своїми програмами. Методологія Docker для швидкої доставки, тестування та розгортання коду, дозволяє значно скоротити затримку між написанням коду та його запуском у виробництво.

Docker надає можливість упаковувати та запускати застосунок у вільно ізольованому середовищі, яке називається контейнером. Ізоляція та безпека дозволяють запускати багато контейнерів одночасно на одному хості. Контейнери легкі, оскільки їм не потрібно додаткове навантаження гіпервізора, але вони працюють безпосередньо в ядрі хост-машини. Це означає, що можливим є запуск більшої кількості контейнерів на певній апаратній комбінації, ніж можливо при використанні віртуальних машин.

Контейнер – це стандартна одиниця програмного забезпечення, яке пакує код та всі його залежності, завдяки чому програма працює швидко та надійно від одного обчислювального середовища до іншого. Шаблон контейнера Docker – це легкий, автономний, виконуваний пакет програмного забезпечення, що включає все необхідне для запуску програми: код, час роботи, системні інструменти, системні бібліотеки та налаштування. Шаблони контейнерів стають контейнерами під час виконання, а у випадку контейнерів Docker – шаблони стають контейнерами, коли вони працюють на Docker Engine. Контейнерне програмне забезпечення, доступне як для Linux, так і для Windows, завжди працюватиме однаково, незалежно від інфраструктури. Контейнери ізолюють про-

грамне забезпечення від навколишнього середовища та забезпечують його рівномірну роботу, незважаючи на різницю, наприклад, між розробкою та інсталяцією.

Docker використовує архітектуру клієнт-сервер. Клієнт Docker розмовляє з Daemon Docker, який займається будівництвом, запуском та розподілом контейнерів Docker. Клієнт Docker і Daemon можуть працювати в одній системі, або можна підключити клієнт Docker до віддаленого Daemon Docker. Клієнт Docker і Daemon взаємодіють за допомогою REST API, через сокети UNIX або мережевий інтерфейс.

Інструменти для управління, масштабування та обслуговування контейнерних програм називаються оркестраторами, найпоширенішими їх прикладами є Kubernetes та Docker Swarm. Розгортання обох цих оркестраторів у середовищі розробки забезпечує Docker Desktop [12].

Додатки, засновані на мікросервісах, часто складаються з кластерів із сотень екземплярів контейнерних служб. Ця група контейнерів повинна бути стійкою до несправностей, доступною та потенційно географічно розподіленою. Зі збільшенням розміру програми зростає складність управління кластером контейнерів, що його визначають. Щоб в ідеалі отримати вигоду від переваг контейнерів, кластер повинен бути стійким до несправностей, доступним, масштабованим та надійним.

Платформи оркестрації контейнерів можна загалом визначити як систему, що забезпечує структуру на рівні представлення для інтеграції та управління контейнерами в масштабі. Ці платформи спрощують управління контейнерами та забезпечують основу не лише для визначення початкового розгортання контейнера, але й для управління кількома контейнерами як єдиного цілого – для цілей доступності, масштабування та роботи в мережі.

Ключовими можливостями платформ оркестрації контейнерів є:

- управління кластером та планування;
- забезпечення високої доступність та стійкість до несправностей;

- забезпечення безпеки;
- спрощення роботи в мережі;
- можливість виявлення послуг;
- можливість постійного розробки;
- забезпечення моніторингу та управління [13].

2 ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1 Моделювання структури web-додатку

Автоматизована БІС «BOOquarium», що пропонується в даній роботі, має на меті вирішення задач читачів та бібліотекарів, що користуються спільним фізичним сховищем книг (домашньою бібліотекою). На потребу читача у видачі книги бібліотекар має мати змогу зареєструвати факт надання книги у користування, зберегти дані до абонементу читача та до історії книги. З метою візуалізації моделі автоматизованої БІС та зручності проектування бази даних було використано методологію функціонального моделювання IDEF0. У відповідній графічній нотації виконано IDEF0-діаграму та її декомпозицію (рис. 2.1 та 2.2)

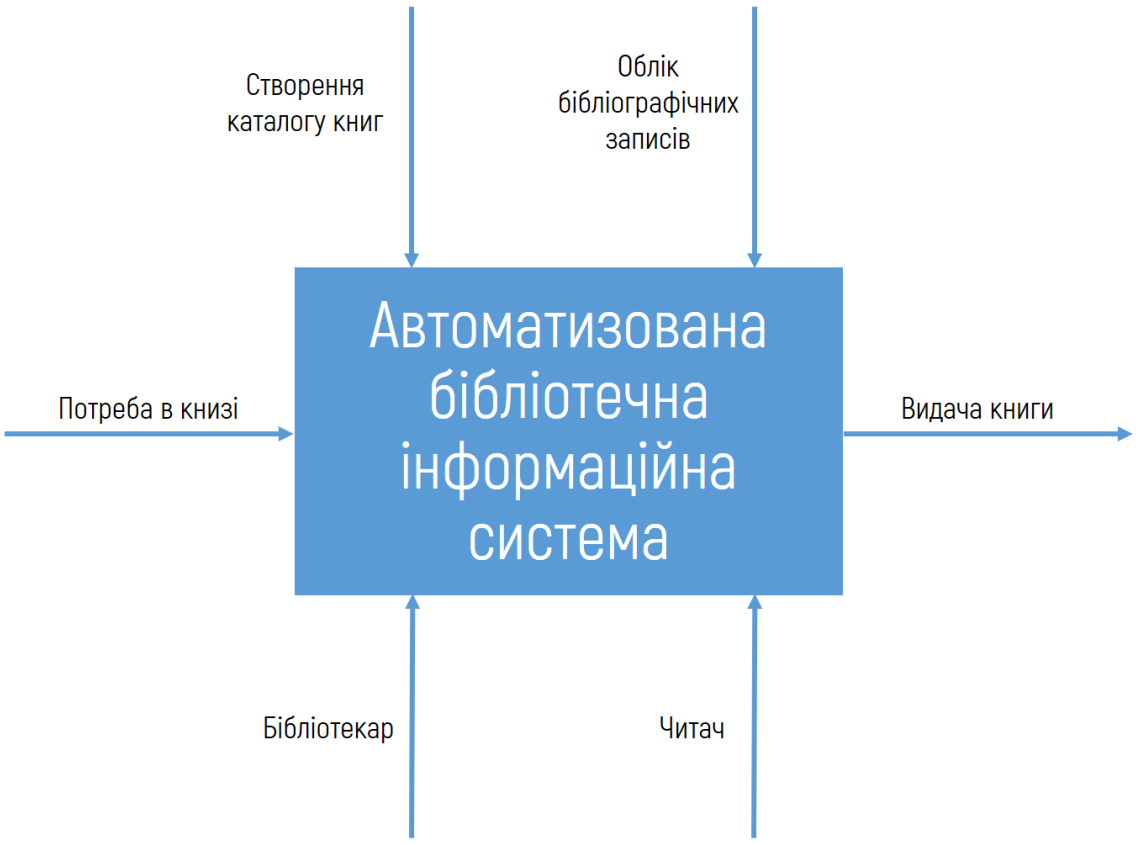


Рисунок 2.1 — Діаграма IDEF0 для автоматизованої БІС «BOOquarium»

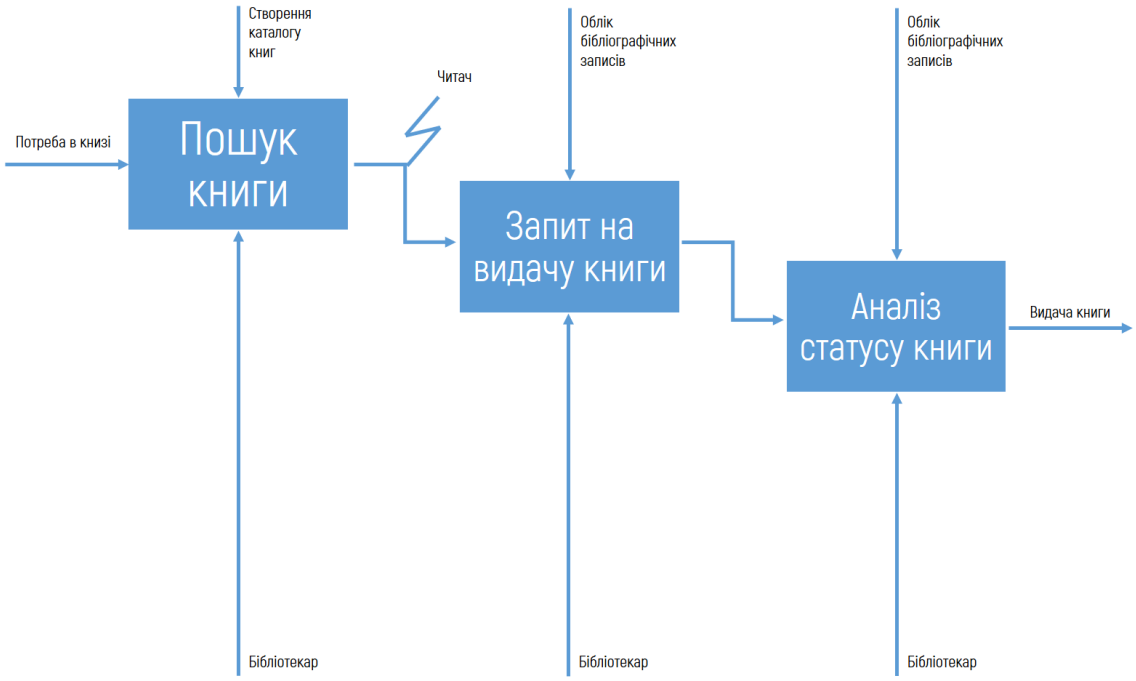


Рисунок 2.2 — Декомпозиція IDEF0-діаграми для автоматизованої БІС «BOOquarium»

Одним з основних етапів моделювання автоматизованої БІС є організація структури головної сторінки (рис. 2.3). Згідно з ідеєю проекта, головна сторінка повинна містити поле пошуку з виведенням списку книг в бібліотеці та кнопки переходу на читачів та форму додавання книг.

Віджет кожної окремої книги в бібліотеці повинен містити наступні поля:

- назва;
- автор.

На кожному віджеті повинні бути присутні три кнопки:

- перегляд книги;
- редагування книги;
- видалення книги.

При натисканні на кнопку перегляду книги, користувач скеровується на сторінку книги, де зазначені (рисунок 2.4):

- назва;

- автор;
- видавництво;
- рік видавництва;
- коментар.

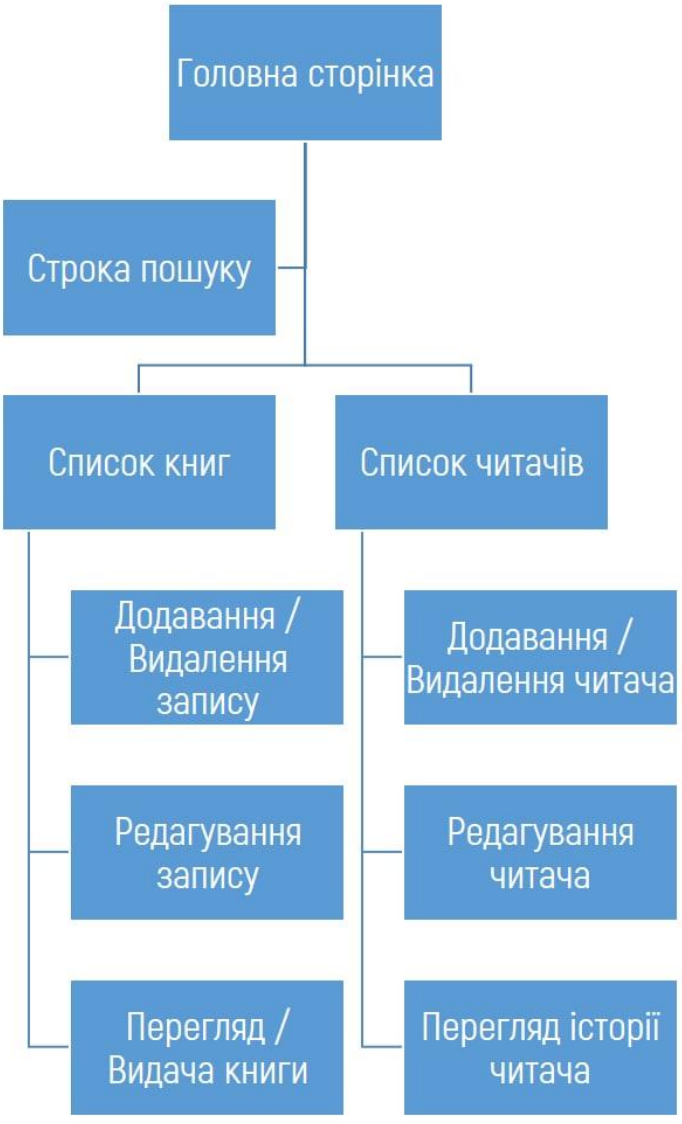


Рисунок 2.3 – Структура автоматизованої БІС

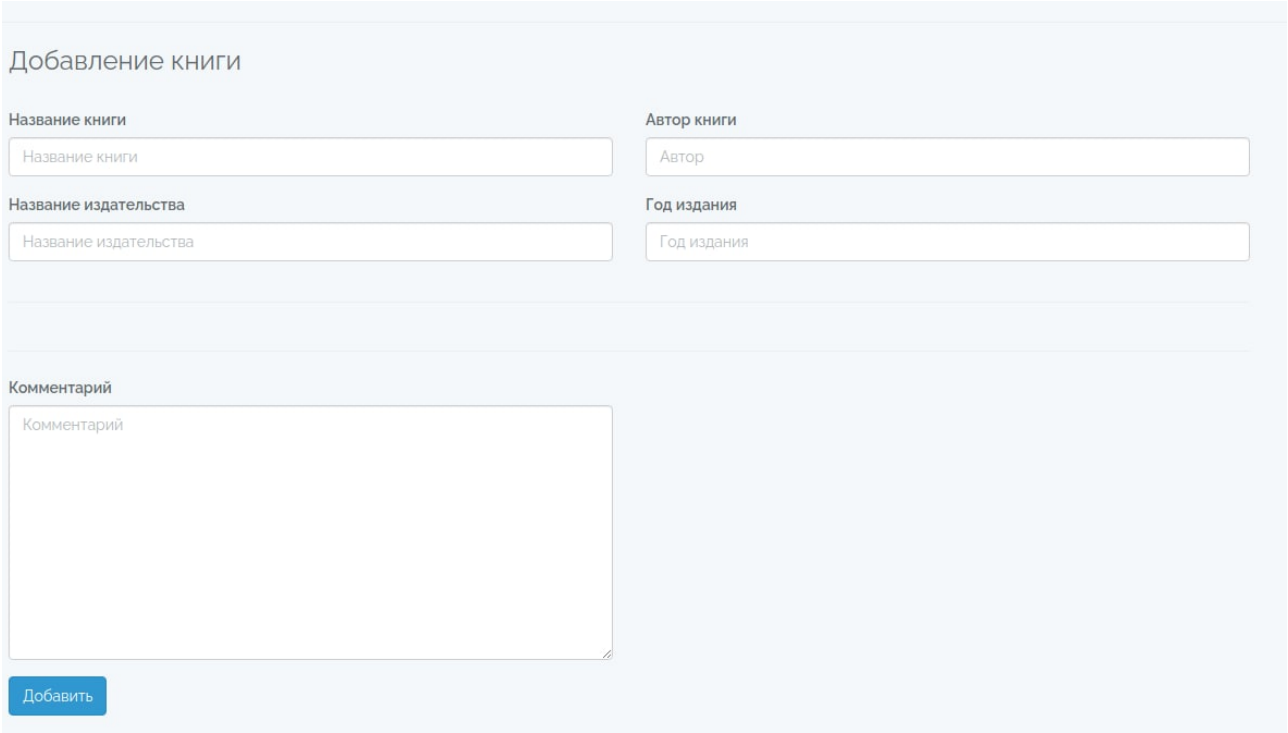


Рисунок 2.4 – Проект сторінки додавання книги

При натисканні на кнопку «Додати книгу», бібліотекарю відкривається форма додавання книги з вищевказаними полями. Ті самі поля, попередньо заповнені попередньо збереженими даними при натисканні на кнопку редагування книги.

При натисканні на кнопку «Читачі» користувач переходить до списку читачів зареєстрованих в системі. Віджет кожного окремого читача в бібліотеці містить ім’я, номер телефону читача, а також кнопки для перегляду/редагування/видалення читача.

При додаванні/редагуванні читача доступні поля для введення ім’я та телефонного номеру.

Для відстежування видачі книг відбуватиметься внесення запису до бази даних щодо того, якому користувачеві було віддано конкретний екземпляр.

2.2 Проектування бази даних

Після того як було розроблено дизайн, була розпочата робота зі створення бази даних. Базу даних було спроектовано з використанням MySQL Workbench.

В рамках бази даних автоматизованої БІС було побудовано 6 таблиць:

- books, що містить поля: унікальний ідентифікатор запису з автоінкрементом, назва книги, ім’я автора, назва видавництва, рік видання, статус наявності, коментар та сигнатури створення та редагування;
- books_readers, що містить поля: унікальний ідентифікатор запису з автоінкрементом, ім’я та прізвище читача, номер телефону, адреса, рейтинг надійності читача, коментар та сигнатури створення та редагування;
- books_locations, що містить поля: унікальний ідентифікатор запису з автоінкрементом, ідентифікатор книги, ідентифікатор читача, коментар та сигнатури створення та редагування;

Таблиця 2.1 Список книг

books
id
name
author
phouse
pyear
on_place
comment
created_at
updated_at

Таблиця 2.2 Список читачів

books_readers
id
name
surname
phone
address
rating
comment
created_at
updated_at

- users, що містить поля: унікальний ідентифікатор запису з автоінкрементом, ім’я, електронна пошта, пароль, хеш для функціоналу запам’ятовування пароля, сигнатури створення та редагування;
- службові таблиці password_resets та migrations.

Таблиця 2.3 Список взятих книг

books_locations
id
book_id
reader_id
comment
created_at
updated_at

Таблиця 2.4 Список бібліотекарів

users
id
name
email
password
remember_token
created_at
updated_at

Таблиця 2.5 Технічна таблиця для скидання паролів

Password_resets
email
token
created_at

Таблиця 2.6 Таблиця міграції

migrations
id
migration
batch

2.3 Проектування UI

Для реалізації візуальної частини планується використання фреймворку Bootstrap. Це інтуїтивно зрозумілий та потужний мобільний фронт-фреймворк

для швидшої та легшої веб-розробки. Він використовує HTML, CSS та Javascript.

Фреймворк Bootstrap має наступні особливості

- складається із стилів First Mobile у всій бібліотеці, а не в окремих файлах;
- підтримується всіма популярними браузерами;
- вимагає лише знань HTML та CSS, а на офіційному сайті Bootstrap розміщена документація;
- адаптивний CSS Bootstrap адаптується до робочих столів, планшетів та мобільних телефонів;
- забезпечує чітке та єдине рішення для побудови інтерфейсу для розробників;
- містить функціональні вбудовані компоненти, які легко налаштувати;
- забезпечує веб-налаштування;
- має відкритий код.

Bootstrap включає наступні пакети:

- Scaffolding: забезпечується базова структура із системою сітки, стилями посилань та фоном;
- CSS: поставляється функція глобальних налаштувань CSS, основними елементами HTML, стилізованими та вдосконаленими розширюваними класами, та вдосконаленою системою сітки;
- Components: Bootstrap містить понад десяток компонентів багаторазового використання, створених для забезпечення іконографії, випадаючих меню, навігації, попереджень, спливаючих вікон та багато іншого;
- JavaScript Plugins: Bootstrap містить понад десяток користувацьких плагінів jQueryз можливістю увімкнення їх усіх або лише окремих;
- Customize: компоненти Bootstrap можна налаштовувати, що потребує менше змінних та плагінів jQuery, щоб отримати власну версію [14].

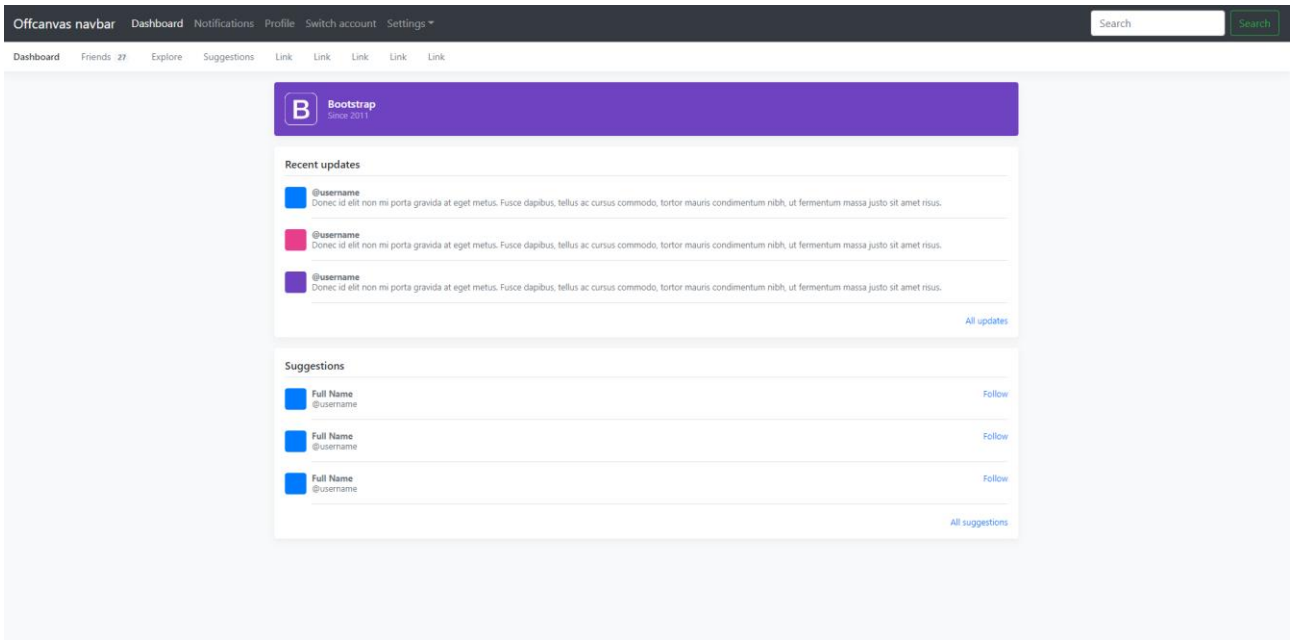


Рисунок 2.5 —Типовий вигляд сторінки, виконаної з використанням BootStrap.

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Конфігурація серверного оточення

Для реалізації БІС в рамках домашньої бібліотеки було прийнято рішення зібрати виділений сервер. В основу білда сервера було покладено енергофективне рішення, що складається з інтегрованої материнської плати ASUS J1800I-C, що вирізняється компактністю та зниженим рівнем споживання електричної енергії; оперативна пам’ять Sodimm DDR3 1,35v об’ємом 4 Гб. В якості носія даних сервер обладнано твердотільним накопичувачем Kingston об’ємом 128 Гб.

Операційною системою на сервері для автоматизованої БІС обрано ubuntu server 20.0.4 без графічної оболонки. Відмова від графічного UI забезпечує швидкодію серверного оточення завдяки відсутності додаткового навантаження на ресурси системи у вигляді рендерингу графічного інтерфейсу.

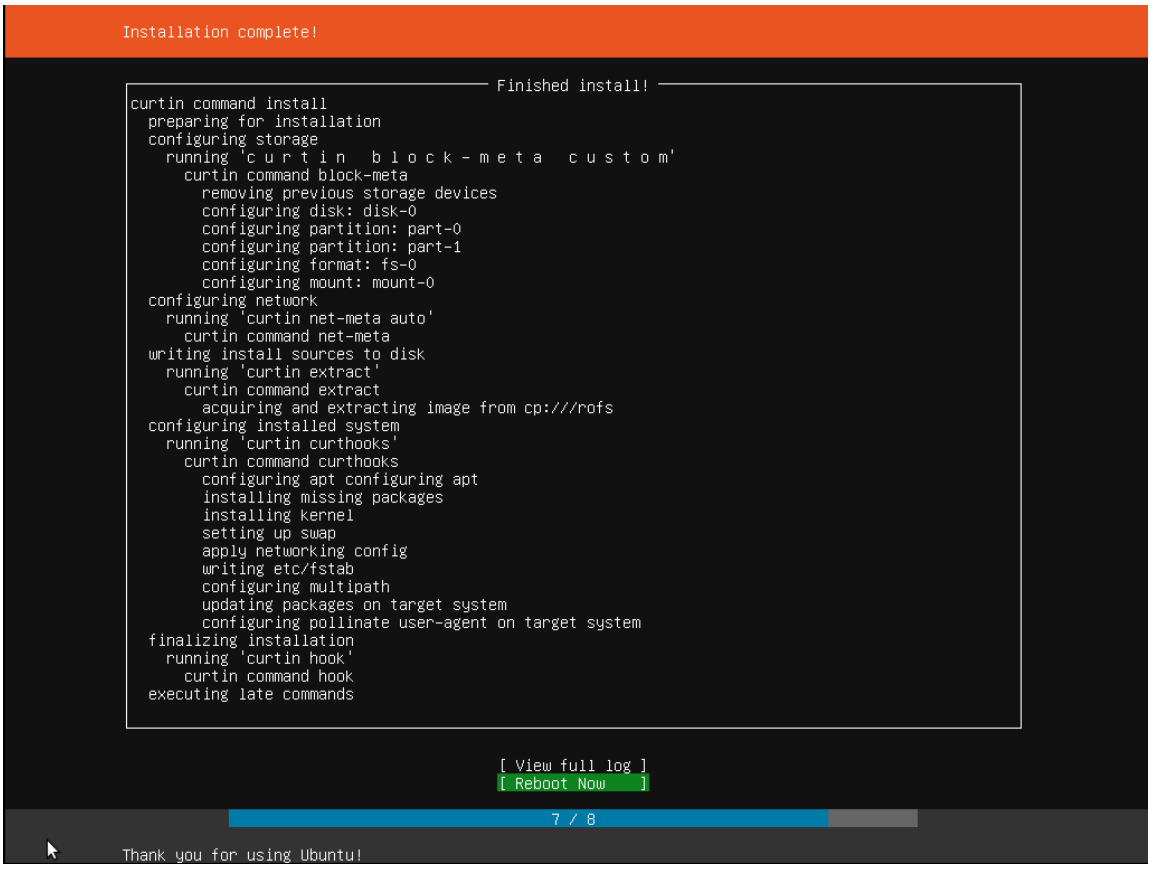


Рисунок 3.1 — Типовий екран успішної інсталяції Ubuntu Server

У якості веб-серверу обрано nginx 1.15, мова реалізації - php 7.3, база даних — MySQL 8.

З метою ефективної оркестрації контейнеризованих процесів було встановлено консольну версію програмного забезпечення Docker та Docker Compose. Так як Ubuntu Server в комплекті поставки містить open-ssh сервер, це дає можливість під'єднуватися до сервера по локальній мережі. Це означає, що налаштування серверу можливо здійснювати віддалено з робочої станції розробника, яка для зручності може працювати під керуванням будь-якої ОС, зокрема, Windows. У цьому випадку підключення до сервера здійснюється за допомогою клієнта putty. Завдяки такій особливості, серверний білд не потребує організації периферійної інфраструктури (монітора, клавіатури).

3.2 Налаштування засобів віртуалізації

Технологія віртуалізації Docker, що використовується у роботі, сумісна з UNIX-системами (у т.ч. MacOS) та Windows 10 (єдина з сімейства Windows). Цей факт пояснюється наявністю у Windows 10 технології апаратної віртуалізації Microsoft Hyper-V, що дозволяє створювати віртуальні машини на системах x86-64 під керуванням Windows.

Верхньорівнева інфраструктура спроектованої БІС конкретизується у файлі-сценарії docker-compose.yml. Всередині цього файлу описано процес контейнеризації фреймоврків та службових програм, що забезпечують функціонування окремих структурних елементів системи. У файлі docker-compose.yml містяться:

- шляхи до сервісних Dockerfile для контейнерів, що містять php, nginx, базу даних, phpmyadmin;
- мапінг директорій для відповідності зовнішніх директорій робочої системи внутрішнім директорії контейнерів;

- зв’язок між контейнерами (наприклад, php-контейнер повинен мати зв’язок з контейнером бази даних);
- описання ряду змінних оточення, наприклад паролі до бази даних, nginx-порти і т.д.

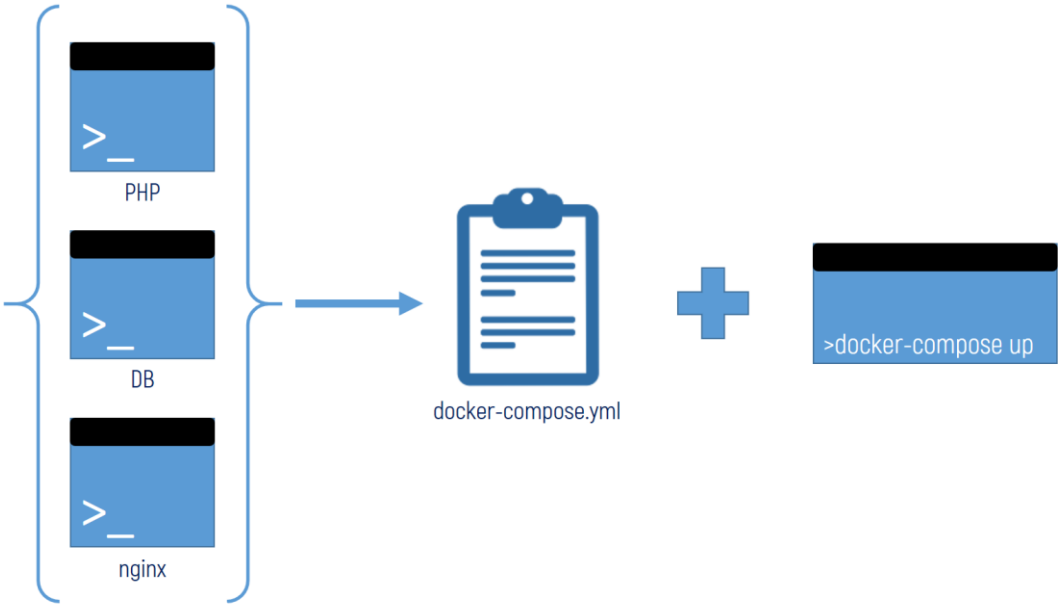


Рисунок 3.2 – Реляційна схема контейнеризації за допомогою Docker-compose

Кожен створений контейнер характеризується окремою областю задач.

В першу чергу контейнеризовано образ веб-сервера nginx 1.15. Технологія nginx може бути використана також для балансування навантаження, в якості proxy-сервера та містить nginx-кеш, що дозволяє істотно знизити кількість запитів на back-end. Цей ефект досягається шляхом зберігання HTTP-відповіді на визначений час, а при повторному зверненні до ресурсу, видачі його кеша без перенаправлення запиту на back-end. Також, в файлі-сценарії docker-compose.yml було проведено налаштування можливості роботи з конфігураціями без необхідності входу всередину контейнера. В блоці параметрів контейнера Nginx описані зв’язки з іншими контейнерами та базові системні налаштування. Одночасно з цим, блок параметрів цього контейнера містить шлях до файлу з переліком команд, що необхідно виконати одразу після першого ство-

рення контейнеру. Кожен наступний контейнер має аналогічний файл налаштувань «Dockerfile».

```
Building nginx
Step 1/3 : FROM nginx:1.15
 ---> 53f3fd8007f7
Step 2/3 : RUN apt update && apt install mc -y && apt install vim -y
 ---> Using cache
 ---> 26fcd34321c9
Step 3/3 : RUN echo "" > /etc/nginx/conf.d/default.conf
 ---> Using cache
 ---> f1eec632cc96
```

Рисунок 3.3 – Білд контейнера nginx

Контейнер «PHP» збудований на базі образу з предвстановленою скриптовою мовою програмування PHP 7.3. Цей контейнер містить додаткові модулі та бібліотеки, необхідні для коректної роботи фреймворка Laravel.

У контейнері «DB» розміщено базу даних на основі MySQL 8. Файл сценаріїв docker-compose.yml містить необхідні константи для роботи з базою даних. Також окремо створено службовий контейнер з веб-застосунком phpMyAdmin.

```
Building db
Step 1/4 : FROM mysql:8
 ---> dd7265748b5d
Step 2/4 : RUN apt update && apt install mc -y && apt install vim -y
 ---> Using cache
 ---> cfb64cb7d7d5
Step 3/4 : RUN mkdir -p /tmp/db_backup && chmod -R 777 /tmp/db_backup
 ---> Using cache
 ---> cea4a01353a8
Step 4/4 : VOLUME /tmp/db_backup
 ---> Using cache
 ---> 4a5bdf9031dd

Successfully built 4a5bdf9031dd
Successfully tagged dbs_db:latest
```

Рисунок 3.4 – Білд контейнера DB

У відповідних Dockerfile, до яких вказано шлях всередині docker-compose.yml, вказано ряд команд для встановлення додаткових бібліотек, зокрема xdebug, mysqli pdo_mysql, php_ext. В кожному контейнері при першому запуску встановлюються допоміжні утіліти для спрощення процесу редагування та навігації по файловій системі: текстовий редактор vim та midnight commander.

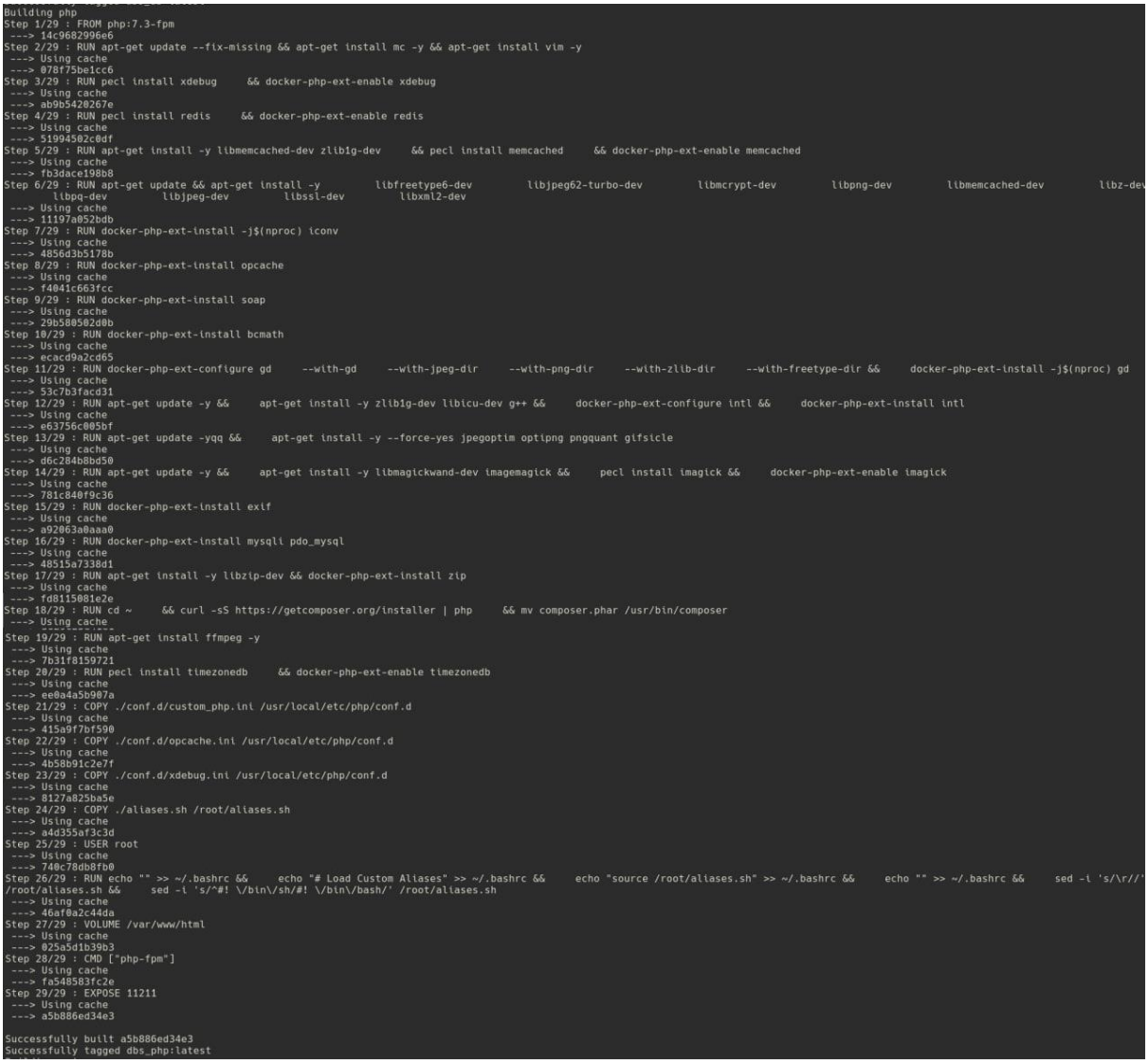


Рисунок 3.5 – Білд контейнера PHP

Гнучке налаштування системи без доступу всередину контейнера забезпечується спеціально створеною та промапленою папкою Config, до якої організовано доступ з робочої станції розробника. Перший запуск контейнеризованої системи відбувається за допомогою команди docker-compose build. Ця команда створює контейнери, завантажуючи необхідне програмне забезпечення, бібліотеки та утиліти з Інтернет-репозитаріїв, шляхи до яких прописано у відповідних Dockerfile.

Було прийнято рішення не використовувати систему контроля версій безпосередньо на сервері, а застосувати її на робочій станції. Завдяки використан-

ню високошвидкісного мережевого обладнання з гігабітним каналом, твердотільних накопичувачів з швидким циклом перезапису на сервері та робочій станції, процес запису та передачі даних при розробці БІС здійснюється без затримок на проходження інфраструктури. Це забезпечує зручний та ефективний процес розробки з віддаленої робочої станції.

Після разового виконання команди «docker-compose build», що будує контейнеризовану структуру системи, кожен наступний запуск системи здійснюється за допомогою команди «docker-compose up», що підіймає раніше збудовану структуру системи.

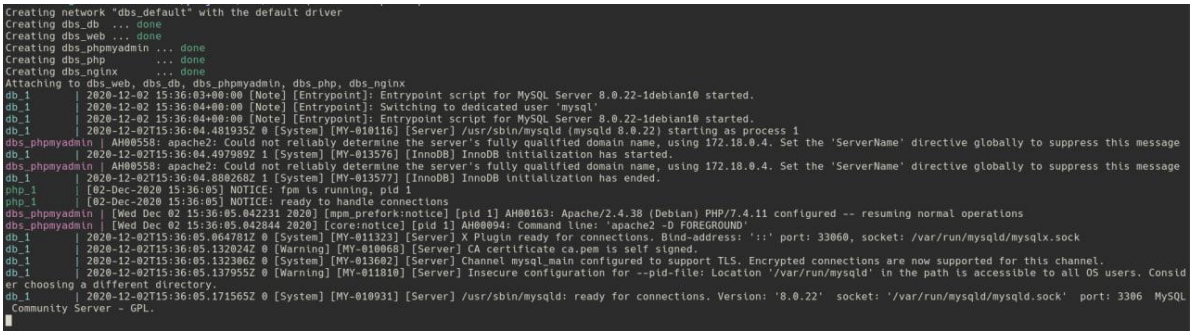


Рисунок 3.6 – Результат виконання команди docker-compose up

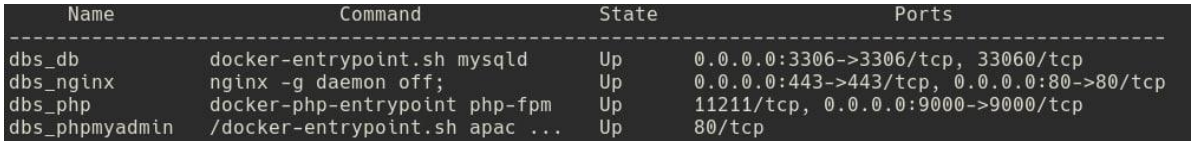


Рисунок 3.7 – Збудовані та підняті контейнери

На робочій станції здійснюється підключення мережевої директорії, що містить файли та директорії проекту. Проект розміщується на репозиторії BitBucket з використанням системи контроля версій Git.

3.3 Процес розробки

У сформованому оточенні процес розробки будується на фрейврорку Laravel. Було створено базові моделі для сутностей books, books_readers, books_locations, у яких описано поля із відповідних таблиць. Одночасно з цим,

для кожної сутності було створено окремий контролер з чотирма базовими методами, так званий CRUD (Create, Read, Update, Delete). У процесі розробки було також написано додаткові методи в контролерах:

- метод index контролера books, призначений для пошуку книг;
- метод view контролера books, призначений для перегляду бібліографічного запису;
- метод return book, що реалізує функціонал повернення книги читачем.

Окрім моделей та контролерів було описано так звані роути — мапінг URL до визначеного методу визначеного контролера, наприклад, books/view/id. Роути необхідні для того, щоб при введенні прямого URL (або по натисненню кнопки, що прив’язана до цього URl) відбувалося відображення коректного view, а також виконувався правильний метод контролера.

Фреймворк Laravel використвоує шаблонізатор «Blade», що має власні зарезервовані функції. Відповідно, для кожного відображення сторінки було створено окремий blade-шаблон, наприклад books.view.blade, books.add.blade. Всередині цього шаблону було створено HTML-розмітку з використанням класів Bootstrap. Однак, в реальній ситуації Bootstrap не завжди здатний вирішити всі задачі, у цьому випадку доводиться створювати унікальні класи, стилі для яких описані в окремих каскадних таблицях стилів у файлі custom.css.

На етапі проектування та початкової реалізації автоматизована БІС мала обмежений функціонал та не передбачувала ряду сценаріїв, що притаманні реальним бібліотекам. Виникла проблема відсутності можливості обліку фізичного розташування книги та додавання електронної копії. У зв’язку з потребою цього функціоналу, у версії 0.8 було імплементовано ряд нововведень, спрямованих на розширення функцій БІС.

Відповідні поля у БД були відсутні на етапі проектування, і їх довелося додавати у існуючу таблицю. Для реалізації подібного сценарію Laravel передбачає функціонал міграцій, за допомогою яких вдалося успішно створити відповідні поля. Додатковою перевагою можливості міграцій є перспектива

паралельної розробки декількома фахівцями, а також наявність способу реверсувати зміни, внесені до бази даних. З цією метою було створено техгічну таблицю migrations (рис.3.8).

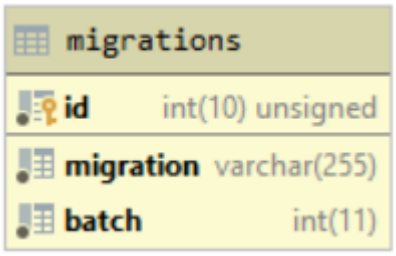


Рисунок 3.8 – Таблиця міграції

Імплементовані нові поля було описано в моделях, а також, зважаючи на внесені зміни, було частково переписано методи контолера books. Було додано новий метод для завантаження електронної копії книги з відповідним роутом. На рисунках 3.9 – 3.13 зображено імплементовані таблиці бази даних з додатковими полями.

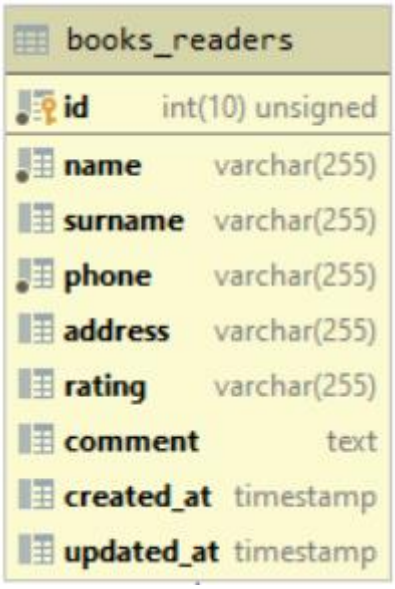


Рисунок 3.9 – Список читачів

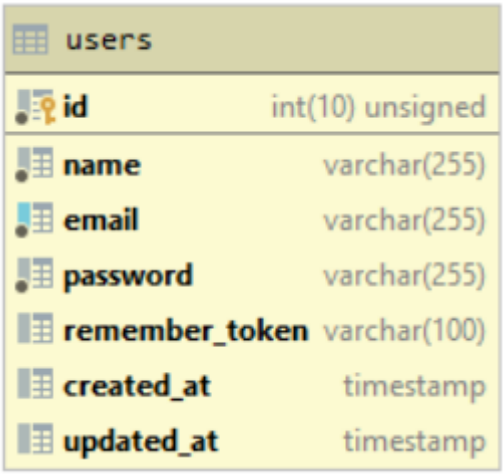


Рисунок 3.10 – Список бібліотекарів

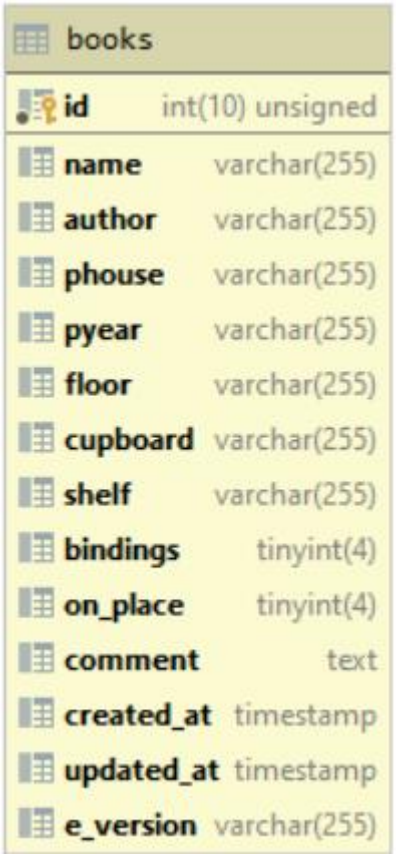


Рисунок 3.11 – Список книг

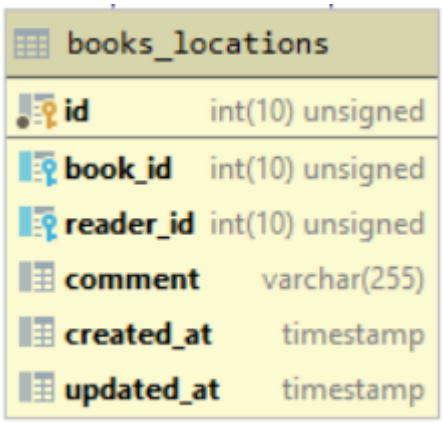


Рисунок 3.12 – Список взятих книг

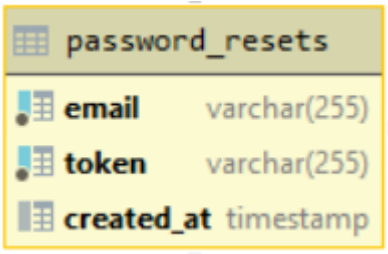


Рисунок 3.13 – Технічна таблиця для скидання паролів

Наповнення бази відбувалося в ручному режимі. В наступній версії 0.9 планується впровадження автоматичної каталогізації книг за допомогою фізичного сканера, що працюватиме по API з веб-застосунком.

3.4 Конфігурація інтерфейсу системи засобами html/css фреймворку BootStrap

У зв’язку з обмеженим часовим та фінансовим бюджетом проекту, розгалуджена розробка дизайн-макету автоматизованої БІС не є доцільною. В якості оформлення візуальної частини БІС було обрано html/css фреймоврк BootStrap. Цей фреймворк містить предналаштовані класи, що присвоюються html-елементам, внаслідок чого досягається задовільний візуальний вигляд сторінки.

Разом з BootStrap було використано Font Awesome – це набір інструментів для шрифтів та піктограм, заснований на CSS і Less, розроблений Дейвом Ганді для використання з Bootstrap, а пізніше був включений у BootstrapCDN. Font Awesome має повністю відкритий код і може використовуватись для комерційних проектів, проектів з відкритим кодом чи будь-яких інших.

Font Awesome має наступні особливості:

- це піктографічна мова веб-дій, що налічує один шрифт та 675 іконок;
- не зважаючи на те, що Font Awesome розроблений для використання з Bootstrap, він сумісний з усіма фреймворками;
- проблеми з сумісністю мінімізуються тим, що Font Awesome не вимагає JavaScript;
- можливим є стилізація кольору піктограм, розмір, тінь та іншого завдяки CSS;
- сумісний з десктопними версіями;
- масштабована векторна графіка дозволяє переглядати піктограми у будь-якому розмірі та на дисплеях з високою роздільною здатністю;
- сумісний з програмами зчитування з екрану [15].

Головна сторінка сайту одночасно виконує функцію списку наявних в бібліотеці книг з можливістю їх пошуку. Також, на сторінці розміщений логотип та три кнопки: «Додати книгу», «Читачі» та «Пошук» (рис.3.14). На кожній картці книги розміщено три іконки, що відповідають за перегляд, редагування та видалення книги.

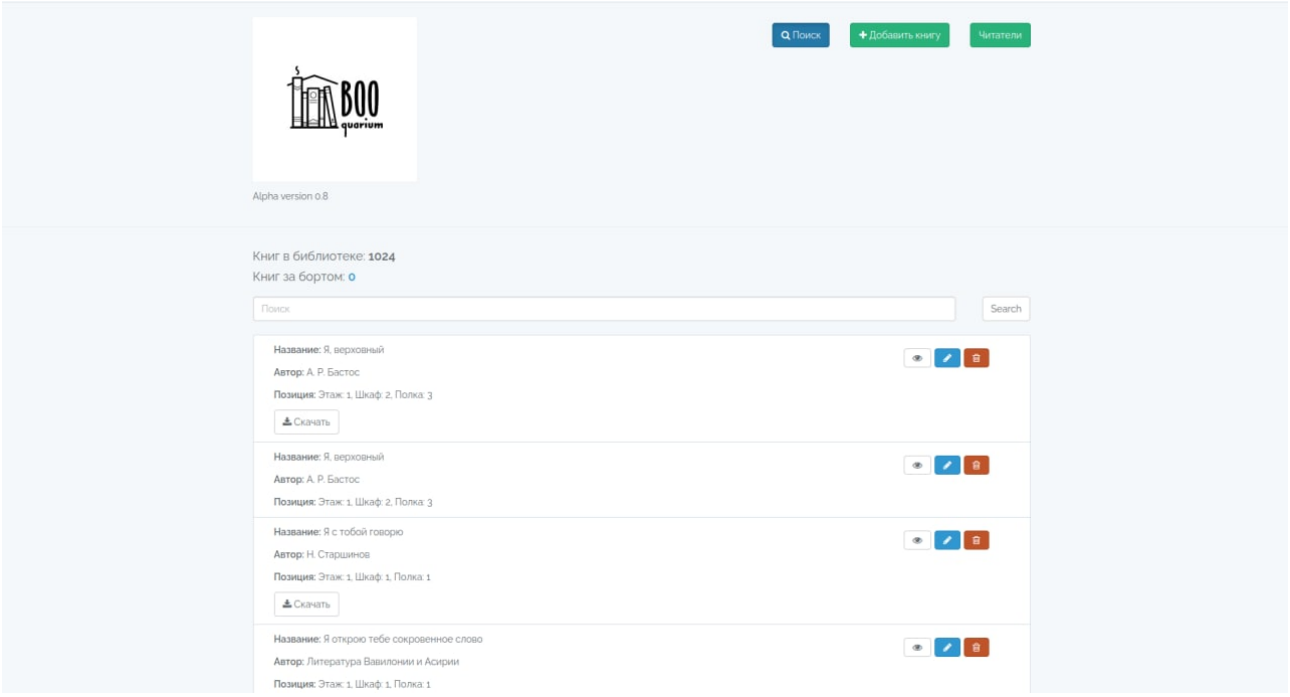


Рисунок 3.14 – Головна сторінка сайту

Пошук книг можливий за такими полями як автор, назва, видавництво, рік видавництва та коментар (рис.3.15).

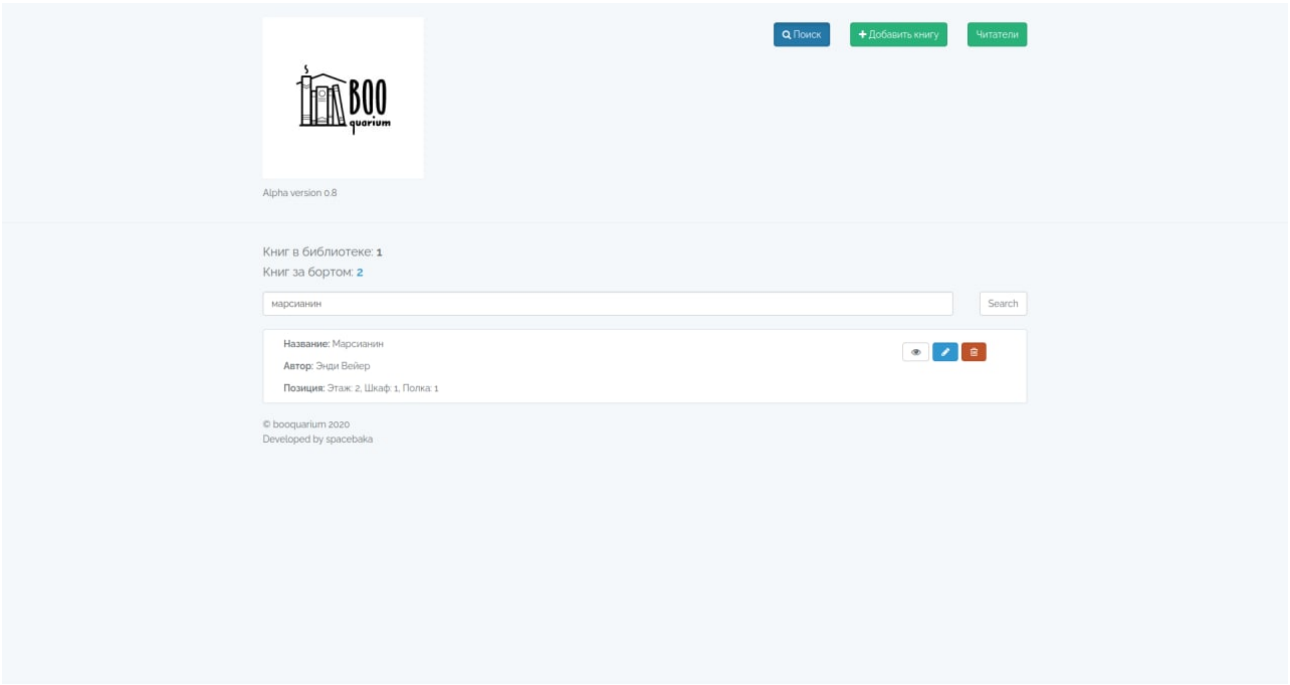


Рисунок 3.15 – Пошук на сайті

При натисканні на кнопку «Додати книгу» користувачу відкривається сторінка додавання книги, де він повинен заповнити поля: назва, автор, видавництво, рік видавництва, перепліт, поверх, шафа, полиця та за потреби залишити коментар (рис.3.16).

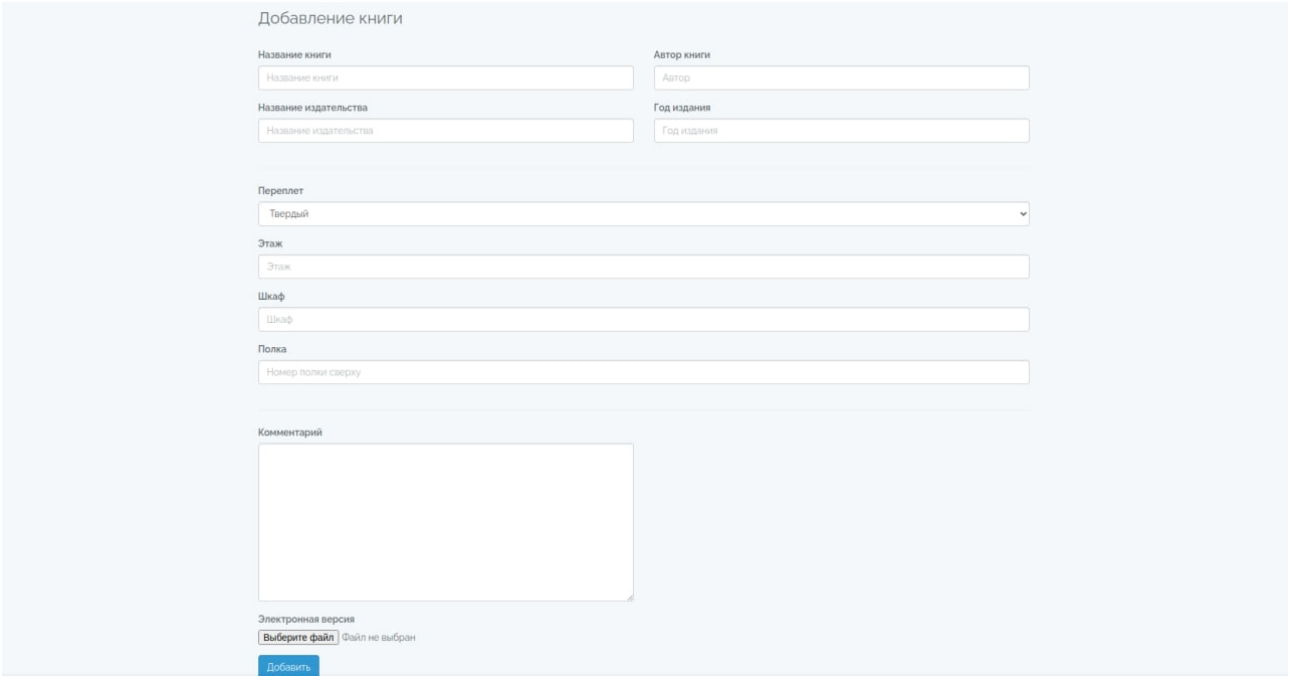


Рисунок 3.16 – Додавання книги

При натисканні іконки з олівцем користувачу відкривається сторінка редагування внеги, що містить ті самі поля, що й сторінка додавання книги, але вони є попередньо заповненими введеними раніше значеннями (рис.3.17).

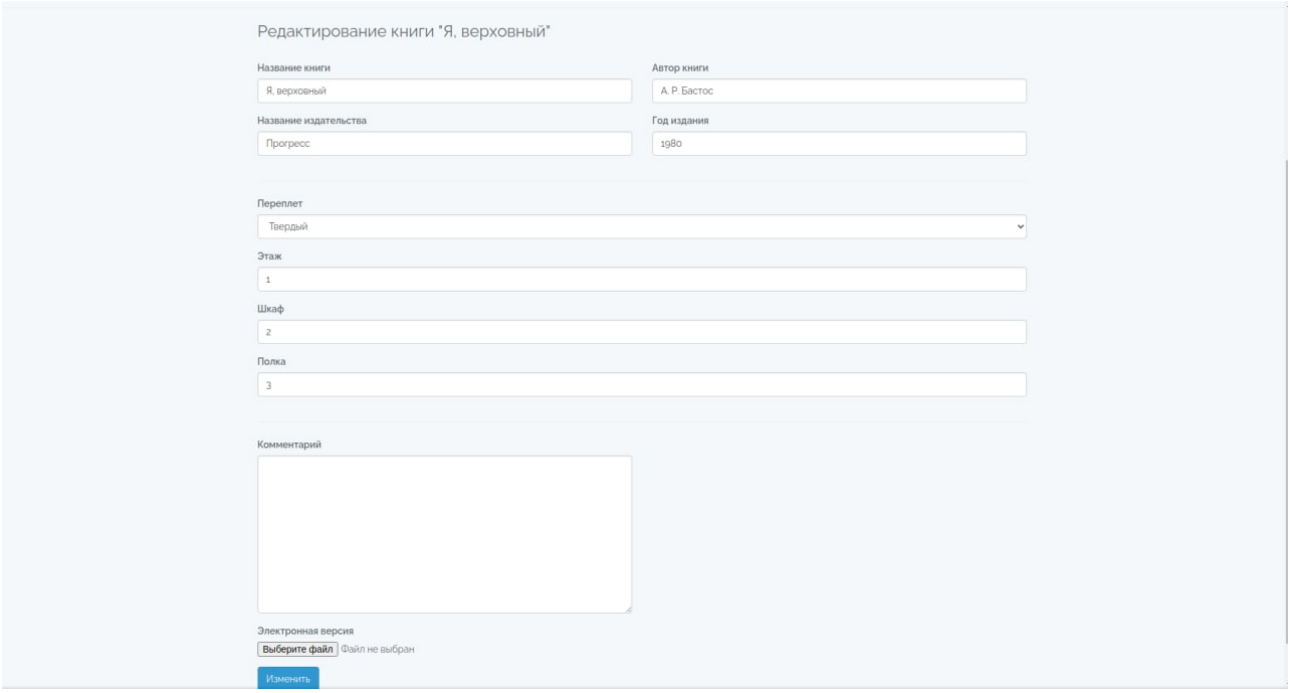


Рисунок 3.17 – Редагування книги

Після того, як користувач знайшов потрібну книгу, він має змогу її переглянути, натиснувши на іконку перегляду. Користувачеві відкривається сторінка книги з основною інформацією про неї та кнопкою «Дати книгу» (рис.3.18), а також, з можливістю завантажити електронний варіант (в разі його наявності) та переглянути історії видач книги (рис.3.19).

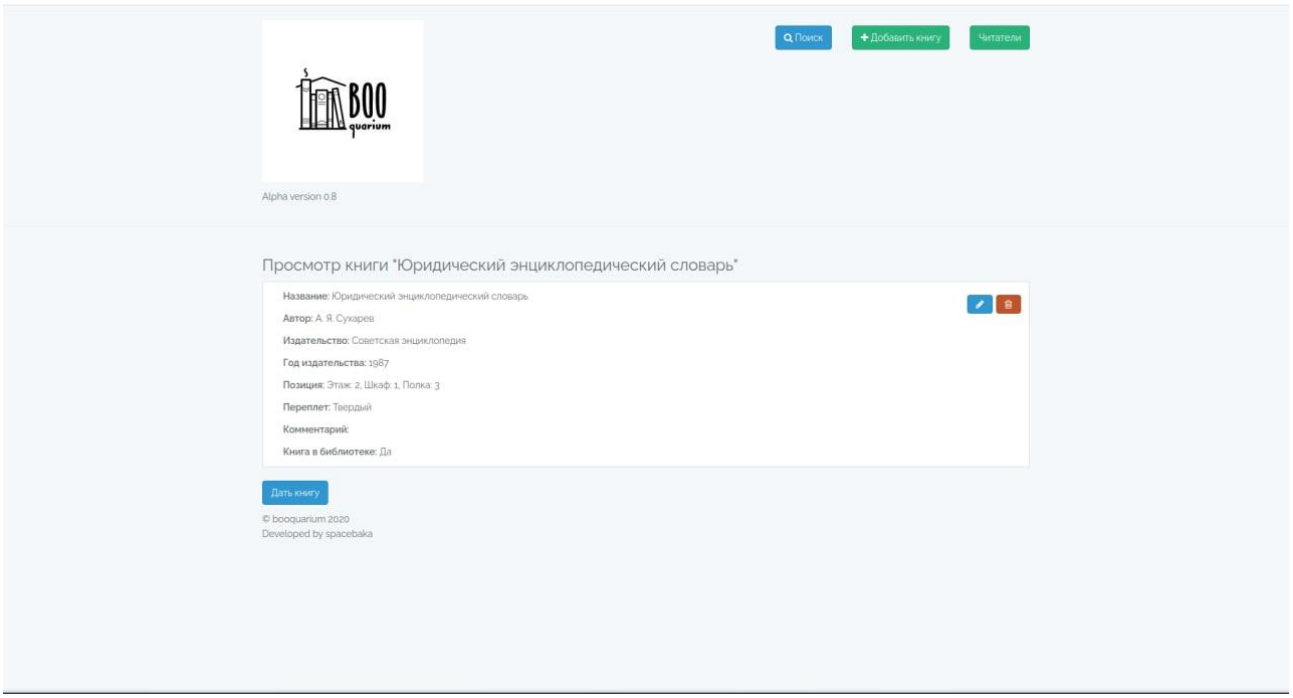


Рисунок 3.18 – Сторінка перегляду книги

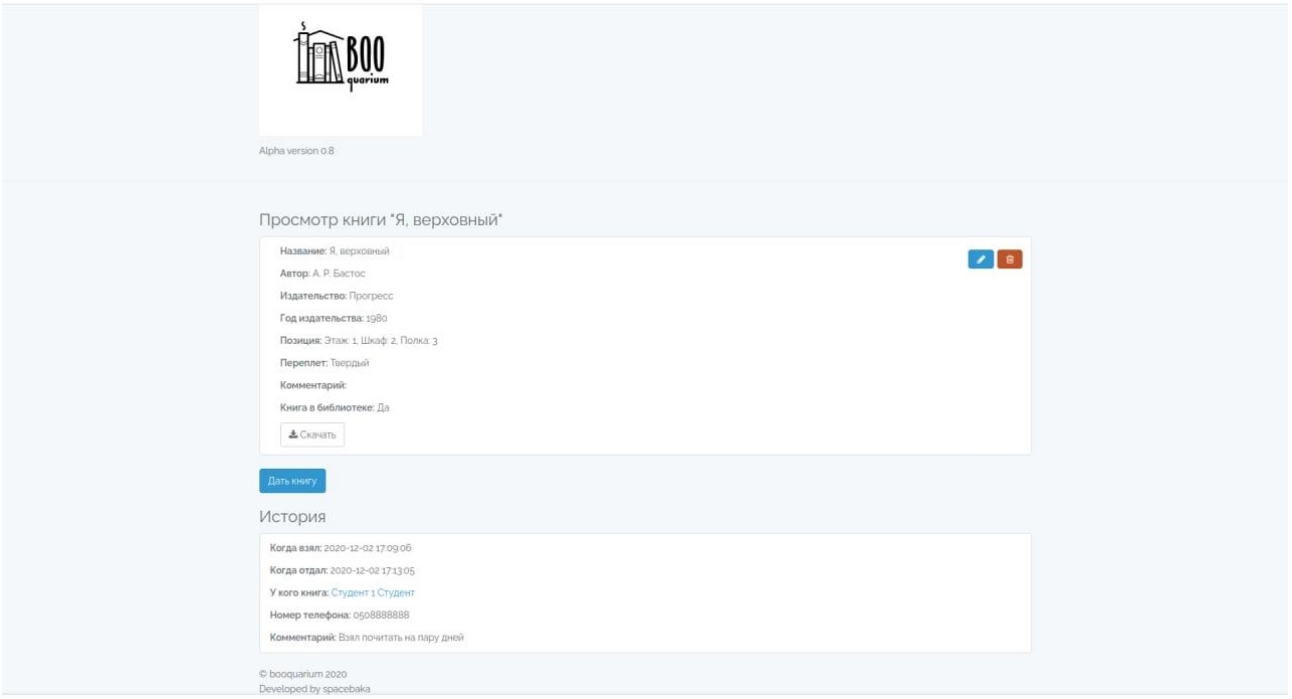


Рисунок 3.19 – Сторінка перегляду книги з електронним варіантом та історією видач

При натисканні на кнопку «Читачі» користувачу відкривається сторінка зі списком внесених адміністратором читачів (рис.3.20). На кожній картці читача розміщено три іконки, що відповідають за перегляд, редагування та видалення читачів.

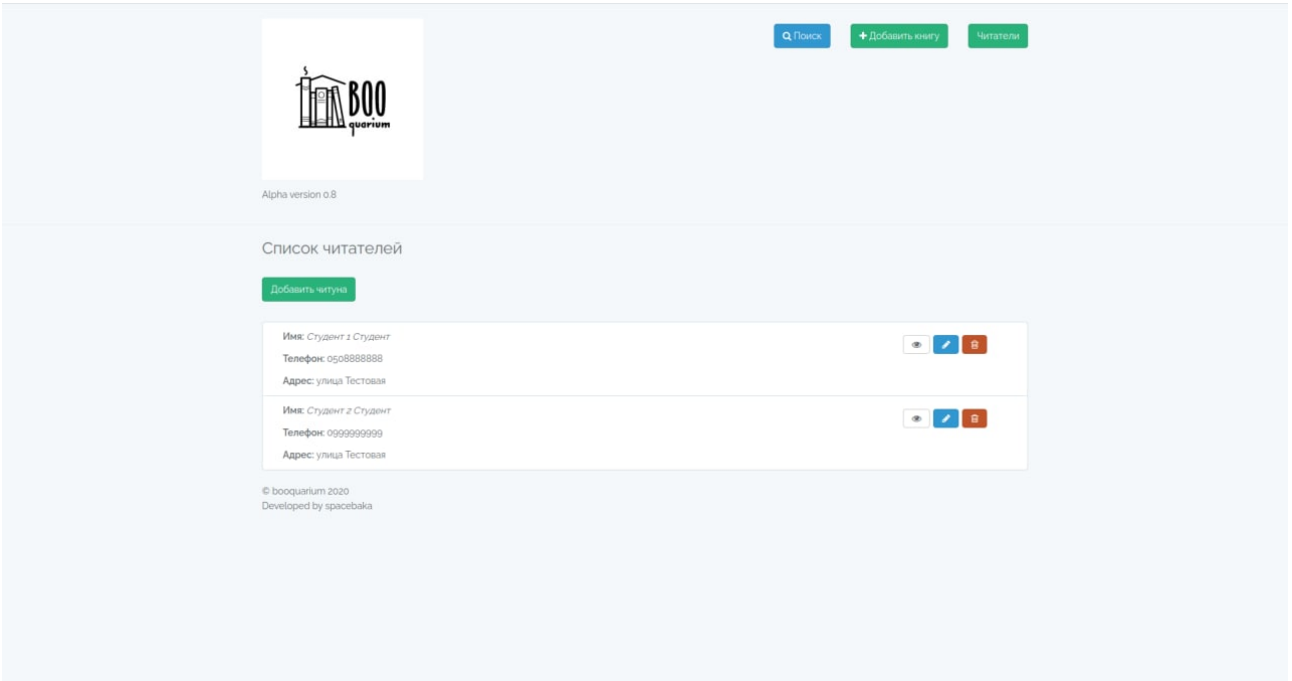


Рисунок 3.20 – Сторінка читачів

Адміністратор має можливість додати нового читача натиснувши кнопку «Додати читача». Після цього відкриється сторінка додавання читача, де необхідно ввести ім’я, прізвище, номер телефону, адресу читача та за необхідності залишити коментар (рис.3.21).

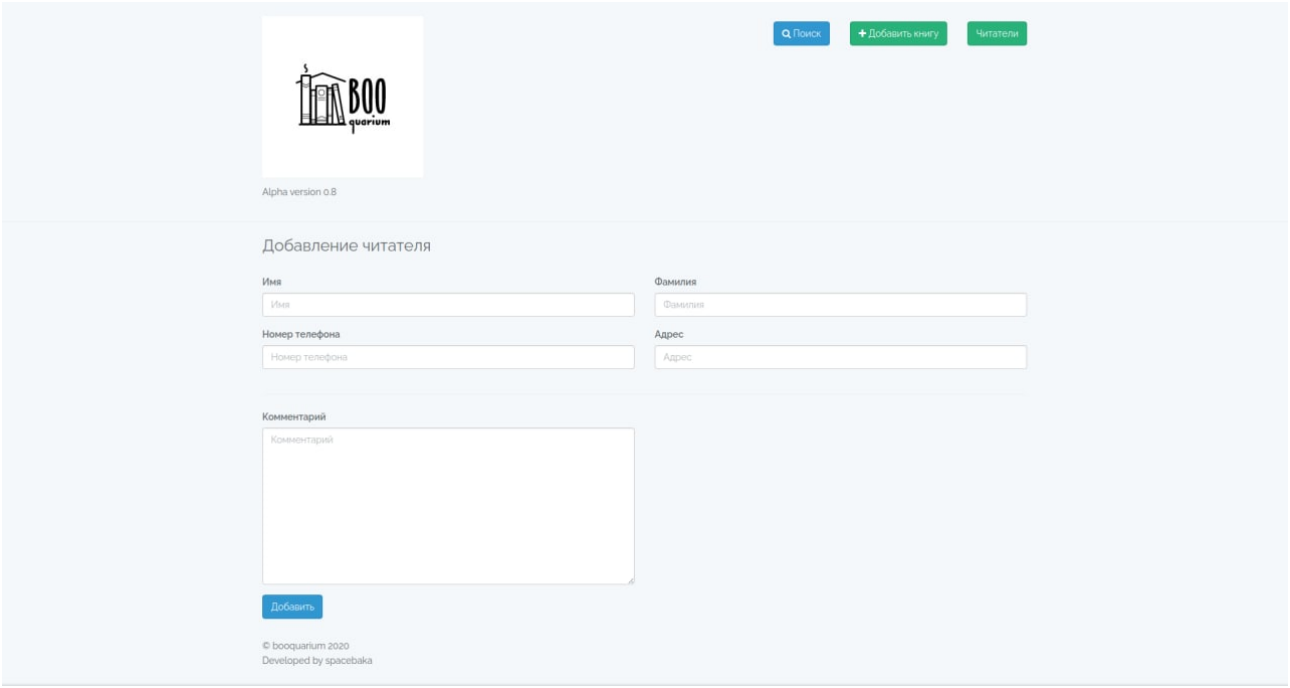


Рисунок 3.21 – Сторінка додавання читача

При натисканні на іконку перегляду відкривається сторінка окремого читача з відомостями про нього, що включають: ім’я, прізвище, номер телефону, адресу, коментар та історію видач книг цьому читачеві (рис.3.22).

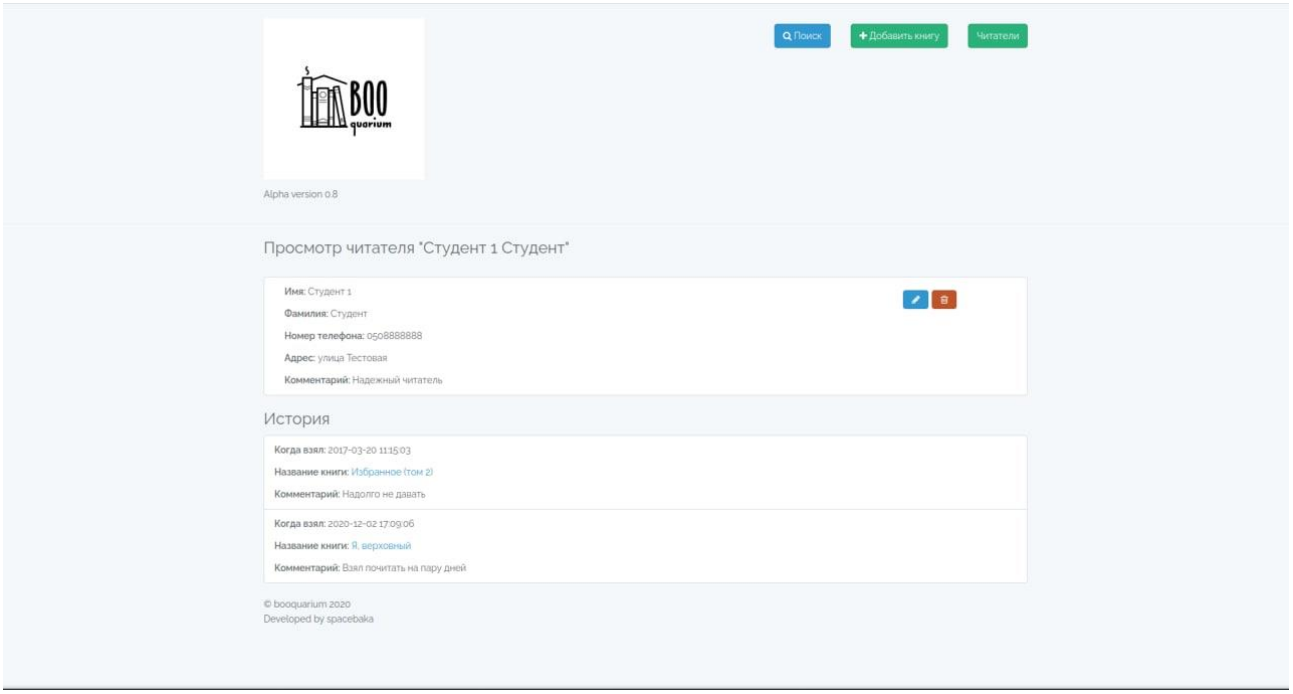


Рисунок 3.22 – Сторінка перегляду читача

При натисканні на іконку редагування користувачу відкривається сторінка редагування читача, що містить ті самі поля, що й додавання читача, але з попередньо заповненими значеннями, які може бути змінено (рис.3.23).

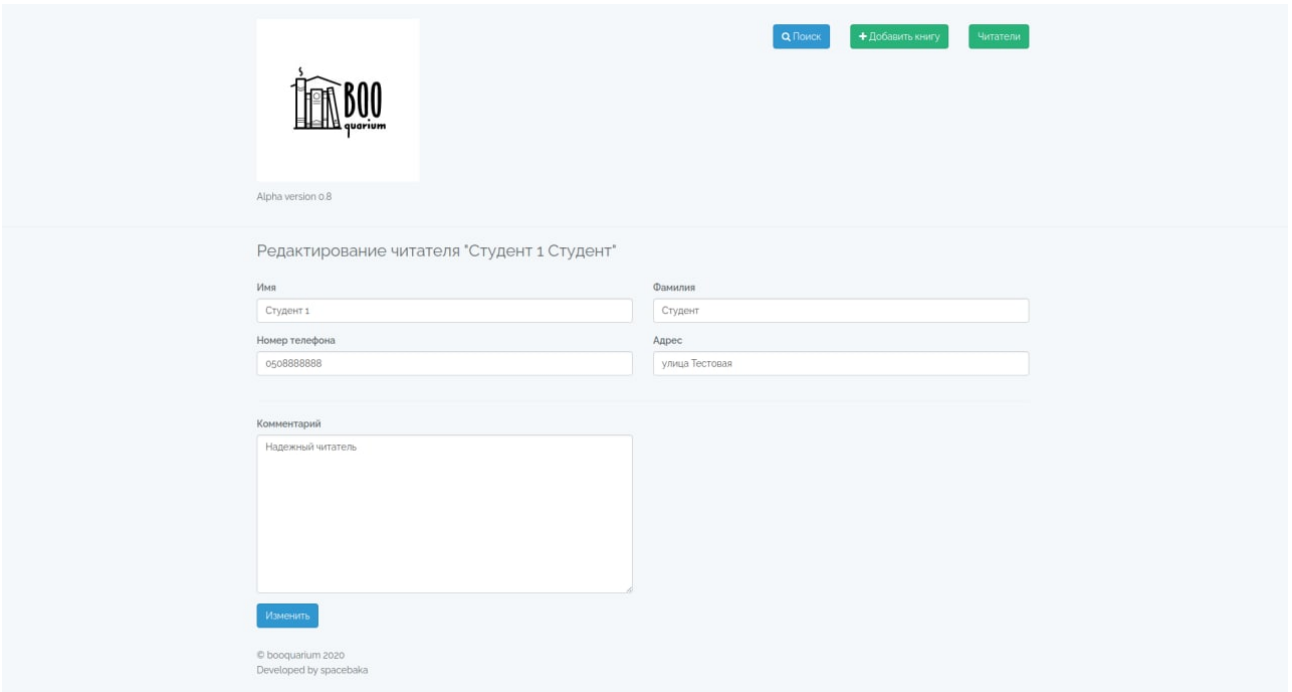


Рисунок 3.23 – Сторінка редагування читача

Після того, як книгу було видано, в загальному списку книг її буде позначено як таку, що відсутня в бібліотеці (рис.3.24).

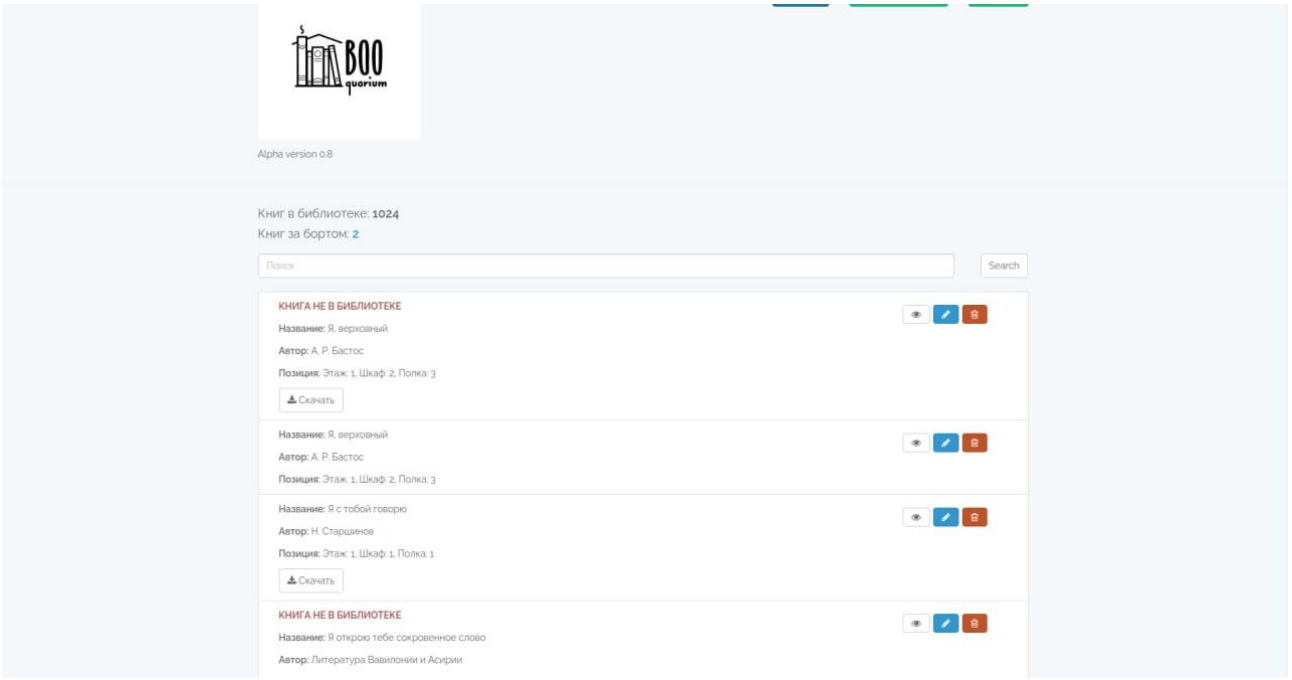


Рисунок 3.24 – Сторінка книги, що видана читачеві

При перегляді таких книг на сторінці з’являється «Історія» з інформацією про те, хто і коли її взяв, а також номер телефону відповідного читача та коментар. Після натискання кнопки «Повернути книгу» запис про її видачу буде скасовано (рис.3.25).

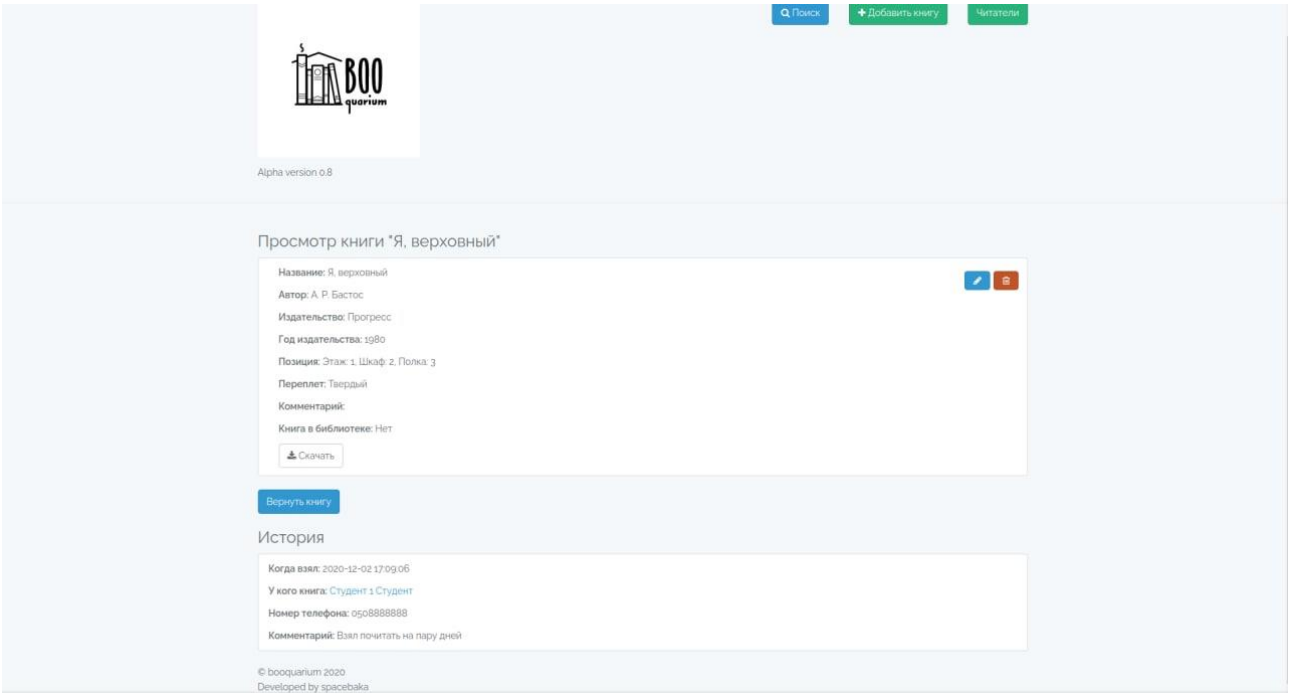


Рисунок 3.25 – Сторінка книги, що видана читачеві

3.5 Мануальне тестування програмного продукту

Зважаючи на наявність доступу до коду тестування проводилося за методом білого ящика. Тестування охоплювало як позитивні кейси (додавання, необхідної інформації, пошук серед наявних книг, видача книги) та і негативні (спроби зберегти книги з незаповненими обов'язвокими полями, пошук відсутніх книг тощо). Зважаючи на те, що проект не є масштабним, а додаток призначений для приватного користування, здійснювалося здебільшого функціональне тестування без необхідності проведення навантажувального тестування.

Для тестування імплементованого продукту було розроблено наступні тест кейси:

- додавання книги;
- пошук/перегляд/редагування/видалення книг в системі;
- додавання/редагування/видалення читачів;
- видача книг читачам.

У наведених нижче таблицях розміщено розроблені тест кейси з результатами їх виконання.

Таблиця 3.1 Тест кейс «додавання книги» та результати його виконання

Step	Description	Expected result	Actual result
1	Відкрити сайт http://dbs.com	Відкрито головну сторінку BOOquarium	Відкрито головну сторінку BOOquarium
2	Натиснути у верхньому правому кутку кнопку «Додати книгу»	Відкрито сторінку додавання книги	Відкрито сторінку додавання книги
3	Вибірково заповнити текстові поля для додавання книги та натиснути кнопку «Додати»	Неможливо зберегти книгу, з’являється валідаційне повідомлення (рис.3.26)	Неможливо зберегти книгу, з’являється валідаційне повідомлення
4	Заповнити всі необхідні текстові поля та натиснути кнопку «Додати»	Книгу успішно додано	Книгу успішно додано

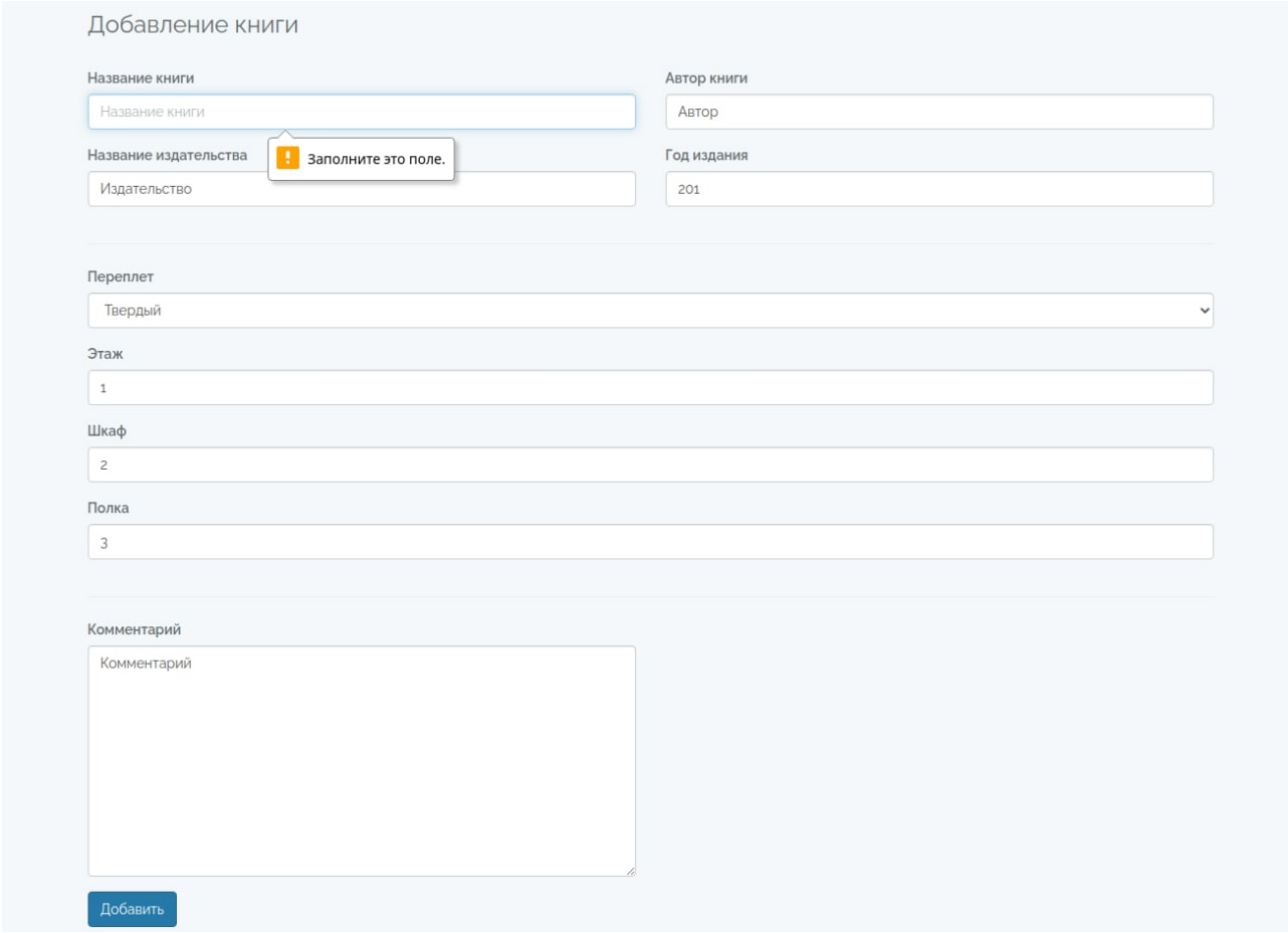


Рисунок 3.26 – Валідаційне повідомлення при необхідності заповнити поле

Таблиця 3.2 Тест кейс «пошук/перегляд/редагування/видалення книг в системі» та результати його виконання

Step	Description	Expected result	Actual result
1	Відкрити сайт http://dbs.com	Відкрито головну сторінку BOOquarium	Відкрито головну сторінку BOOquarium
2	Ввести в рядку пошуку назву/автора/видавництво/рік видавництва книги доданої у попередньому кейсі	Книгу знайдено та відображено в результатах пошуку (рис.3.27)	Книгу знайдено та відображено в результатах пошуку
3	Натиснути іконку перегляду на знайденому елементі	Відкрито сторінку з повною	Відкрито сторінку з повною

		інформацією про книгу	інформацією про книгу
4	Натиснути іконку редагування на книзі	Відкрито сторінку редагування книги з передзаповненими полями відповідно до внесеної раніше інформації про книгу	Відкрито сторінку редагування книги з передзаповненими полями відповідно до внесеної раніше інформації про книгу
5	Видалити текст в полі назви	Неможливо зберегти книгу, з’являється валідаційне повідомлення	Неможливо зберегти книгу, з’являється валідаційне повідомлення
6	Заповнити поле назви новим значенням та натиснути кнопку «Змінити»	Книгу успішно відредаговано	Книгу успішно відредаговано
7	Ввести в рядку пошуку нову назву книги	Книгу знайдено та відображено в результатах пошуку (рис.3.28)	Книгу знайдено та відображено в результатах пошуку
8	Натиснути іконку видалення на знайденому елементі	Книгу успішно видалено	Книгу успішно видалено
9	Ввести в рядку пошуку нову назву книги	Результатів не знайдено (рис.3.29)	Результатів не знайдено

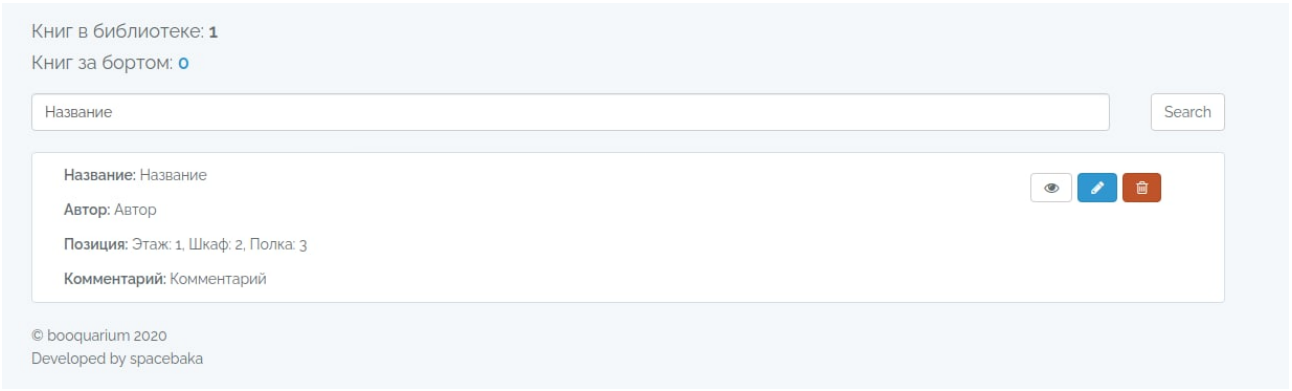


Рисунок 3.27 – Відображення результатів пошуку

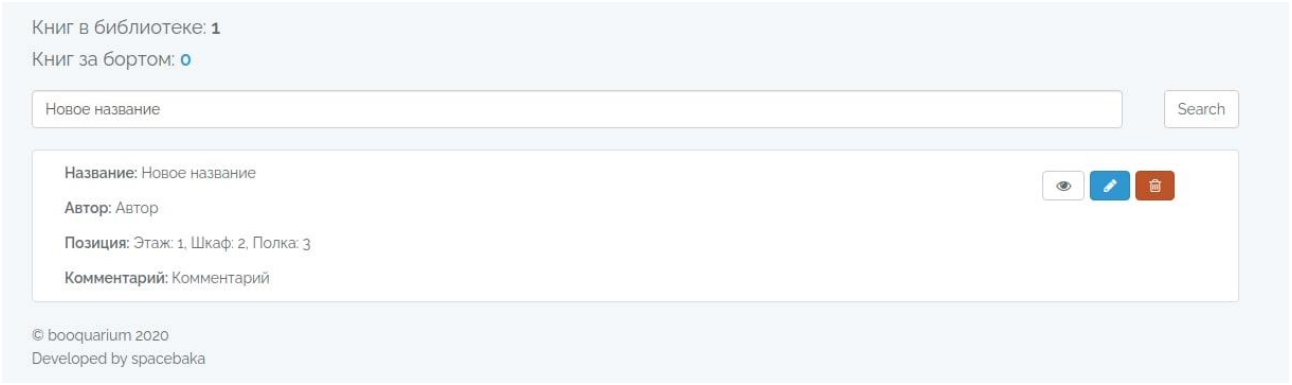


Рисунок 3.28 – Відображення результатів пошуку після редагування книги

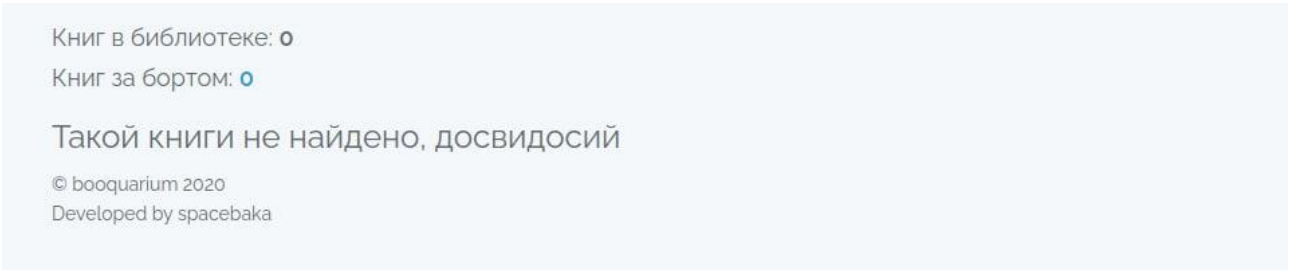


Рисунок 3.29 – Відображення результатів пошуку після видалення книги

Таблиця 3.3 Тест кейс «додавання/редагування/видалення читачів» та результати його виконання

Step	Description	Expected result	Actual result
1	Відкрити сайт http://dbs.com	Відкрито головну сторінку BOOquarium	Відкрито головну сторінку BOOquarium
2	Натиснути у верхньому	Відкрито сторінку зі	Відкрито сторінку

	правому кутку кнопку «Читачі»	списком усіх читачів	зі списком усіх читачів
3	Натиснути на сторінці кнопку «Додати читача»	Відкрито сторінку додавання читача	Відкрито сторінку додавання читача
4	Вибірково заповнити текстові поля для додавання читача та натиснути кнопку «Додати»	Неможливо зберегти читача, з'являється валідаційне повідомлення (рис.3.30)	Неможливо зберегти читача, з'являється валідаційне повідомлення
5	Заповнити всі необхідні текстові поля та натиснути кнопку «Додати»	Книгу успішно додано	Книгу успішно додано
6	Натиснути іконку редагування на книзі	Відкрито сторінку редагування читача з передзаповненими полями відповідно до внесеної раніше інформації	Відкрито сторінку редагування читача з передзаповненими полями відповідно до внесеної раніше інформації
7	Видалити текст в полі Імені	Неможливо зберегти читача, з'являється валідаційне повідомлення	Неможливо зберегти читача, з'являється валідаційне повідомлення
8	Заповнити поле назви новим значенням та натиснути кнопку «Змінити»	Читача успішно відредаговано	Читача успішно відредаговано
9	Натиснути іконку видалення на відредагованому читачі	Читача успішно видалено зі списку	Читача успішно видалено зі списку

		читачів	читачів

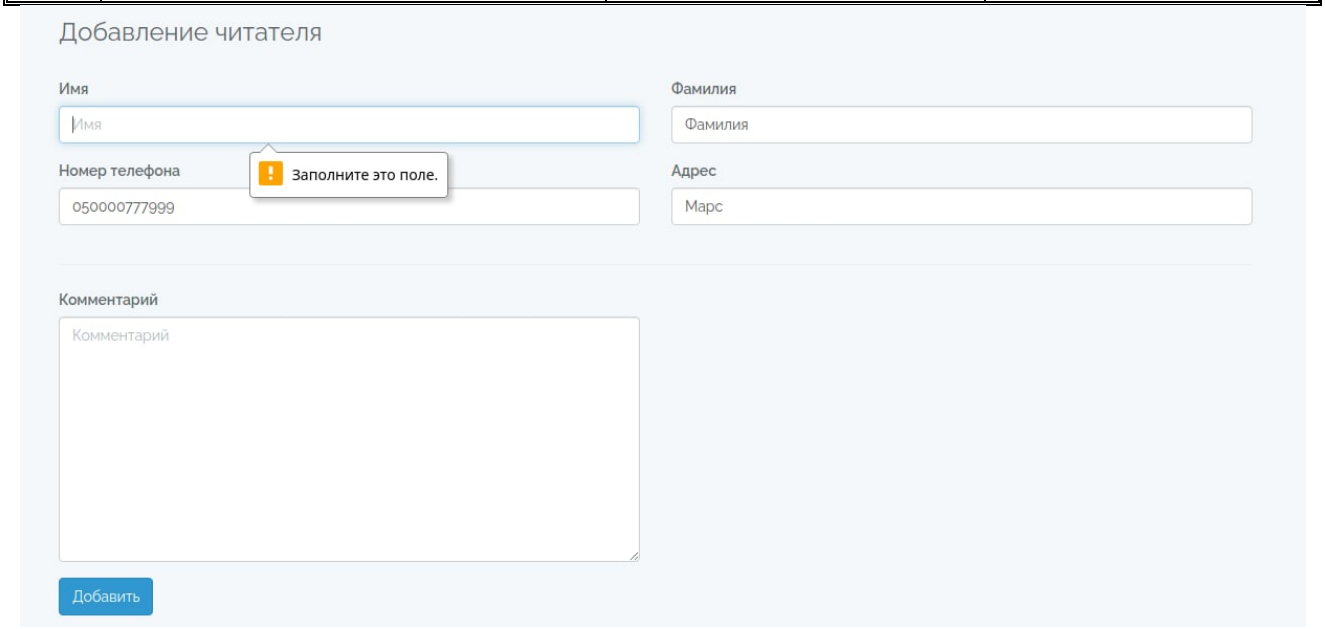


Рисунок 3.30 – Валідаційне повідомлення при необхідності заповнити поле

Таблиця 3.4 Тест кейс «видача книг читачам» та результати його виконання

Step	Description	Expected result	Actual result
1	Відкрити сайт http://dbs.com	Відкрито головну сторінку BOOquarium	Відкрито головну сторінку BOOquarium
2	Ввести в рядку пошуку назву книги доданої у попередньому кейсі	Книгу знайдено та відображено в результатах пошуку	Книгу знайдено та відображено в результатах пошуку
3	Натиснути кнопку «Дати»	Відкрито сторінку видачі книги	Відкрито сторінку видачі книги
4	Відкрити випадаючий список читачів	Відображено всіх внесених в систему читачів	Відображено всіх внесених в систему читачів
5	Зі списку вибрати користувача та натиснути кнопку «Додати»	Книгу закріплено за читачем	Книгу закріплено за читачем

6	Знайти в пошуку видану книгу	На картці книги вказано, що книга знаходиться не в бібліотеці	На картці книги вказано, що книга знаходиться не в бібліотеці
7	Натиснути іконку перегляду на знайденій книзі	Відкрито сторінку з повною інформацією про книгу, присутня кнопка «Повернути книгу», а в секції «Історія» вказано читача, у якого знаходиться книга	Відкрито сторінку з повною інформацією про книгу, присутня кнопка «Повернути книгу», а в секції «Історія» вказано читача, у якого знаходиться книга
8	Натиснути кнопку «Повернути книгу»	Книгу успішно повернуто, повідомлення про те що книги немає в бібліотеці відсутнє на картці книги	Книгу успішно повернуто, повідомлення про те що книги немає в бібліотеці відсутнє на картці книги

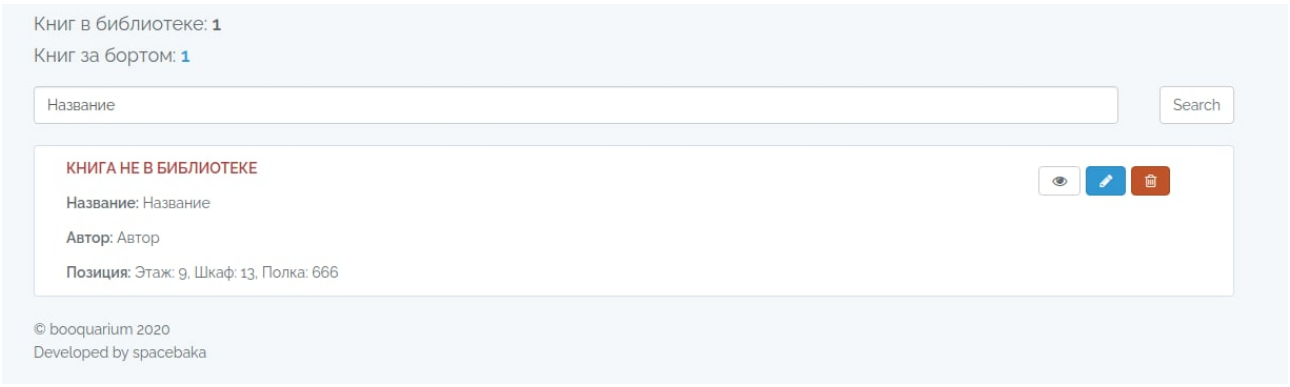


Рисунок 3.31 – Валідаційне повідомлення при необхідності заповнити поле

ВИСНОВКИ

У роботі побудована інформаційна система каталогізації бібліографічних описів з інтегрованим репозиторієм. Продемонстровано конкурентоспроможність розробленої системи з сучасними freeware та enterprise рішеннями, присутніми на ринку.

Показано, що завдяки сучасним засобам віртуалізації досягається високий рівень гнучкості процесу розробки та інтероперабельність пакету автоматизованої БІС на будь-яких операційних системах. Завдяки реалізації мікросервісного підходу наявна можливість розгалудженої підтримки окремих компонентів системи.

Продемонстровано, що мануальне тестування автоматизованої бібліотечної інформаційної системи дозволяє на пре-релізних етапах виявити та усунути проблеми, пов’язані з каталогізацією, комплектуванням та циркуляцією бібліографічних записів.

Представлена система має перспективи використання у приватних онлайн-бібліотеках (за умови наявності необхідного серверного оточення), у бібліотеках закладів початкової та середньої освіти для каталогізації та діджиталізації наявної літератури.

СПИСОК ДЖЕРЕЛ ПОСИЛАННЯ

1. Akeroy J. Integrated Library Management Systems: overview / J. Akeroy, A. Cox. // VINE. – 1999. – С. 3–10.

2. Xiaohua L. What Would be the F ould be the Future of the Integr e of the Integrated Libr ated Library Systems? y Systems? / L. Xiaohua. // Proceedings of the IATUL Conferences. – 2014. – №3.

3. Gohain A. Use and Users Satisfaction on Online Public Access Catalogue (OPAC) Services among B.T vices among B.Tech. Students of School of ech. Students of School of Engineering in T Engineering in Tezpur Univ ezpur University: a sur ersity: a survey / A. Gohain, M. Saikia. // Library Philosophy and Practice (e-journal). – 2013.

4. Librarika [Електронний ресурс] – Режим доступу до ресурсу: https://librarika.com/.

5. Evergreen [Електронний ресурс] – Режим доступу до ресурсу: https://evergreen-ils.org/.

6. LibLime Koha [Електронний ресурс] – Режим доступу до ресурсу: https://koha.org/.

7. Biblionix [Електронний ресурс] – Режим доступу до ресурсу: https://www.biblionix.com/.

8. Follett [Електронний ресурс] – Режим доступу до ресурсу: https://www.follett.com/.

9. Analysis on Web Frameworks / D.Curie, J. Jaison, J. Yadav, R. Fiona. // Journal of Physics: Conference Series. – 2019.

10. Xianjun C. Restful API Architecture Based on Laravel Framework / Chen Xianjun. // Journal of Physics Conference Series. – 2017. – №910.

11. Laravel [Електронний ресурс] – Режим доступу до ресурсу: https://laravel.com/.

12. Docker [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/.

13. Asif K. Key Characteristics of a Container Orchestration Platform to Enable a Modern Application / Khan Asif. // IEEE Cloud Computing. – 2017. – №4.

14. Bootstrap tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/bootstrap/bootstrap_tutorial.pdf.

15. Font Awesome [Електронний ресурс] – Режим доступу до ресурсу: https://fontawesome.com/.

16. Куликов С. С. Тестирование программного обеспечения. Базовый курс / / С. С. Куликов. – Минск, 2017. – 312 с. – (Четыре четверти).

17. Di Pietro R. To Docker or Not to Docker: A Security Perspective / Roberto Di Pietro. // IEEE Cloud Computing. – 2016. – №3. – С. 54–62.

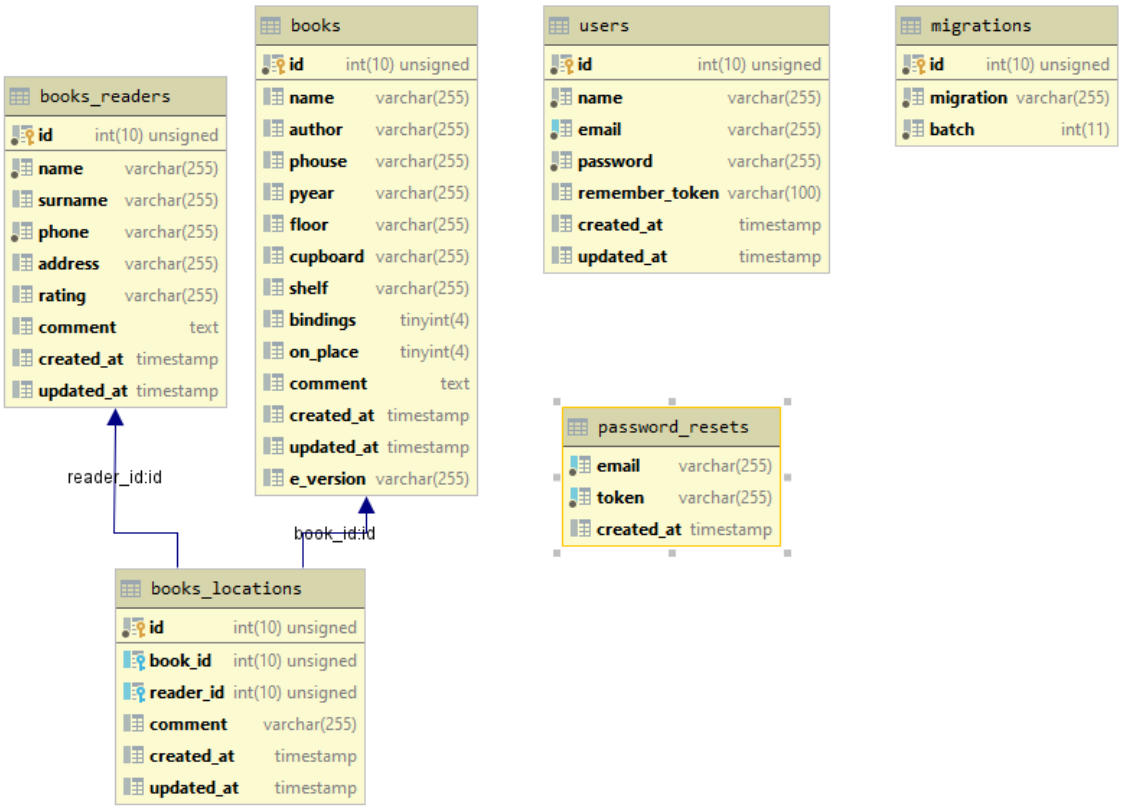
18. Ahmadi M. An Introduction to Docker and Analysis of its Performance / M. Ahmadi, H. Bhatti, B. Rad. // IJCSNS International Journal of Computer Science and Network Security. – 2017. – С. 228–235.

19. Gu Y. Study and Optimization Based on MySQL Storage Engine / Y.H. Gu. // Advances in Intelligent and Soft Computing. – 2011. – №129. – С. 85–189.

20. Letkowski J. Doing database design with MySQL / Jerzy Letkowski. // Journal of Technology Research. – 2015. – №6.

ДОДАТОК А

Діаграма зв’язків між таблицями БД



ДОДАТОК Б

Програмний код

Dockerfile для MySQL, nginx та php відповідно:

```
FROM mysql:8
RUN apt update && apt install mc -y && apt install vim -y
RUN mkdir -p /tmp/db_backup && chmod -R 777 /tmp/db_backup
VOLUME /tmp/db_backup
FROM nginx:1.15
RUN apt update && apt install mc -y && apt install vim -y
RUN echo "" > /etc/nginx/conf.d/default.conf
FROM php:7.3-fpm
# !---- default installed
# libcurl4-openssl-dev
# libedit-dev
# libsqlite3-dev
# libssl-dev
# ftp
# mbstring
# mysqlnd
# curl
# libedit
# openssl
# zlib
RUN apt-get update --fix-missing && apt-get install mc -y && apt-get install vim
-y
# xdebug
RUN pecl install xdebug \
    && docker-php-ext-enable xdebug
#        &&  echo  "zend_extension=$(find  /usr/local/lib/php/extensions/  -name
xdebug.so)" > /usr/local/etc/php/conf.d/1-xdebug.ini
#RUN    apt    install    git    -y    &&    cd    /tmp    &&    git    clone
https://github.com/xdebug/xdebug.git \
#    && cd xdebug && ./rebuild.sh \
#    && docker-php-ext-enable xdebug
#imagick
#RUN apt-get list --upgradable \
#    && apt-get update \
#    && apt-get install libmagickwand-dev imagemagick -y \
#    && pecl install imagick \
#    && echo "extension=$(find /usr/local/lib/php/extensions/ -name imagick.so)"
> /usr/local/etc/php/conf.d/imagick.ini
#redis
RUN pecl install redis \
    && docker-php-ext-enable redis
#    && echo "extension=$(find /usr/local/lib/php/extensions/ -name redis.so)" >
/usr/local/etc/php/conf.d/1-redis.ini
#memcached
RUN apt-get install -y libmemcached-dev zlib1g-dev \
    && pecl install memcached \
    && docker-php-ext-enable memcached
#          &&   echo   "extension=$(find   /usr/local/lib/php/extensions/   -name
memcached.so)" > /usr/local/etc/php/conf.d/1-memcached.ini
RUN apt-get update && apt-get install -y \
        libfreetype6-dev \
        libjpeg62-turbo-dev \
        libmcrypt-dev \
        libpng-dev \
        libmemcached-dev \
        libz-dev \
```

```
        libpq-dev \
        libjpeg-dev \
        libssl-dev \
        libxml2-dev
RUN docker-php-ext-install -j$(nproc) iconv
RUN docker-php-ext-install opcache
RUN docker-php-ext-install soap
RUN docker-php-ext-install bcmath
# GD
RUN docker-php-ext-configure gd \
    --with-gd \
    --with-jpeg-dir \
    --with-png-dir \
    --with-zlib-dir \
    --with-freetype-dir && \
    docker-php-ext-install -j$(nproc) gd
# Intl
RUN apt-get update -y && \
    apt-get install -y zlib1g-dev libicu-dev g++ && \
    docker-php-ext-configure intl && \
    docker-php-ext-install intl
# Image optimizers:
RUN apt-get update -yqq && \
    apt-get install -y --force-yes jpegoptim optipng pngquant gifsicle
# ImageMagick:
RUN apt-get update -y && \
    apt-get install -y libmagickwand-dev imagemagick && \
    pecl install imagick && \
    docker-php-ext-enable imagick
# Exif:
RUN docker-php-ext-install exif
RUN docker-php-ext-install mysqli pdo_mysql
# ZipArchive
RUN apt-get install -y libzip-dev && docker-php-ext-install zip
# composer
RUN cd ~ \
    && curl -sS https://getcomposer.org/installer | php \
    && mv composer.phar /usr/bin/composer
# install ffmpeg
RUN apt-get install ffmpeg -y
#timezone db
RUN pecl install timezonedb \
    && docker-php-ext-enable timezonedb
COPY ./conf.d/custom_php.ini /usr/local/etc/php/conf.d
COPY ./conf.d/opcache.ini /usr/local/etc/php/conf.d
COPY ./conf.d/xdebug.ini /usr/local/etc/php/conf.d
####################################
# User Aliases
####################################
COPY ./aliases.sh /root/aliases.sh
USER root
RUN echo "" >> ~/.bashrc && \
    echo "# Load Custom Aliases" >> ~/.bashrc && \
    echo "source /root/aliases.sh" >> ~/.bashrc && \
    echo "" >> ~/.bashrc && \
    sed -i 's/\r//' /root/aliases.sh && \
    sed -i 's/^#! \/bin\/sh/#! \/bin\/bash/' /root/aliases.sh
VOLUME /var/www/html
CMD ["php-fpm"]
EXPOSE 11211
```

docker-compose.yml

```
version: '2.4'
services:
  web:
    build: ./services/web
    container_name: '${CONTAINER_PREFIX}_web'
    restart: ${CONTAINER_RESTART}
    volumes:
      - '${WWW_DIR}:/var/www/html'
    tty: true
  php:
    build: ./services/php
    container_name: '${CONTAINER_PREFIX}_php'
    restart: ${CONTAINER_RESTART}
    volumes_from:
      - web
    depends_on:
      - db
    links:
      - db
    ports:
      - '9000:9000'
    extra_hosts:
      - 'dockerhost:${DOCKER_HOST_IP}'
  nginx:
    build: ./services/nginx
    restart: ${CONTAINER_RESTART}
    container_name: '${CONTAINER_PREFIX}_nginx'
    depends_on:
      - php
      - myadmin
    volumes:
      - './config/nginx/site.conf:/etc/nginx/conf.d/site.conf'
      - './data/logs/nginx:/var/log/nginx'
    environment:
      - NGINX_HOST=localhost
      - NGINX_PORT=80
    volumes_from:
      - web
    ports:
      - '${NGINX_FORWARD_PORT}:80'
      - '${NGINX_FORWARD_PORT_SSL}:443'
    networks:
      default:
        aliases:
          - api.heyproducer.loc
  db:
    build: ./services/db
    volumes:
      - './data/db/:/var/lib/mysql'
      - './data/bkp/:/tmp/db_backup'
      - './config/db/:/etc/mysql/conf.d'
    restart: ${CONTAINER_RESTART}
    container_name: '${CONTAINER_PREFIX}_db'
    environment:
      - 'MYSQL_DATABASE=${MYSQL_DATABASE}'
      - 'MYSQL_USER=${MYSQL_USER}'
      - 'MYSQL_PASSWORD=${MYSQL_PASSWORD}'
      - 'MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}'
    ports:
      - '${MYSQL_FORWARD_PORT}:3306'
```

```
  myadmin:
    image: phpmyadmin/phpmyadmin:${PMA_VERSION}
    container_name: '${CONTAINER_PREFIX}_phpmyadmin'
    depends_on:
      - db
```

контролери:

BooksController:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Input;
use Illuminate\Support\Facades\Redirect;
use App\Book;
use App\BooksReadersHistory;
use App\BooksReader;
use Carbon\Carbon;
use Illuminate\Support\Facades\Storage;

class BooksController extends Controller
{
    public function index(Request $request) {
        if (Input::get('outside_books') == 1) {
            $books = Book::where('book_reader_id', '!=', null)->paginate(10);
        } else {
            if (!empty($request)) {
                $query = $request->query('search');
                $books = Book::where('name', 'LIKE', '%' . $query . '%')
                    ->orWhere('author', 'LIKE', '%' . $query . '%')
                    ->orWhere('phouse', 'LIKE', '%' . $query . '%')
                    ->orWhere('pyear', 'LIKE', '%' . $query . '%')
                    ->orWhere('comment', 'LIKE', '%' . $query . '%')
                    ->orderBy('name', 'desc')
                    ->paginate(10);
            } else {
                $books = Book::orderBy('name', 'desc')->paginate(10);
            }
        }

        $books_outside = Book::where('book_reader_id','!=',null)->count();
        return view('books.index', ['books' => $books, 'books_outside' =>
$books_outside]);
    }

    public function create()
    {
        return view('books.add');
    }

    public function view($id)
    {
        $book = Book::find($id);
        $book_histories = BooksReadersHistory::where('book_id', '=', $id)-
>orderBy('created_at', 'desc')->get();
        return view('books.view', ['book' => $book, 'book_histories' =>
$book_histories]);
    }

    public function return_book($id) {
```

```
        $book_history   =   BooksReadersHistory::where('book_id',   '=',   $id)-
>first();
        $book_history->rent_end_time = Carbon::now()->format('Y-m-d H:i:s');
        $book_history->save();
        $book = Book::find($id);
        $book->book_reader_id = null;
        $book->save();
        return Redirect::to('/');
    }

    public function show_outside_books() {

    }

    public function edit($id)
    {
        $book = Book::find($id);
        return view('books.add', ['book' => $book]);
    }

    public function update(Request $request, Book $book)
    {
        $filename = $request->file('e_version')->store('e_versions');
        $book->fill($request->all());
        $book->e_version = $filename;
        $book->save();
        return Redirect::to('/');
    }

    public function store(Request $request)
    {
        $book = new Book;
        $data = $request->all();
        $filename = $request->file('e_version')->store('e_versions');
        $book->fill($data);
        $book->e_version = $filename;
        $book->save();
        return Redirect::to('/');
    }

    public function books_download(Request $request, Book $book) {
        return          response()->download(Storage::disk('local')->path($book-
>e_version));
    }

    public function destroy($id) {
         $book = Book::find($id);
         $book->delete();
         return Redirect::to('/');
    }
}
```

BooksReadersHistory

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Redirect;
use App\Book;
use App\BooksReadersHistory;
use App\BooksReader;
```

```
class BooksReadersHistoryController extends Controller
{
    /
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

    /
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create($book_id)
    {
        $readers = BooksReader::pluck('name', 'id');
        $book_history = BooksReadersHistory::find($book_id);
        return  view('books_history.add',  ['books_history'  =>  $book_history,
'readers' => $readers, 'book_id' => $book_id]);
    }

    /
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $book = Book::find($request->book_id);
        $book->book_reader_id = $request->reader_id;
        $book->save();
        $data = $request->all();
        $history_item = new BooksReadersHistory();
        $history_item->create($data);
        return Redirect::to('/');
    }

    /
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
```

```
    }

    /
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        //
    }
}
```

ReadersController:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Redirect;
use App\BooksReader;
use App\BooksReadersHistory;

class ReadersController extends Controller
{
    /
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $readers = BooksReader::paginate(10);
        return view('readers.index', ['readers' => $readers]);
    }

    /
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        return view('readers.add');
    }

    /
     * Store a newly created resource in storage.
     *
```

```
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $reader = new BooksReader;
        $data = $request->all();
        $reader->create($data);
        return Redirect::to('readers/');
    }

    /
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        $reader = BooksReader::find($id);
        $book_histories  =  BooksReadersHistory::where('reader_id',  '=',  $id)-
>get();
        return  view('readers.view',  ['reader'  =>  $reader,  'book_histories'  =>
$book_histories]);
    }

    /
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        $reader = BooksReader::find($id);
        return view('readers.add', ['reader' => $reader]);
    }

    /
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        $reader = BooksReader::find($id);
        $reader->update($request->all());
        return Redirect::to('readers/');
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        $reader = BooksReader::find($id);
        $reader->delete();
        return Redirect::to('readers/');
```

```
    }
}
```

routes:

```
<?php

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

//Route::get('/', ['as' => 'book_search', 'uses' => 'BooksController@index']);
Route::get('/', 'BooksController@index')->name('book_index');
Route::get('/?outside_books=1',                          'BooksController@index')-
>name('show_outside_books');
Route::get('add', ['as' => 'book_add', 'uses' => 'BooksController@create']);
Route::post('add', ['as' => 'book_store', 'uses' => 'BooksController@store']);
Route::get('delete/{id}',     ['as'     =>     'book_delete',     'uses'     =>
'BooksController@destroy']);
Route::get('return/{id}',     ['as'     =>     'book_return',     'uses'     =>
'BooksController@return_book']);
Route::get('view/{id}',      ['as'      =>      'book_view',      'uses'      =>
'BooksController@view']);
Route::get('edit/{id}',      ['as'      =>      'book_edit',      'uses'      =>
'BooksController@edit']);
Route::post('edit/{book}',     ['as'     =>     'book_update',     'uses'     =>
'BooksController@update']);
Route::get('books_history/',                    'BooksController@books_history')-
>name('book_location');
Route::get('book/{book}/download',            'BooksController@books_download')-
>name('books_download');

/* Reader routes */
Route::get('readers/', 'ReadersController@index')->name('readers_index');
Route::get('readers/add', 'ReadersController@create')->name('reader_add');
Route::post('readers/add', 'ReadersController@store')->name('reader_store');

Route::get('readers/view/{id}', 'ReadersController@show')->name('reader_view');
Route::get('readers/edit/{id}', 'ReadersController@edit')->name('reader_edit');
Route::post('readers/edit/{id}',                    'ReadersController@update')-
>name('reader_update');

Route::get('readers/delete/{id}',                 'ReadersController@destroy')-
>name('reader_delete');

/* Book Location */
Route::get('books_readers_history/add/{id}',
'BooksReadersHistoryController@create')->name('book_history_add');
Route::post('books_readers_history/add/',
'BooksReadersHistoryController@store')->name('book_history_store');
custom.css:
ul {
    list-style: none;
}

.header {
```

```
}
    .header .container {
        overflow: hidden;
    }
        .header .container {
        }
            .header .container h1 {
                float: left;
            }
            .header .container ul.menu {
                float: right;
                margin: 30px 0 0 0;
                overflow: hidden;
            }
                .header .container ul.menu li {
                    float: left;
                    margin: 0 0 0 30px;
                }

.location-block {
    margin: 20px 0 0 0;
}
.search-table {
    margin: 30px 0 0 0;
}
    .search-table table {
        background: white;
    }
        .search-table table td a {
            display: block;
            float: left;
        }
        .search-table table td a:nth-child(2) {
            margin-left: 2px;
        }
.list-group-item {
    padding-bottom: 0;
}
.actions-wrp {
    height: 45px;
    margin-bottom: 10px;
    display: flex;
    align-items: center;
}
    .actions-wrp a {
        display: block;
        margin: 0 5px 0 0;
    }

.search-row {
    margin: 20px 0;
    overflow: hidden;
}
    .search-row .col-md-11 {
        padding-left: 0;
    }
    .search-row .col-md-1 {
        padding-right: 0;
        padding-left: 24px;
    }

.margin {
```

```
    margin: 0 0 30px 0;
}
.give_book_btn {
    margin-left: 0;
    margin-bottom: 10px;
}
```

BookModel:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Book extends Model
{
    protected $fillable = ['name', 'author', 'phouse', 'pyear', 'comment',
'floor', 'cupboard', 'shelf', 'bindings', 'on_place', 'e_version'];
}
```

BookReadersModel:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class BooksReader extends Model
{
    protected $fillable = ['name', 'surname', 'phone', 'address', 'comment'];
}
```

BooksReadersHistoryModel:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class BooksReadersHistory extends Model
{
    protected $fillable = ['book_id', 'reader_id', 'comment'];

    public function reader() {
        return $this->belongsTo(BooksReader::class, 'reader_id');
    }

    public function book() {
        return $this->belongsTo(Book::class, 'book_id');
    }
}
```

Books index view:

```
@extends('layouts.master')
@section('content')
    <h4>Книг в библиотеке: <strong>{{$books->total()}}</strong></h4>
    <h4>Книг          за          бортом:          <strong><a          href="{{
URL::route('show_outside_books')}}">{{$books_outside}}</a></strong></h4>
    @if (count($books))
        <div class="search-row">
        {!! Form::open(array('route' => 'book_index', 'method' => 'GET', 'class'
=> 'form')) !!}
            <div class="col-md-11">
            {!! Form::text('search', null,
```

```
                    array('required',
                          'class'=>'form-control',
                          'placeholder'=>'Поиск')) !!}
            </div>
            <div class="col-md-1">
            {!! Form::submit('Search',
                                array('class'=>'btn btn-default')) !!}
            </div>
        {!! Form::close() !!}
        </div>
        <ul class="list-group">
            @foreach($books as $key => $book)
                <li class="list-group-item">
                    <div class="col-md-10">
                        @if ($book->book_reader_id != null)<p class="text-danger
text-uppercase"><b>Книга не в библиотеке</b></p>@endif
                        <p><b>Название:</b> {{$book->name}}</p>
                        <p><b>Автор:</b> {{$book->author}}</p>
                        <p><b>Позиция:</b>    Этаж:    {{$book->floor}},    Шкаф:
{{$book->cupboard}}, Полка: {{$book->shelf}}</p>
                        @if (!empty($book->comment))
                            <p><b>Комментарий:</b> {{$book->comment}}</p>
                        @endif
                        @if           ($book->e_version)<p><a           href="{{
URL::route('books_download', $book->id) }}" class="btn btn-default"><i class="fa
fa-download" aria-hidden="true"></i> Скачать</a></p>@endif
                    </div>
                    <div class="col-md-2">
                        <div class="actions-wrp">
                            <a  href="{{  URL::route('book_view',  $book->id)  }}"
class="btn btn-default"><i class="fa fa-eye" aria-hidden="true"></i></a>
                            <a  href="{{  URL::route('book_edit',  $book->id)  }}"
class="btn btn-primary"><i class="fa fa-pencil" aria-hidden="true"></i></a>
                            <a href="{{ URL::route('book_delete', $book->id) }}"
class="btn     btn-danger     delete"><i     class="fa     fa-trash-o"     aria-
hidden="true"></i></a>
                        </div>
                    </div>
                    <div style="clear: both;"></div>
                </li>
            @endforeach
        </ul>
        {{ $books->links() }}
        @else
            <h3>Такой книги не найдено, досвидосий</h3>
        @endif
@stop
```

add/edit book view:

```
@extends('layouts.master')
@section('content')

    @if(isset($book))
{{--        {!! Breadcrumbs::render('book_edit', $book) !!}--}}
        <h3 class="margin">Редактирование книги "{{$book->name}}"</h3>
        {{ Form::model($book, ['route' => ['book_update', $book->id], 'files' =>
true]) }}
    @else
{{--        {!! Breadcrumbs::render('book_add') !!}--}}
        <h3 class="margin">Добавление книги</h3>
        {!! Form::open(array('route' => 'book_store', 'class' => 'form', 'files'
=> true)) !!}
```

```
    @endif
    <div class="row">
        <div class="col-md-6 form-group">
            {!! Form::label('Название книги') !!}
            {!! Form::text('name', null,
                array('required',
                      'class'=>'form-control',
                      'placeholder'=>'Название книги')) !!}
        </div>
        <div class="col-md-6 form-group">
            {!! Form::label('Автор книги') !!}
            {!! Form::text('author', null,
                array('required',
                      'class'=>'form-control',
                      'placeholder'=>'Автор')) !!}
        </div>
    </div>
    <div class="row">
        <div class="col-md-6 form-group">
            {!! Form::label('Название издательства') !!}
            {!! Form::text('phouse', null,
                array('required',
                      'class'=>'form-control',
                      'placeholder'=>'Название издательства')) !!}
        </div>
        <div class="col-md-6 form-group">
            {!! Form::label('Год издания') !!}
            {!! Form::text('pyear', null,
                array('required',
                      'class'=>'form-control',
                      'placeholder'=>'Год издания',
                      'maxlength' => 4)) !!}
        </div>
    </div>
    <hr/>
    <div class="row">
        <div class="col-md-12 form-group">
            {!! Form::label('Переплет') !!}
            {!!  Form::select('bindings', ['1' => 'Твердый', '2' => 'Мягкий'],
null, ['class' => 'form-control']) !!}
        </div>
    </div>
    <div class="row">
        <div class="col-md-12 form-group">
            {!! Form::label('Этаж') !!}
            {!!  Form::text('floor', null, ['required', 'class'=>'form-control',
'placeholder'=> 'Этаж']) !!}
        </div>
        <div class="col-md-12 form-group">
            {!! Form::label('Шкаф') !!}
            {!!   Form::text('cupboard',   null,   ['required',   'class'=>'form-
control', 'placeholder'=> 'Шкаф']) !!}
        </div>
        <div class="col-md-12 form-group">
            {!! Form::label('Полка') !!}
            {!!  Form::text('shelf', null, ['required', 'class'=>'form-control',
'placeholder'=> 'Номер полки сверху']) !!}
        </div>
    </div>
    <hr/>
    <div class="row">
        <div class="col-md-6 form-group">
            {!! Form::label('Комментарий') !!}
```

```
            {!! Form::textarea('comment', null,
                array('class'=>'form-control')) !!}
        </div>
    </div>
    <div class="row">
        <div class="col-md-6 form-group">
            {!! Form::label('Электронная версия') !!}
            {!! Form::file('e_version', null,
                array('class'=>'form-control',)) !!}
        </div>
    </div>
    <span class="clearfix">

    <div class="form-group">
        @if(isset($book))
            {!! Form::submit('Изменить',
          array('class'=>'btn btn-primary')) !!}
        @else
            {!! Form::submit('Добавить',
          array('class'=>'btn btn-primary')) !!}
        @endif

    </div>
    {!! Form::close() !!}
@stop
```

book view view:

```
@extends('layouts.master')
@section('content')
{{--    {!! Breadcrumbs::render('book_edit', $book) !!}--}}

    <h3>Просмотр книги "{{$book->name}}"</h3>

    <ul class="list-group">
        <li class="list-group-item">
            <div class="col-md-11">
                <p><b>Название:</b> {{$book->name}}</p>
                <p><b>Автор:</b> {{$book->author}}</p>
                <p><b>Издательство:</b> {{$book->phouse}}</p>
                <p><b>Год издательства:</b> {{$book->pyear}}</p>
                <p><b>Позиция:</b>  Этаж:  {{$book->floor}},  Шкаф:  {{$book-
>cupboard}}, Полка: {{$book->shelf}}</p>
                <p><b>Переплет:</b> @if($book->bindings == 1) Твердый @else
Мягкий @endif</p>
                <p><b>Комментарий:</b> {{$book->comment}}</p>
                <p><b>Книга в библиотеке:</b> @if ($book->book_reader_id ==
null)Да@else Нет @endif</p>
                @if               ($book->e_version)<p><a               href="{{
URL::route('books_download', $book->id) }}" class="btn btn-default"><i class="fa
fa-download" aria-hidden="true"></i> Скачать</a></p>@endif
            </div>
            <div class="col-md-1">
                <div class="actions-wrp">
                    <a   href="{{   URL::route('book_edit',   $book->id)   }}"
class="btn btn-primary"><i class="fa fa-pencil" aria-hidden="true"></i></a>
                    <a   href="{{   URL::route('book_delete',   $book->id)   }}"
class="btn btn-danger"><i class="fa fa-trash-o" aria-hidden="true"></i></a>
                </div>
            </div>
            <div style="clear: both;"></div>
        </li>
    </ul>
```

```
    @if ($book->book_reader_id == null)
        <div           class="row           give_book_btn"><a           href="{{
URL::route('book_history_add', $book->id) }}" class="btn btn-primary">Дать кни-
гу</a></div>
    @else
        <div  class="row  give_book_btn"><a  href="{{  URL::route('book_return',
$book->id) }}" class="btn btn-primary">Вернуть книгу</a></div>
    @endif

    @if (count($book_histories))
        <h3>История</h3>
        <ul class="list-group">
        @foreach ($book_histories as $book_history)
            <li class="list-group-item">
                <p><b>Когда взял:</b> {{$book_history->created_at}}</p>
                @if ($book_history->rent_end_time != null)
                    <p><b>Когда отдал:</b> {{$book_history->rent_end_time}}</p>
                @endif
                <p><b>У кого книга:</b>  <a href="{{ URL::route('reader_view',
$book_history->reader->id)  }}">{{$book_history->reader->name}}  {{$book_history-
>reader->surname}}</a></p>
                <p><b>Номер телефона:</b> {{$book_history->reader->phone}}</p>
                @if ($book_history->comment)
                    <p><b>Комментарий:</b> {{$book_history->comment}}</p>
                @endif
            </li>
        @endforeach
        </ul>
    @endif
@stop
```

header layout

```
<div class="header">
    <div class="container">
        <h1><a         href="{{         URL::route('book_index')         }}"><img
src="{{URL::asset('images/header/booquakium-logo.jpg')}}"/></a></h1>
        <ul class="menu">
            <li><a href="{{ URL::route('book_index') }}" class="btn btn-primary
@if  (Request::path()  ==  '/')  active  @endif"><i  class="fa  fa-search"  aria-
hidden="true"></i> Поиск</a></li>
            <li><a  href="{{  URL::route('book_add')  }}"  class="btn  btn-success
@if  (Request::path()  ==  'add')  active  @endif"><i  class="fa  fa-plus"  aria-
hidden="true"></i> Добавить книгу</a></li>
            <li><a  href="{{  URL::route('readers_index')  }}"  class="btn  btn-
success">Читатели</a></li>
        </ul>
    </div>
</div>
<div class="container">
    <p>Alpha version 0.8</p>
</div>
<hr/>
```

footer layout

```
<div class="container">
    &copy; booquarium 2020<br/>
    Developed by spacebaka
</div>
```