OurLib

Programing and Internet Technologies

Kornel Stefańczyk
Technical University of Varna
Erasmus+ student from:
Wrocław University of Science and Technology

28 January 2021
version: 1.1

Contents

1 Requirements

Design and implementation of an information system - Library. The system must store and processing data for books and readers. The system allows multiple access (Many users in same time). The system supports two types of users - administrator and clients (operators and readers) with different roles to access the functionalities in the system.

1.1 Users

User operations:

- Creation of operators by an administrator
- Creating and unsubscribing readers from an operator
- Form for creating a reader profile
- Renting books

1.2 Books

The system supports operations for working with books:

- Adding new books (Inventory number, title, author, genre…)
- Borrowing books (different level of security when borrowing books (reading room, for home))
- Return of a book
- Disposal of damaged books
- Archiving of old editions (using only in the reading room)

1.3 Reports

The system maintains reports for:

- Submitted forms (date, status, content of the form)
- Books (Book status, book information)
- Users (approval date, book list, user information)
- User rating (loyal and disloyal readers)

1.4 Event Notifications

The system supports Event Notifications(working with threads):

- Request for opening a reader profile
- Notification of the need to archive a book
- Delayed return of books

2 System modules

OurLib project can be divided in three main modules.

2.1 Database

Database module is obligatory to store users and books data.

2.2 Backend

Backend module is the bridge between database module and GUI module.

This module is responsible for implementation of business logic. Backend receive data from GUI, validate it and communicate with database to read, create or modify data stored in database module.

2.3 GUI

Graphical user interface module is only available for system user module . This module is responsible for communication with backend module. GUI interface changes according to data get from backend module. It presents all features available for user using graphical elements.

3 System design

3.1 Database

MySQL was selected as database service technology.

ERD schematic is presented at the figure 1.

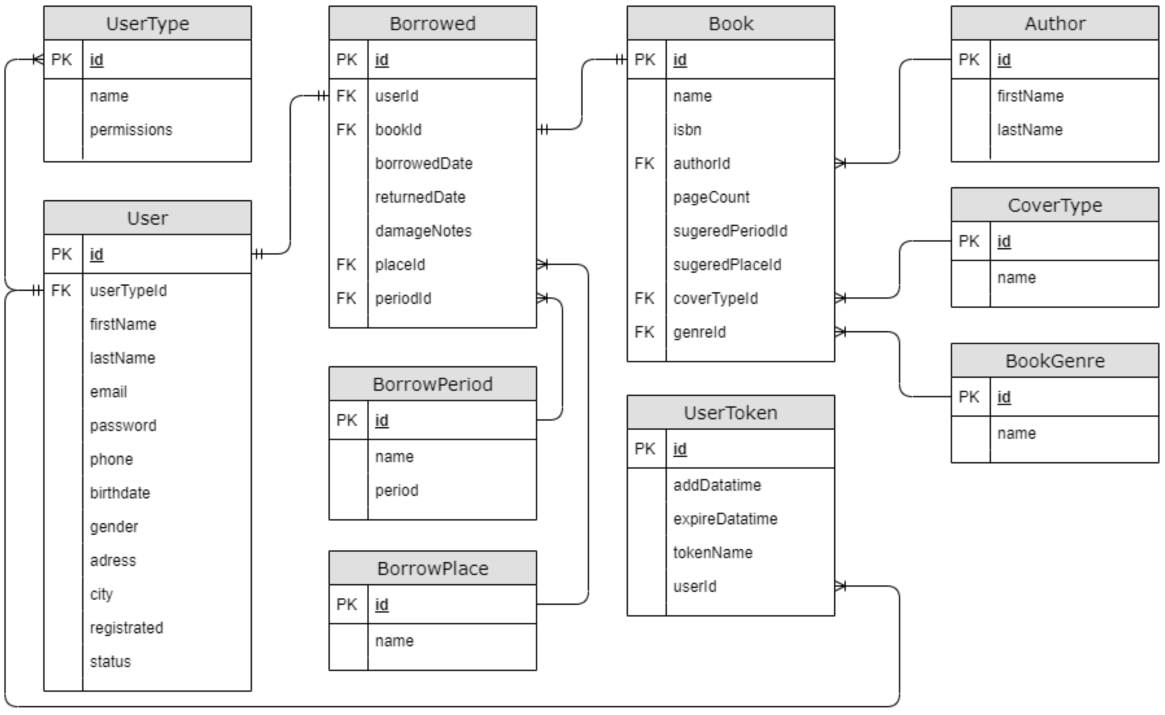


Figure 1: ERD schematic diagram

Database tables:

- *UserType* table should be prepared before system deployment. Data from this table will be not available to change from the system.
- *User* table contains user data. Each of users in the system has separated entry. User type is defined by userTypeId that reefers to the *UserType* table.
- *UserToken* table contains security information about logged in users.
- *BorrowedPeriod* table contains information about how long book can be borrowed by user.
- *BorrowedPlace* table contains the name of place where book can be read.

- *BookGenere* table contains name of the type of the book.
- *CoverType* table contains all possible cover types of the books in library.
- *Author* table store basic personal data of the authors of the books.
- *Book* table store all books existing in library.
- *Borrowed* table store all borrow orders details.

3.2 Backend

Backend of the system is based on two main sub modules:

- Business logic and data layer
- RestAPI

3.2.1 Business logic and data layer

Backend uses Java SpringBoot technology.

Data layer uses Hibernate. It allows to perform all CRUD methods using JPA api. Including setup of database.

3.2.2 RestAPI

RestApi was designed using Swagger API Documentation and Development tool. It allows to generate models of classes containing data and Rest points structure. Source code of RestAPI specification can be found in Appendix A

3.3 Graphical user interface

Graphical user interface is based on JavaScript React framework.

4 Implementation

4.1 Database

MySQL database *library_db* was set up and new user account was created. Backend is connected to database through *api* user.

Backend create on startup default values needed for system set up.

4.2 Backend

4.2.1 Models

Models of data were generated using Swagger web app. Definition of all models can be found in section 6. For example definition of User class in Swagger model is presented in listing 1.

Listing 1: User model

```
  User:
    type: object
    properties:
      id:
        type: integer
        format: int64
      userTypeId:
        type: integer
        format: int64
      firstName:
        type: string
      lastName:
        type: string
      email:
        type: string
      password:
        type: string
      phone:
        type: string
      birthdate:
        type: string
        format: date-time
      registrated:
        type: string
        format: date-time
      gender:
        type: string
        enum:
          - male
          - female
          - other
      address:
        type: string
      city:
        type: string
      status:
        type: string
        description: User Status
        enum:
          - active
          - suspended
          - inactive
          - to veryfication
    xml:
      name: User
  UserType:
```

It was needed to add annotation from *javax.persistence* like: *Entity*, *GeneratedValue*, *GenerationType*, *Id*, *Table* to default generated models. These annotations are using to communicate with database properly. Example of setting up code can be seen in listing 2.

Listing 2: Model persistence annotations

```java
@Entity
@Table(name = "book")
@JsonRootName("Book")
public class Book   {
  @Id
  @GeneratedValue(strategy=GenerationType.AUTO)
  @JsonProperty("id")
  private Long id = null;

  @JsonProperty("name")
  private String name = null;
```

4.2.2 Services

Services are implementation of business logic and all needed manipulation of data in database.

Each of main data segments had implemented needed functionalities. Example book service functionalities are included in listing 3.

Listing 3: Book services interface

```
public interface BookService {
  public Void initBookValues();
  public int countBooks();
  public int countAvailableBooks();
  public Book createBook(Book book);
  public List<Book> getAllBooks();
  public Book deleteBookById(Long id);
  public Book getBookById(Long id);
  public String getBookNameById(Long id);
  public Book updateBookById(Long id, Book body);
  public Boolean updateBookStatus(Long id, StatusEnum status);
  public List<Book> getAllAvailableBooks();
```

Services were using JPA repositories to manipulate data stored in database like in listing 4.

Listing 4: User service create user

```
@Override
public User createUser(User user) {
  OffsetDateTime registrated = OffsetDateTime.now(ZoneId.of("UTC"));
  user.setRegistrated(registrated);

  Optional<User> od = userRepository.getUserByEmail(user.getEmail());
  if(!od.isPresent()) {
      log.info("Create new user:\n"+user.toString());
      return userRepository.save(user);
  } else {
      log.error("Sorry, user with  given email exist. Email: "+user.
          getEmail());
  }
  return null;
}
```

4.2.3 RestAPI

Models of data were generated using Swagger web app. The RestAPI model was defined in Swagger 2.0 format. Definition of whole RestAPI is included in section 6.

Definition of *Create User* method was described in Swagger 2.0 format in listing 5.

Listing 5: Create user Swagger 2.0 definition

```
paths:
  /user:
    post:
      tags:
      - user
      summary: Create user
      description: This can only be done by the logged in user.
      operationId: createUser
      consumes:
      - application/json
      parameters:
      - in: body
        name: body
        description: Created user object
        required: true
```

```
          schema:
            $ref: '#/definitions/User'
        - name: token
          in: query
          required: false
          type: string
        responses:
          200:
            description: successful operation
          400:
            description: "Bad request"
          401:
            description: Unauthorized, no access
          403:
            description: "Forbidden"
        security:
        - bearerAuth: []
```

Output generated from Swagger web app was as two files describing RestAPI interface. First was containing interface of RestAPI methods. Example of those functions is in listing 6.

Listing 6: Create user interface generated by Swagger

```
  @ApiOperation(value = "Create user", nickname = "createUser", notes = "This can only be done by the logged in user.", authorizations = {
  @Authorization(value = "bearerAuth")
}, tags={ "user", })
@ApiResponses(value = {
  @ApiResponse(code = 200, message = "successful operation"),
  @ApiResponse(code = 400, message = "Bad request"),
  @ApiResponse(code = 401, message = "Unauthorized, no access"),
  @ApiResponse(code = 403, message = "Forbidden") })
@RequestMapping(value = "/user",
  consumes = { "application/json" },
  method = RequestMethod.POST)
ResponseEntity<Void> createUser(@ApiParam(value = "Created user object" ,required=true ) @Valid @RequestBody User body,@ApiParam(value = "") @Valid @RequestParam(value = "token", required = false) String token);
```

Second file was containing implementation of RestAPI functionalities. Example from this file is listing 7. In those implementation usage of different HTTP status codes can be seen.

Listing 7: Create user implementation

```
public ResponseEntity<Void> createUser(@ApiParam(value = "Created user object" ,required=true ) @Valid @RequestBody User body,@ApiParam(value = "") @Valid @RequestParam(value = "token", required = false) String token) {
  String accept = request.getHeader("Accept");
  if (accept != null && (accept.contains("application/json") || accept.contains("*/*")) ){
     if(userService.isModifyPermittedForToken(body, token)){
         body = userService.initNewUserAccordingToPermissions(body, token);
         if(userService.createUser(body) != null){
             return new ResponseEntity<Void>(HttpStatus.OK);
         } else {
             return new ResponseEntity<Void>(HttpStatus.FORBIDDEN);
         }
```

```
        } else {
            return new ResponseEntity<Void>(HttpStatus.UNAUTHORIZED);
        }
    } else {
        return new ResponseEntity<Void>(HttpStatus.BAD_REQUEST);
    }
}
```

4.3 Graphical user interface

GUI was written in React framework.

4.3.1 Connection with backend

Connection with RestAPI was realized by HTTP requests like *POST*, *PUT*, *GET*, *DELETE*. Example of *POST* and *PUT* requests are in listing 8.

Listing 8: Connection with backend through RestAPI

```
token: auth.authState.token
}
setSaveLoadingBookFullDetailsForm(true);
const { data } = credentials.id ? (
    await fetchContext.authAxios.put(
        `/book/` + credentials.id + '?' + qs.stringify(queryValues),
        saveBookItem
    )
) : (
        await fetchContext.authAxios.post(
            `/book?` + qs.stringify(queryValues),
            saveBookItem
        )
    );
```

4.3.2 Sidebar

Main navigation element of web app is sidebar. It changes according to logged in user. Each of user type has different set of visible buttons on sidebar. Listing 9 contains possible endpoints with label and allowed role for each button. Button will be visible only when user type will be included in allowedRoles array.

Listing 9: Sidebar buttons with allowed user types

```
const navItems = [
    {
        label: 'components.sidebar_component.link_label.find_books',
        path: '/find-books',
        icon: faBook,
        allowedRoles: [reader, librarian]
    },
    {
        label: 'components.sidebar_component.link_label.orders',
        path: '/manage-orders',
        icon: faBookOpen,
        allowedRoles: [reader]
    },
    {
        label: 'components.sidebar_component.link_label.manage_books',
        path: '/manage-books',
        icon: faEdit,
        allowedRoles: [librarian]
    },
    {
        label: 'components.sidebar_component.link_label.manage_orders',
        path: '/manage-orders',
        icon: faTasks,
        allowedRoles: [librarian]
    },
    {
        label: 'components.sidebar_component.link_label.all_users',
        path: '/users',
        icon: faUsersCog,
        allowedRoles: [admin]
    },
    {
        label: 'components.sidebar_component.link_label.readers',
        path: '/readers',
        icon: faBookReader,
        allowedRoles: [admin, librarian]
    },
    {
        label: 'components.sidebar_component.link_label.reports',
        path: '/reports',
        icon: faList,
        allowedRoles: [admin]
    },
    {
        label: 'components.sidebar_component.link_label.account',
        path: '/account',
        icon: faAddressCard,
        allowedRoles: [reader, librarian, admin]
    }
];
```

4.3.3 Translations

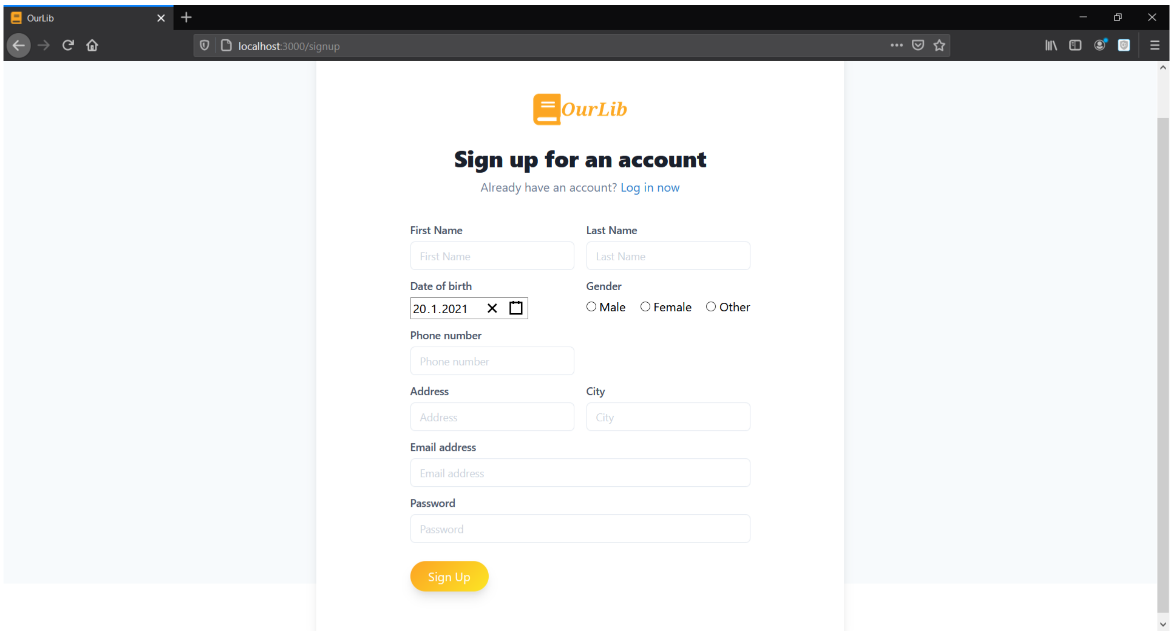
Translations was realized using *react-i18next* component. Translations are stored in common JSON file with translations. Translate of needed text is done by using *useTranslation* hook. Example of execution is *t('components.book_item_overview_short_component.labels.book_id_header')* that contains path of structure in JSON translation files that is used as an object. Adding new file with translated texts it is possible to change texts in whole web app without changing source code of app.

4.3.4 Forms

Forms was created using Formik component. Most of fields was standard text or number fields. The rest of input components was *react-select* and *react-date-picker* components.

4.3.5 Screenshots of GUI

Screenshots of different web app views are presented below.



OurLib

localhost:3000/signup

OurLib

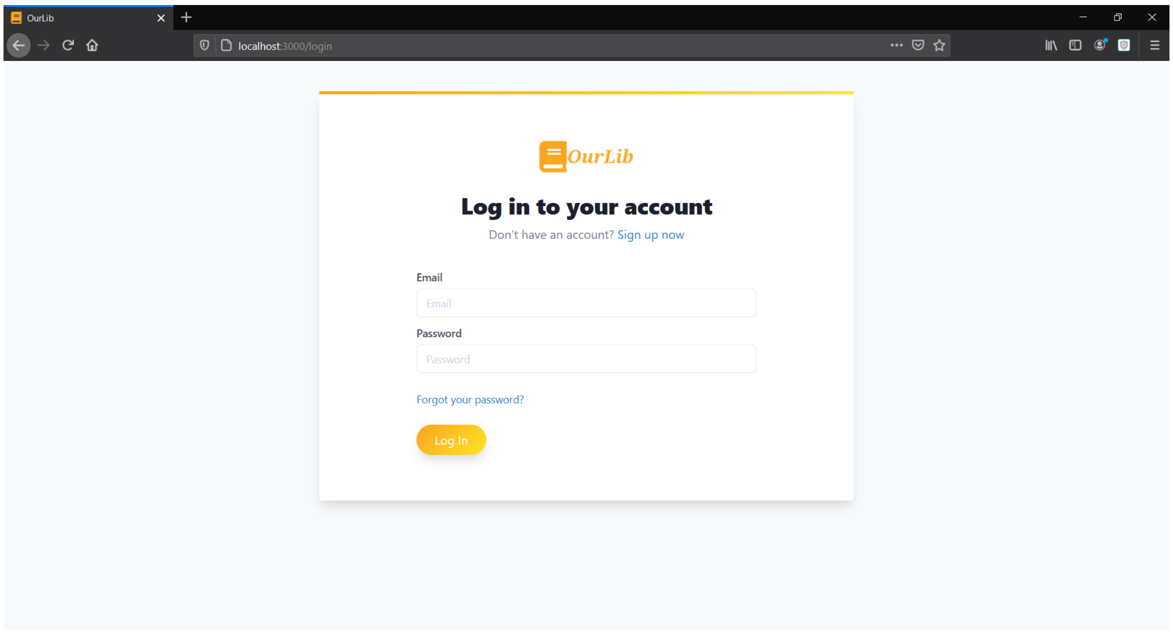
Sign up for an account

Already have an account? Log in now

First Name	Last Name
Date of birth: 20.1.2021	Gender: ○ Male ○ Female ○ Other
Phone number	
Address	City
Email address	
Password	

Sign Up

Figure 2: Sign up page



OurLib

localhost:3000/login

OurLib

Log in to your account

Don't have an account? Sign up now

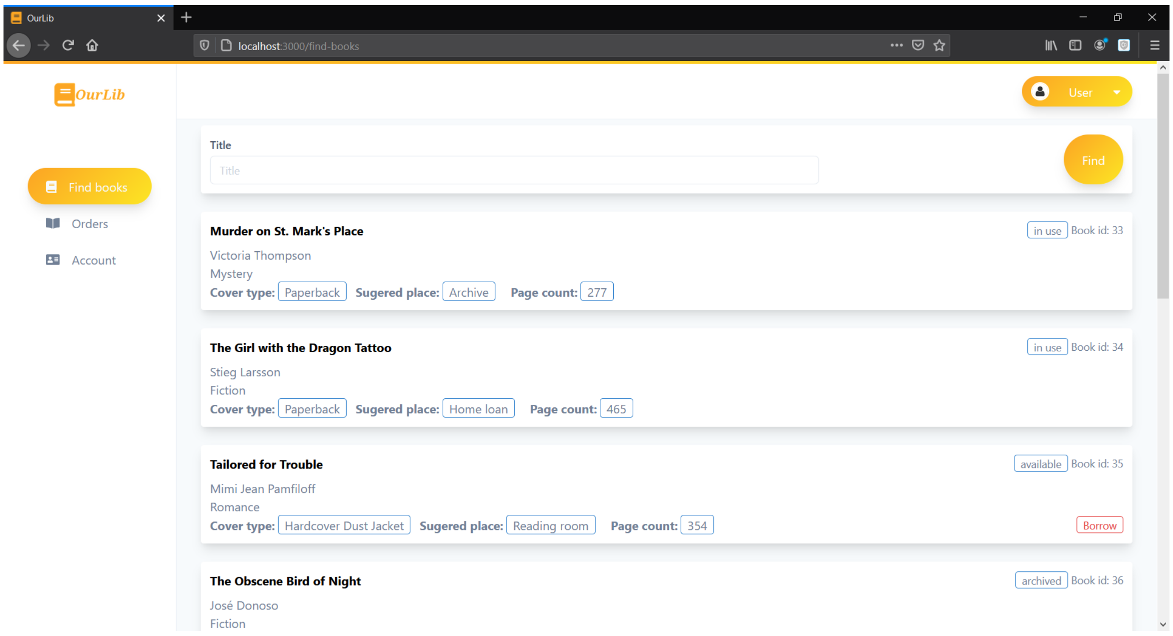
Email

Password

Forgot your password?

Log In

Figure 3: Log in page



Title

Title

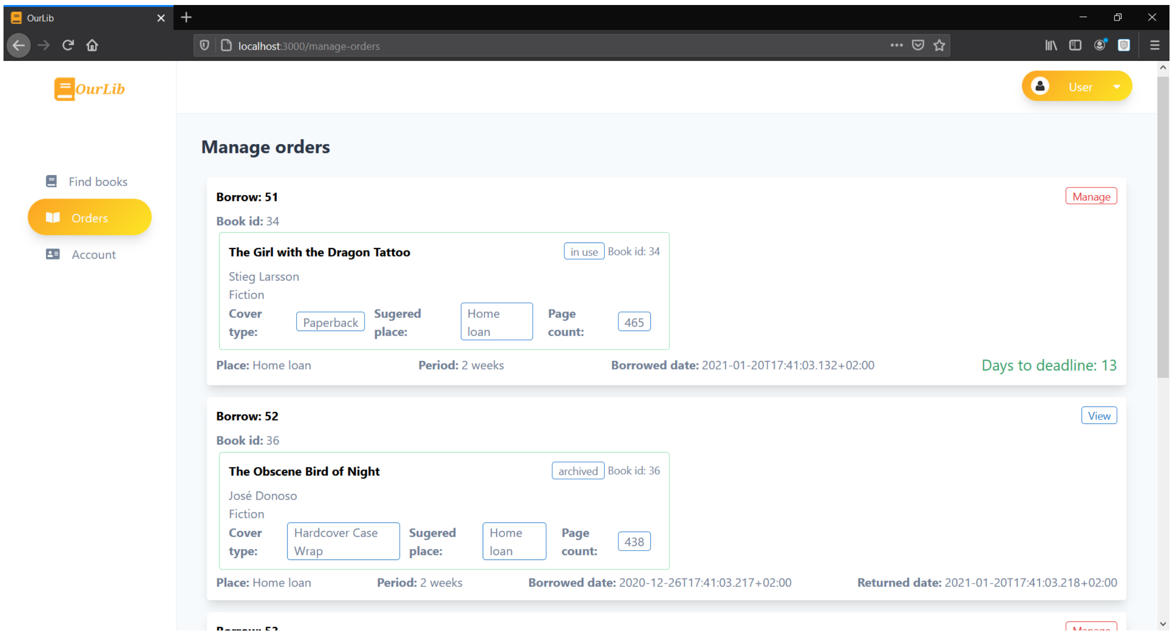
Murder on St. Mark's Place in use Book id: 33
Victoria Thompson
Mystery
Cover type: Paperback **Sugered place:** Archive **Page count:** 277

The Girl with the Dragon Tattoo in use Book id: 34
Stieg Larsson
Fiction
Cover type: Paperback **Sugered place:** Home loan **Page count:** 465

Tailored for Trouble available Book id: 35
Mimi Jean Pamfiloff
Romance
Cover type: Hardcover Dust Jacket **Sugered place:** Reading room **Page count:** 354 Borrow

The Obscene Bird of Night archived Book id: 36
José Donoso
Fiction

Figure 4: Find books from reader account



Manage orders

Borrow: 51 Manage
Book id: 34

The Girl with the Dragon Tattoo in use Book id: 34
Stieg Larsson
Fiction
Cover type: Paperback **Sugered place:** Home loan **Page count:** 465

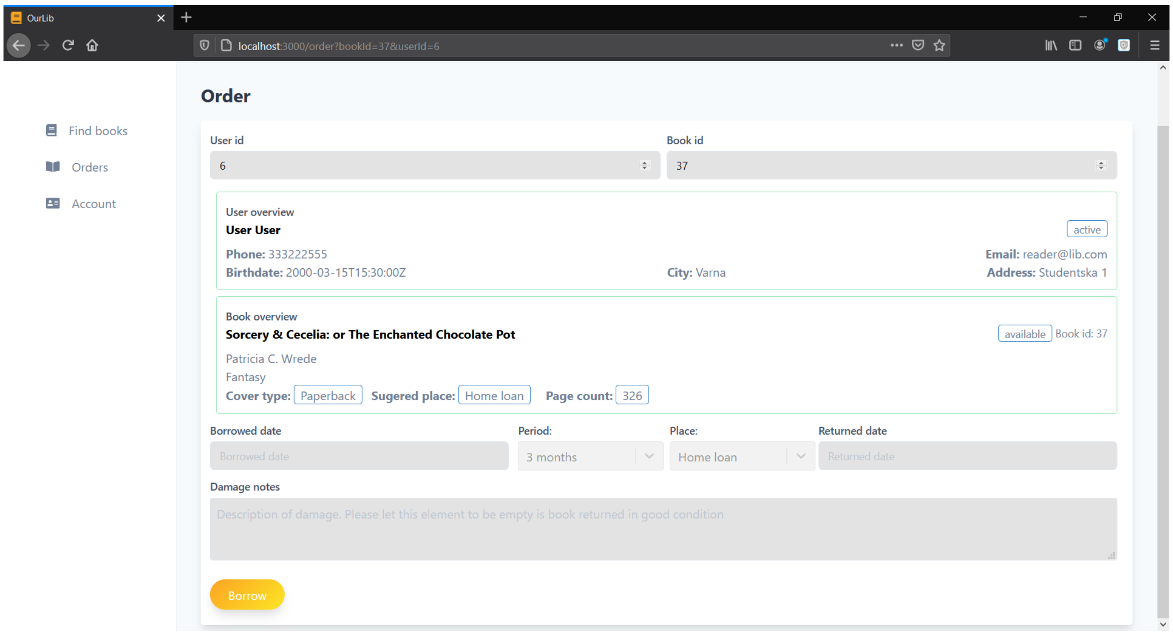
Place: Home loan **Period:** 2 weeks **Borrowed date:** 2021-01-20T17:41:03.132+02:00 Days to deadline: 13

Borrow: 52 View
Book id: 36

The Obscene Bird of Night archived Book id: 36
José Donoso
Fiction
Cover type: Hardcover Case Wrap **Sugered place:** Home loan **Page count:** 438

Place: Home loan **Period:** 2 weeks **Borrowed date:** 2020-12-26T17:41:03.217+02:00 **Returned date:** 2021-01-20T17:41:03.218+02:00

Figure 5: Manage current reader orders



Order

User id: 6

Book id: 37

User overview
User User active
Phone: 333222555
Email: reader@lib.com
Birthdate: 2000-03-15T15:30:00Z
City: Varna
Address: Studentska 1

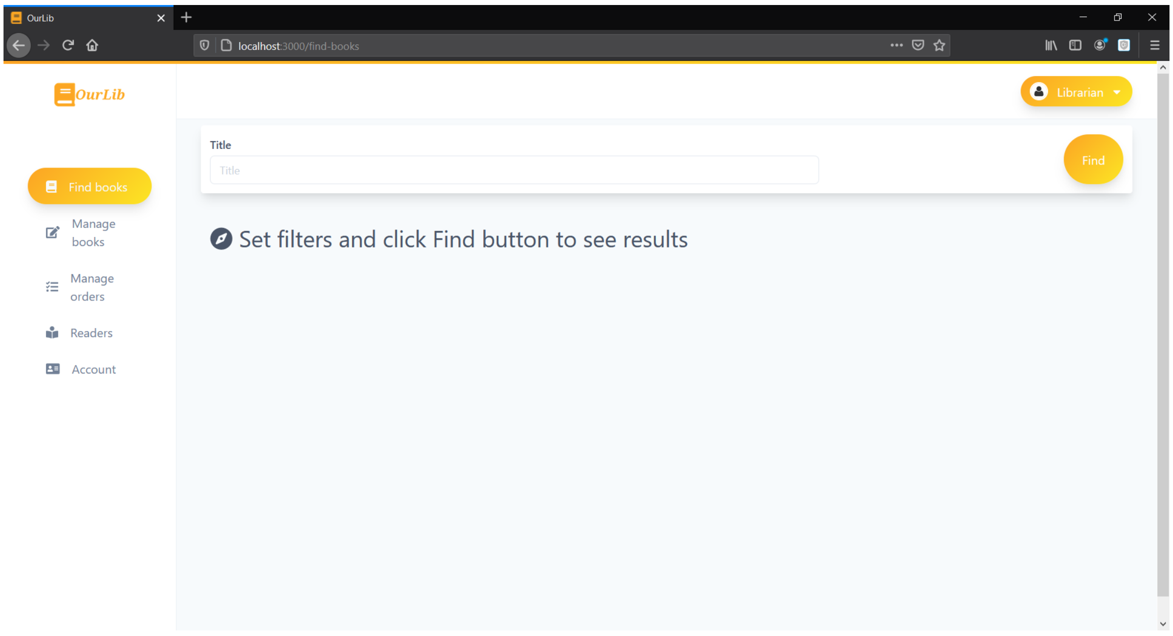
Book overview
Sorcery & Cecelia: or The Enchanted Chocolate Pot available Book id: 37
Patricia C. Wrede
Fantasy
Cover type: Paperback **Sugered place:** Home loan **Page count:** 326

Borrowed date: Borrowed date
Period: 3 months
Place: Home loan
Returned date: Returned date

Damage notes: Description of damage. Please let this element to be empty is book returned in good condition

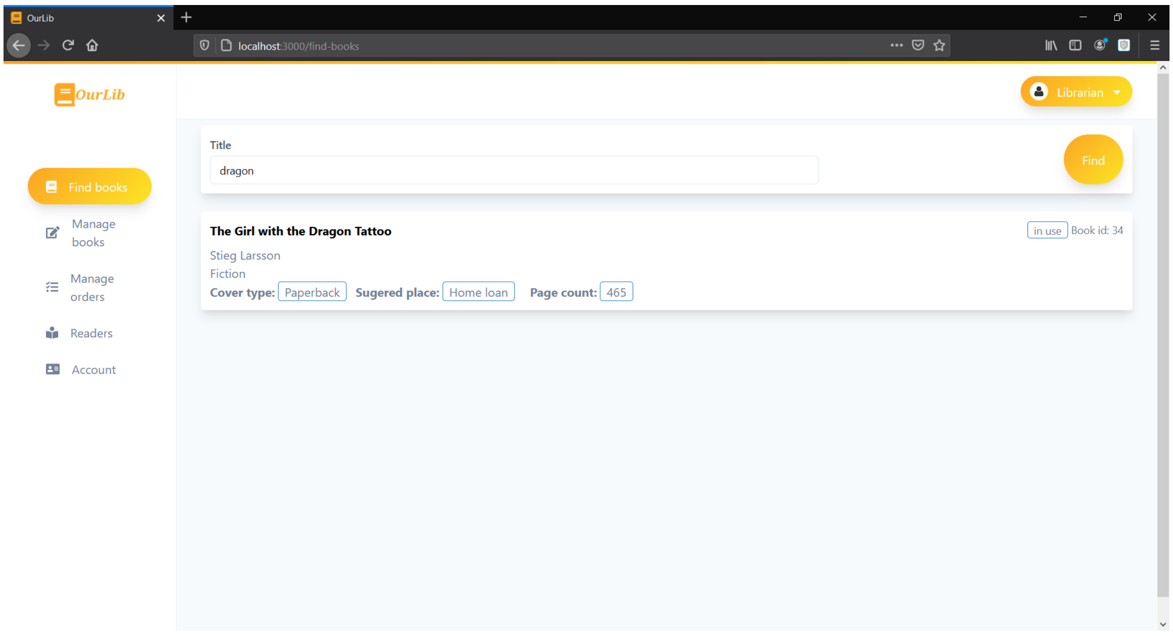
Borrow

Figure 6: Borrow available book by reader



Set filters and click Find button to see results

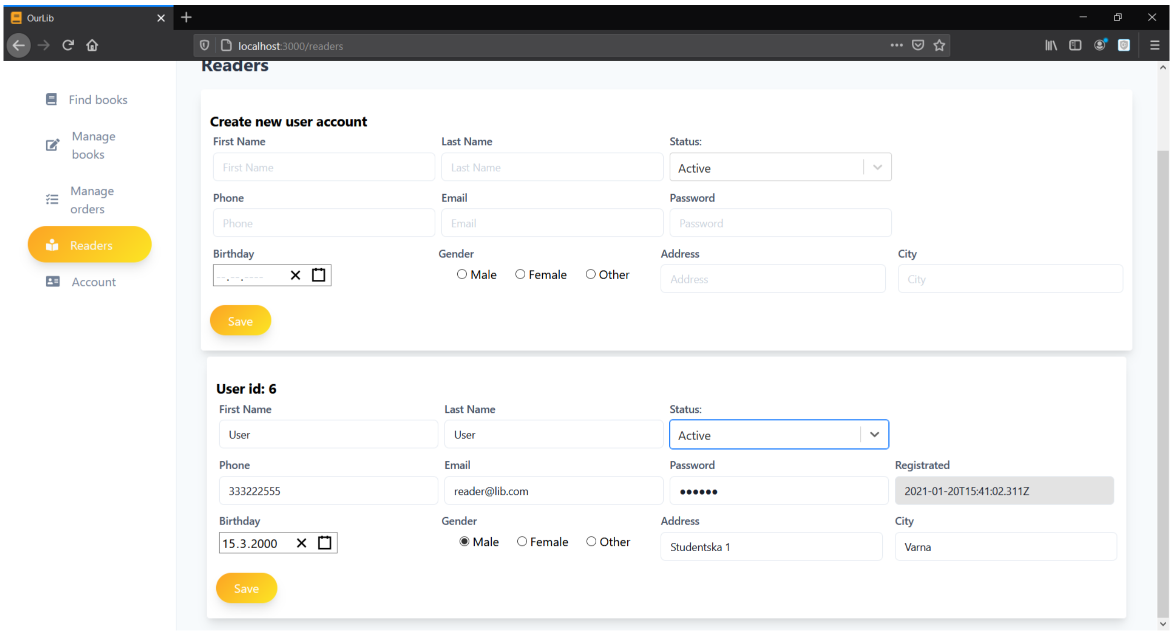
Figure 7: Find book page before find button was pressed



Title: dragon

The Girl with the Dragon Tattoo
in use Book id: 34
Stieg Larsson
Fiction
Cover type: Paperback **Sugered place:** Home loan **Page count:** 465

Figure 8: Searching phrase in search bar



Readers

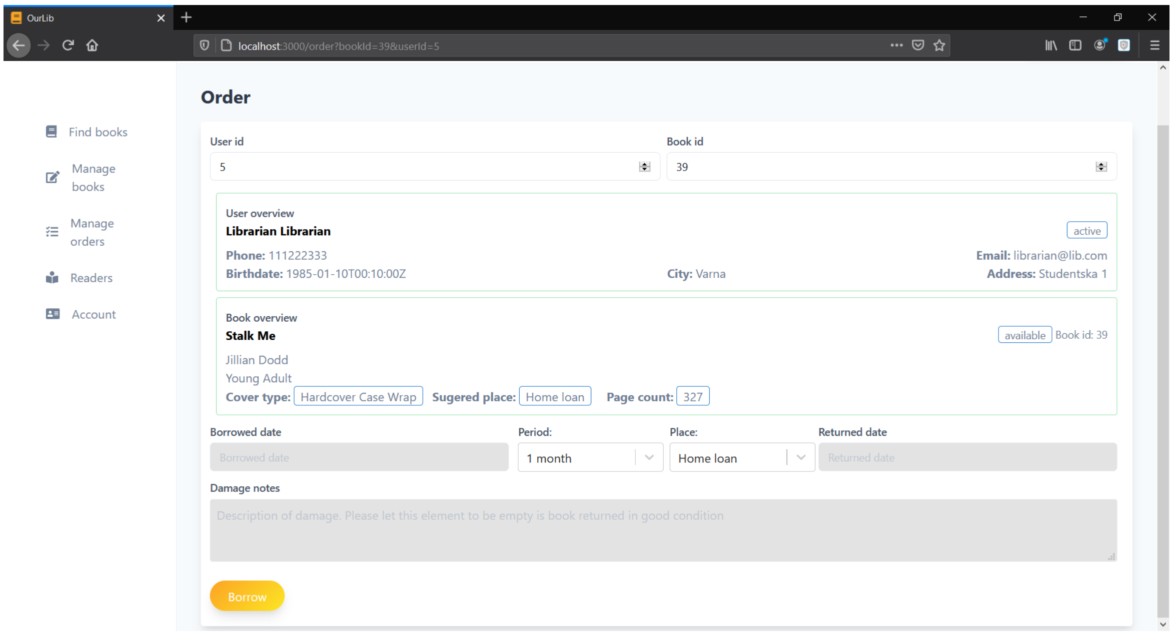
Create new user account

First Name: First Name	Last Name: Last Name	Status: Active	
Phone: Phone	Email: Email	Password: Password	
Birthday: --.--.----	Gender: Male / Female / Other	Address: Address	City: City

User id: 6

First Name: User	Last Name: User	Status: Active	
Phone: 333222555	Email: reader@lib.com	Password: ••••••	Registrated: 2021-01-20T15:41:02.311Z
Birthday: 15.3.2000	Gender: [✓ Male] Female Other	Address: Studentska 1	City: Varna

Figure 9: Manage all readers available in system


OurLib

localhost:3000/order?bookId=39&userId=5

Find books

Manage books

Manage orders

Readers

Account

Order

User id: 5

Book id: 39

User overview
Librarian Librarian active

Phone: 111222333

Email: librarian@lib.com

Birthdate: 1985-01-10T00:10:00Z

City: Varna

Address: Studentska 1

Book overview
Stalk Me available Book id: 39

Jillian Dodd

Young Adult

Cover type: Hardcover Case Wrap **Sugered place:** Home loan **Page count:** 327

Borrowed date: Borrowed date

Period: 1 month

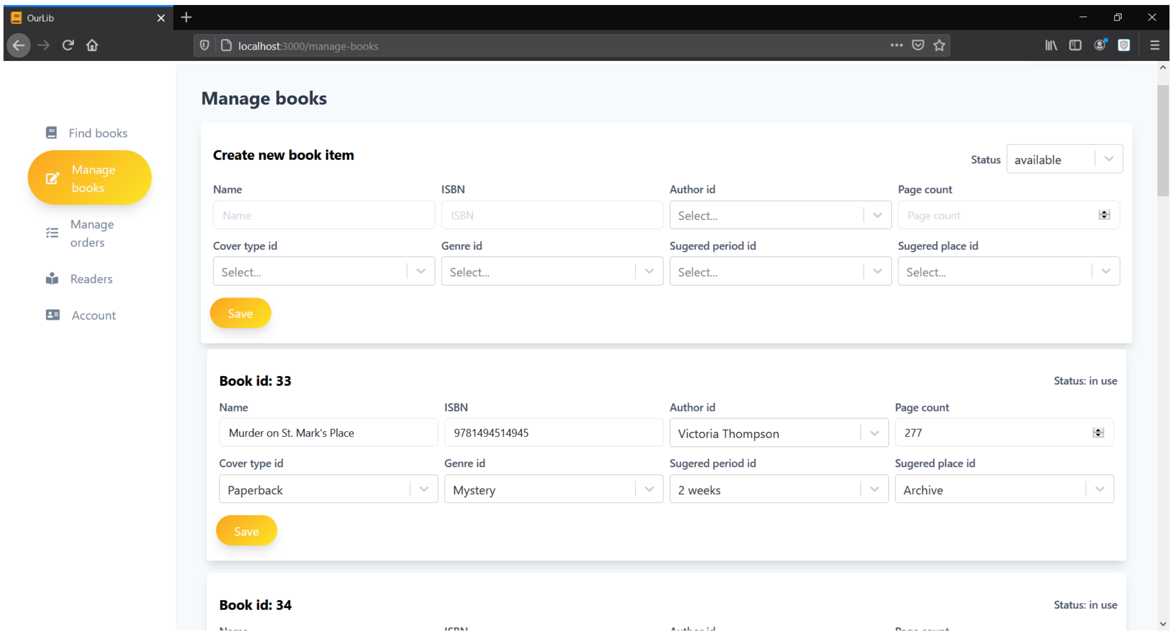
Place: Home loan

Returned date: Returned date

Damage notes: Description of damage. Please let this element to be empty is book returned in good condition

Borrow

Figure 10: Borrow any available book for any reader


OurLib

localhost:3000/manage-books

Find books

Manage books

Manage orders

Readers

Account

Manage books

Create new book item

Status: available

Name	ISBN	Author id	Page count
Name	ISBN	Select...	Page count

Cover type id	Genre id	Sugered period id	Sugered place id
Select...	Select...	Select...	Select...

Save

Book id: 33 Status: in use

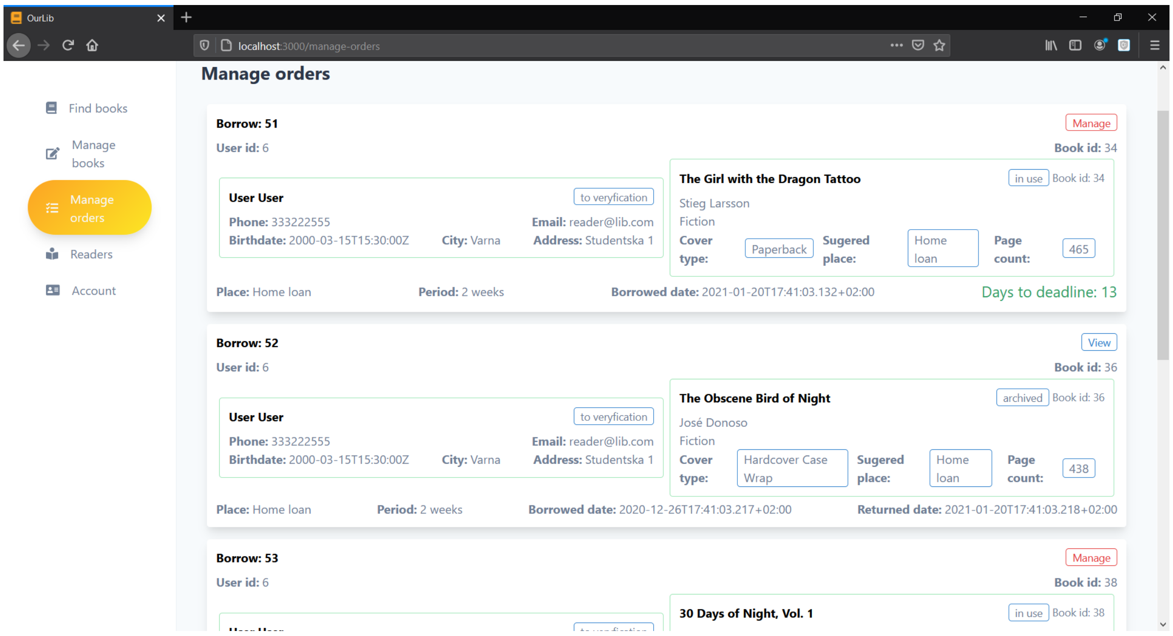
Name	ISBN	Author id	Page count
Murder on St. Mark's Place	9781494514945	Victoria Thompson	277

Cover type id	Genre id	Sugered period id	Sugered place id
Paperback	Mystery	2 weeks	Archive

Save

Book id: 34 Status: in use

Figure 11: Manage all books in system



Manage orders

Borrow: 51 Manage
User id: 6 **Book id:** 34

User User to veryfication
Phone: 333222555 **Email:** reader@lib.com
Birthdate: 2000-03-15T15:30:00Z **City:** Varna **Address:** Studentska 1

The Girl with the Dragon Tattoo in use Book id: 34
Stieg Larsson
Fiction
Cover type: Paperback **Sugered place:** Home loan **Page count:** 465

Place: Home loan **Period:** 2 weeks **Borrowed date:** 2021-01-20T17:41:03.132+02:00 Days to deadline: 13

Borrow: 52 View
User id: 6 **Book id:** 36

User User to veryfication
Phone: 333222555 **Email:** reader@lib.com
Birthdate: 2000-03-15T15:30:00Z **City:** Varna **Address:** Studentska 1

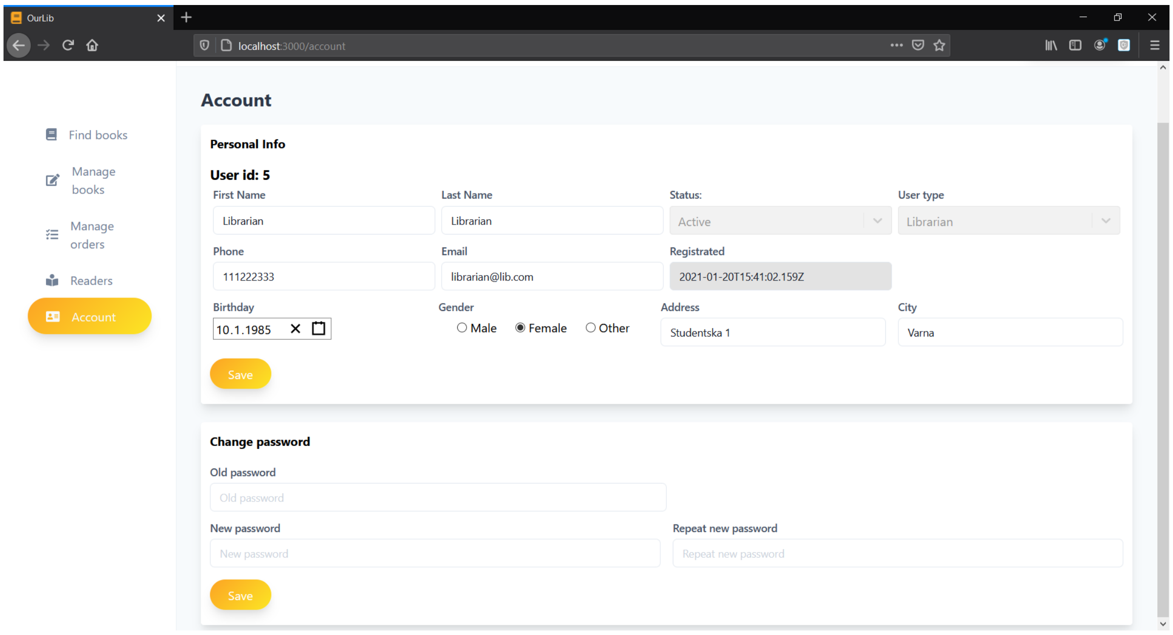
The Obscene Bird of Night archived Book id: 36
José Donoso
Fiction
Cover type: Hardcover Case Wrap **Sugered place:** Home loan **Page count:** 438

Place: Home loan **Period:** 2 weeks **Borrowed date:** 2020-12-26T17:41:03.217+02:00 **Returned date:** 2021-01-20T17:41:03.218+02:00

Borrow: 53 Manage
User id: 6 **Book id:** 38

30 Days of Night, Vol. 1 in use Book id: 38

Figure 12: Manage all orders in system



Account

Personal Info

User id: 5

First Name	Last Name	Status:	User type
Librarian	Librarian	Active	Librarian
Phone	Email	Registrated	
111222333	librarian@lib.com	2021-01-20T15:41:02.159Z	
Birthday	Gender	Address	City
10.1.1985	Male [✓ Female] Other	Studentska 1	Varna

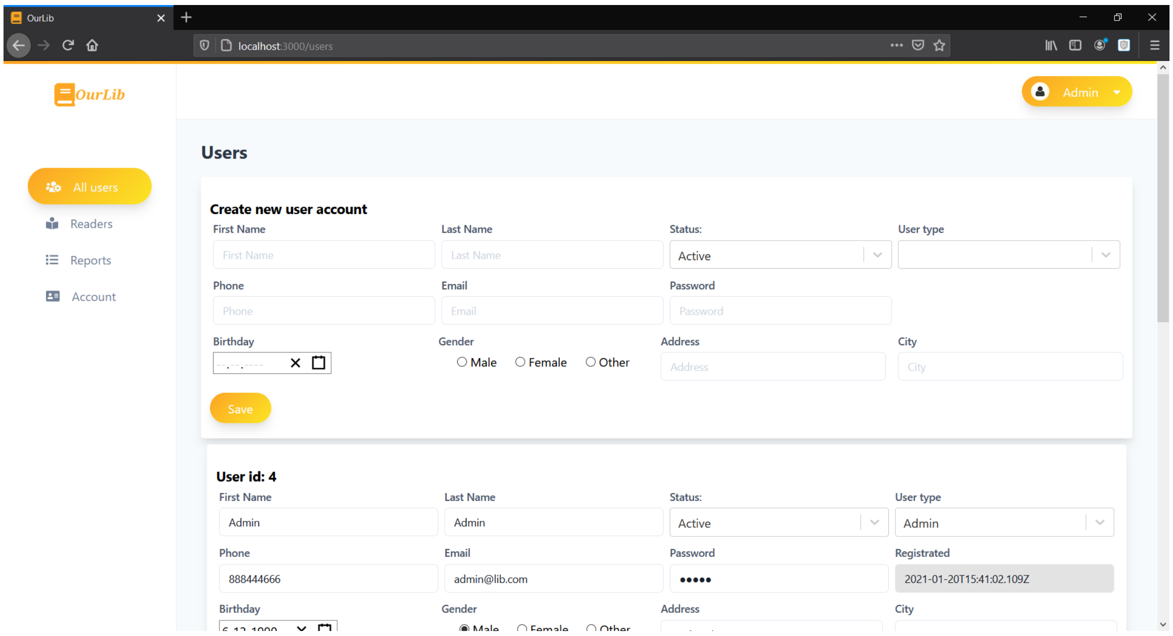
Save

Change password

Old password: Old password
New password: New password
Repeat new password: Repeat new password

Save

Figure 13: Change credentials of current user



Users

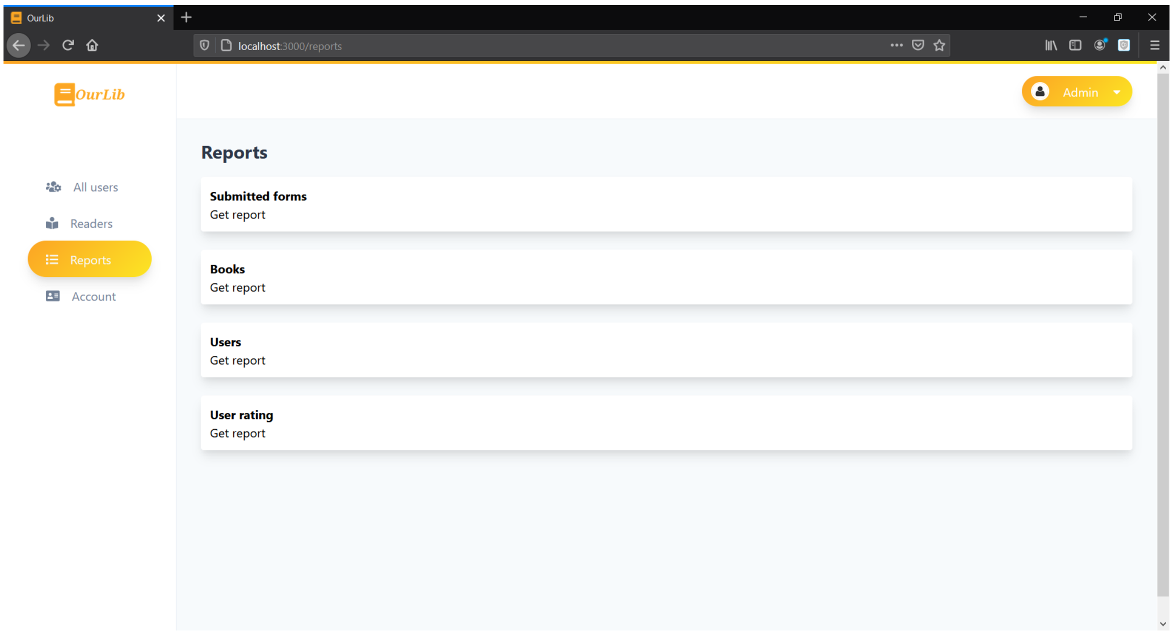
Create new user account

First Name	Last Name	Status:	User type
First Name	Last Name	Active	
Phone	Email	Password	
Phone	Email	Password	
Birthday	Gender	Address	City
--.--.----	○ Male ○ Female ○ Other	Address	City

User id: 4

First Name	Last Name	Status:	User type
Admin	Admin	Active	Admin
Phone	Email	Password	Registrated
888444666	admin@lib.com	●●●●●	2021-01-20T15:41:02.109Z
Birthday	Gender	Address	City
6.12.1990	[✓ Male] ○ Female ○ Other		

Figure 14: Manage all users in system



Reports

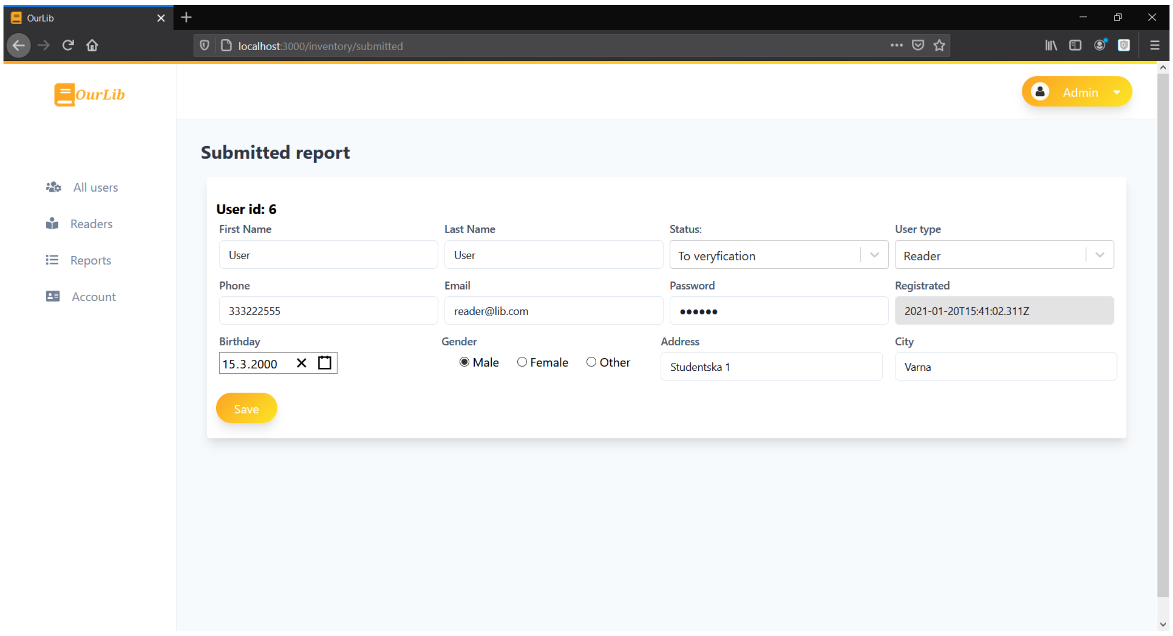
Submitted forms
Get report

Books
Get report

Users
Get report

User rating
Get report

Figure 15: Reports main navigation page


OurLib

localhost:3000/inventory/submitted

Admin

All users
Readers
Reports
Account

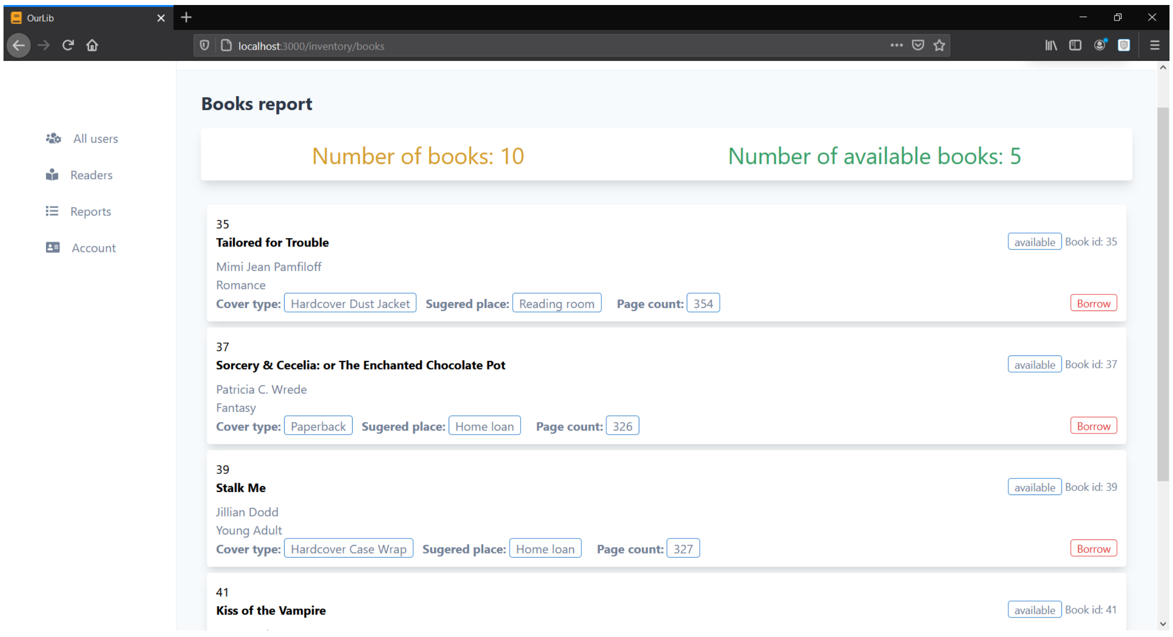
Submitted report

User id: 6

First Name	Last Name	Status:	User type
User	User	To veryfication	Reader
Phone	**Email**	**Password**	**Registrated**
333222555	reader@lib.com	••••••	2021-01-20T15:41:02.311Z
Birthday	**Gender**	**Address**	**City**
15.3.2000	[● Male] ○ Female ○ Other	Studentska 1	Varna

Save

Figure 16: Manage new submitted readers


OurLib

localhost:3000/inventory/books

All users
Readers
Reports
Account

Books report

Number of books: 10 Number of available books: 5

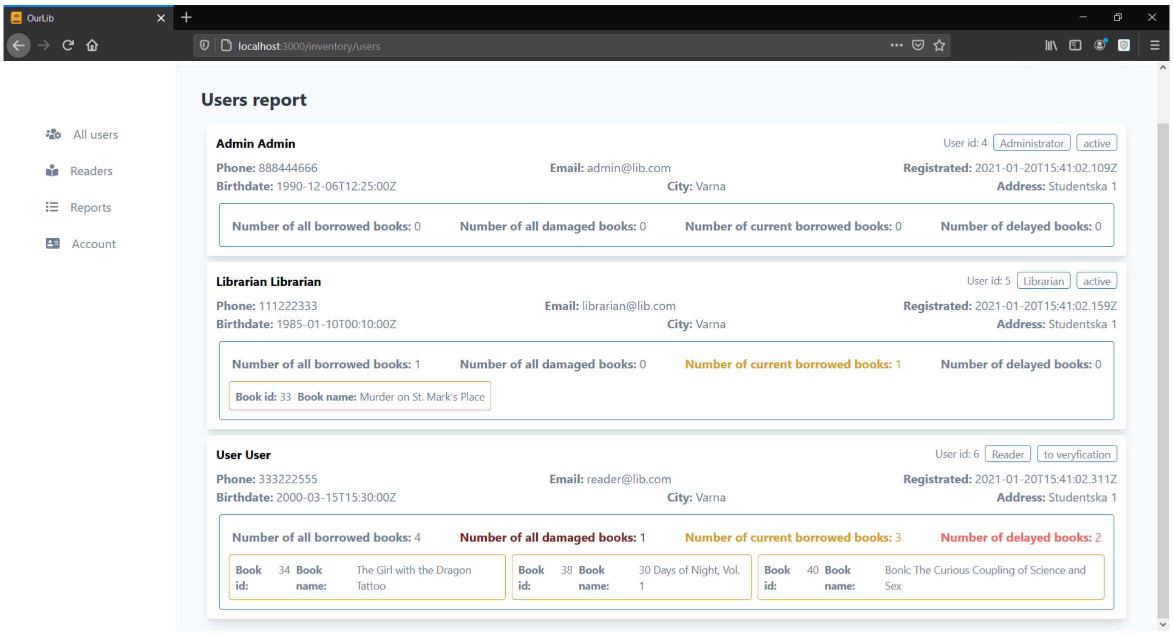
35
Tailored for Trouble
Mimi Jean Pamfiloff
Romance
Cover type: Hardcover Dust Jacket **Sugered place:** Reading room **Page count:** 354
available Book id: 35
Borrow

37
Sorcery & Cecelia: or The Enchanted Chocolate Pot
Patricia C. Wrede
Fantasy
Cover type: Paperback **Sugered place:** Home loan **Page count:** 326
available Book id: 37
Borrow

39
Stalk Me
Jillian Dodd
Young Adult
Cover type: Hardcover Case Wrap **Sugered place:** Home loan **Page count:** 327
available Book id: 39
Borrow

41
Kiss of the Vampire
available Book id: 41

Figure 17: Books in system report



Users report

Admin Admin User id: 4 Administrator active

Phone: 888444666 **Email:** admin@lib.com **Registrated:** 2021-01-20T15:41:02.109Z
Birthdate: 1990-12-06T12:25:00Z **City:** Varna **Address:** Studentska 1

Number of all borrowed books: 0 **Number of all damaged books:** 0 **Number of current borrowed books:** 0 **Number of delayed books:** 0

Librarian Librarian User id: 5 Librarian active

Phone: 111222333 **Email:** librarian@lib.com **Registrated:** 2021-01-20T15:41:02.159Z
Birthdate: 1985-01-10T00:10:00Z **City:** Varna **Address:** Studentska 1

Number of all borrowed books: 1 **Number of all damaged books:** 0 **Number of current borrowed books:** 1 **Number of delayed books:** 0

Book id: 33 **Book name:** Murder on St. Mark's Place

User User User id: 6 Reader to veryfication

Phone: 333222555 **Email:** reader@lib.com **Registrated:** 2021-01-20T15:41:02.311Z
Birthdate: 2000-03-15T15:30:00Z **City:** Varna **Address:** Studentska 1

Number of all borrowed books: 4 **Number of all damaged books:** 1 **Number of current borrowed books:** 3 **Number of delayed books:** 2

Book id: 34 **Book name:** The Girl with the Dragon Tattoo

Book id: 38 **Book name:** 30 Days of Night, Vol. 1

Book id: 40 **Book name:** Bonk: The Curious Coupling of Science and Sex

Figure 18: Users in system report

5 Developer version deployment

To deploy development version of OurLib system a few steps are needed.

5.1 Database

Initialize database with following steps:

- MySQL server should be installed on your computer.
- Table *library_db* should be set up.
- New database user account with credentials login: *api* and password: *api* should be created.

5.2 Backend

Initialize backend with following steps:

- Maven and JDK tools should be installed on your computer.

5.3 GUI

Initialize backend with following steps:

- NodeJS should be installed on your computer.

5.4 Run project

To run project on computer with Windows OS script *run.bat* can be used.

6 Appendix A

```yaml
swagger: '2.0'
info:
  description: Library API
  version: 1.0.3
  title: Library Varna
  termsOfService: http://swagger.io/terms/
  contact:
    email: kornelstefanczyk@wp.pl
  license:
    name: Apache 2.0
    url: "http://www.apache.org/licenses/LICENSE-2.0.html"
tags:
  - name: user
    description: Operations about user
  - name: book
    description: Everything about your Books
  - name: library
    description: Everything about Library
  - name: dataset
    description: Print datasets stored in database
host: localhost:8080
# basePath: /
schemes:
- http
paths:
  /user:
    post:
      tags:
      - user
      summary: Create user
      description: This can only be done by the logged in user.
      operationId: createUser
      consumes:
      - application/json
      parameters:
      - in: body
        name: body
        description: Created user object
        required: true
        schema:
          $ref: '#/definitions/User'
      - name: token
        in: query
        required: false
        type: string
      responses:
        200:
          description: successful operation
        400:
          description: "Bad request"
        401:
          description: Unauthorized, no access
        403:
          description: "Forbidden"
      security:
      - bearerAuth: []
    get:
      tags:
        - user
      summary: Finds user items by params
      description: Multiple status values can be provided with comma separated strings
      operationId: findUsersByStatus
      produces:
        - application/json
      parameters:
        - name: token
          in: query
          required: false
          type: string
        - name: firstName
          in: query
          description: User first name
          required: false
          type: string
        - name: lastName
          in: query
          description: User last name
          required: false
          type: string
        - name: email
          in: query
          description: User last name
          required: false
          type: string
        - name: userTypeId
          in: query
          description: User type id
          required: false
          type: integer
          format: int64
        - name: status
          in: query
          description: User status
          required: false
          type: string
          enum:
            - active
            - suspended
            - inactive
            - to veryfication
          default: available
        - name: gender
          in: query
          description: User gender, not implemented
          required: false
          type: string
      responses:
        200:
          description: successful operation
          schema:
            type: array
            items:
              $ref: "#/definitions/User"
        400:
          description: Invalid status value
      security:
        - bearerAuth: []

  /user/login:
    post:
      tags:
      - user
      summary: Logs user into the system
      operationId: loginUser
      consumes:
      - application/json
      produces:
      - application/json
      parameters:
      - in: body
        name: body
        description: Created user object
        required: true
        schema:
          $ref: "#/definitions/UserLoginBody"
      security:
      - bearerAuth: []
      responses:
        200:
          description: successful operation
          schema:
            $ref: "#/definitions/User"
          headers:
            token:
              type: string
              description: Personal token generated for user
            X-Rate-Limit:
              type: integer
              format: int32
              description: calls per hour allowed by the user
            X-Expires-After:
              type: string
              format: date-time
              description: date in UTC when token expires
        400:
          description: Invalid username/password supplied
  /user/logout:
    get:
      tags:
      - user
      summary: Logs out current logged in user session
      operationId: logoutUser
      parameters:
      - name: token
        in: query
        required: false
        type: string
      responses:
        200:
          description: successful operation
        401:
          description: Unauthorized, no access
        default:
          description: successful operation
      security:
      - bearerAuth: []
  "/user/{id}":
    get:
      tags:
      - user
      summary: Get user by id
      operationId: getUserById
      produces:
      - application/json
      parameters:
      - name: id
        in: path
        type: integer
        format: int64
        description: The id that needs to be fetched.
        required: true
      - name: token
        in: query
        required: true
        type: string
      responses:
        200:
          description: successful operation
          schema:
            $ref: '#/definitions/User'
        400:
          description: Invalid id supplied
        401:
          description: Unauthorized, no access
        404:
          description: User not found
      security:
      - bearerAuth: []
    put:
      tags:
      - user
      summary: Updated user
      description: This can only be done by the logged in user.
      operationId: updateUserById
      consumes:
      - application/json
      parameters:
      - name: id
        in: path
        description: Id that need to be updated
        required: true
        type: integer
        format: int64
      - in: body
        name: body
        description: Updated user object
        required: true
```

```
          schema:
            $ref: '#/definitions/User'
        - name: token
          in: query
          required: true
          type: string
        responses:
          200:
            description: successful operation
          400:
            description: Invalid Id supplied
          401:
            description: Unauthorized, no access
          404:
            description: User not found
        security:
        - bearerAuth: []
      delete:
        tags:
        - user
        summary: Delete user
        description: This can only be done by the logged in user.
        operationId: deleteUserById
        parameters:
        - name: id
          in: path
          description: The Id that needs to be deleted
          required: true
          type: integer
          format: int64
        - name: token
          in: query
          required: true
          type: string
        responses:
          200:
            description: successful operation
          400:
            description: Invalid Id supplied
          401:
            description: Unauthorized, no access
          404:
            description: User not found
        security:
        - bearerAuth: []

    "/user/{id}/stats":
      get:
        tags:
        - user
        summary: Get user statistics by user name
        operationId: getStatsById
        produces:
        - application/json
        parameters:
        - name: id
          in: path
          required: true
          type: integer
          format: int64
        - name: token
          in: query
          required: true
          type: string
        responses:
          200:
            description: successful operation
            schema:
              $ref: '#/definitions/UserStatus'
          400:
            description: Invalid Id supplied
          401:
            description: Unauthorized, no access
          404:
            description: User not found
        security:
          - bearerAuth: []
  /book:
    post:
      tags:
      - book
      summary: Add a new book to the library
      operationId: addBook
      consumes:
      - application/json
      parameters:
      - in: body
        name: body
        description: order placed for purchasing the book
        required: true
        schema:
          $ref: '#/definitions/Book'
      - name: token
        in: query
        required: true
        type: string
      responses:
        200:
          description: successful operation
        400:
          description: "Bad request"
        401:
          description: Unauthorized, no access
      security:
        - bearerAuth: []
    get:
      tags:
        - book
      summary: Get all books
      description: Can be done only by librarian
      operationId: getAllBooks
      produces:
      - application/json
      parameters:
```

```yaml
      - name: token
        in: query
        required: true
        type: string
      responses:
        200:
          description: successful operation
          schema:
            type: array
            items:
              $ref: "#/definitions/Book"
        400:
          description: Invalid status value
      security:
        - bearerAuth: []
  "/book/{id}":
        # ****************************************
    get:
      tags:
      - book
      summary: Get book by book id
      operationId: getBookById
      produces:
      - application/json
      parameters:
        - name: id
          in: path
          description: The id that needs to be fetched.
          required: true
          type: integer
          format: int64
        - name: token
          in: query
          required: false
          type: string
      responses:
        200:
          description: successful operation
          schema:
            $ref: "#/definitions/Book"
        400:
          description: Invalid id supplied
        401:
          description: Unauthorized, no access
        404:
          description: Book not found
      security:
      - bearerAuth: []
    put:
      tags:
      - book
      summary: Updated book
      description: This can only be done by the logged in user.
      operationId: updateBookById
      consumes:
      - application/json
      parameters:
        - name: id
          in: path
          description: id of book that need to be updated
          required: true
          type: integer
          format: int64
        - in: body
          name: body
          description: Updated user object
          required: true
          schema:
            $ref: '#/definitions/Book'
        - name: token
          in: query
          required: true
          type: string
      responses:
        200:
          description: successful operation
        400:
          description: Invalid id supplied
        401:
          description: Unauthorized, no access
        404:
          description: Book not found
      security:
      - bearerAuth: []
    delete:
      tags:
      - book
      summary: Delete book
      description: This can only be done by the logged in user.
      operationId: deleteBookById
      parameters:
        - name: id
          in: path
          description: The id of the book that needs to be deleted
          required: true
          type: integer
          format: int64
        - name: token
          in: query
          required: true
          type: string
      responses:
        200:
          description: successful operation
        400:
          description: Invalid id supplied
        401:
          description: Unauthorized, no access
        404:
          description: Book not found
      security:
      - bearerAuth: []

  /book/findByStatus:
    get:
      tags:
        - book
      summary: Finds Books by status
      description: Multiple status values can be provided with comma separated strings
      operationId: findBookByStatus
      produces:
      - application/json
      parameters:
        - name: status
          in: query
          description: Status values that need to be considered for filter
          required: false
          type: array
          items:
            type: string
            enum:
              - available
              - all
            default: available
        - name: author
          in: query
          description: Author of the book
          required: false
          type: string
        - name: title
          in: query
          description: Title of the book
          required: false
          type: string
        - name: genere
          in: query
          description: Genere of the book
          required: false
          type: string
      responses:
        200:
          description: successful operation
          schema:
            type: array
            items:
              $ref: ”#/definitions/Book”
        400:
          description: Invalid status value
      security:
        - bearerAuth: []
  /dataset/author:
    get:
      tags:
        - dataset
      summary: Get authors dataset
      description: Get authors dataset
      operationId: getDatasetAuthors
      produces:
      - application/json
      parameters:
        - name: id
          in: query
          description: The id of the author. If not entered then all authors are listed
          required: false
          type: integer
          format: int64
      responses:
        200:
          description: successful operation
          schema:
            type: array
            items:
              $ref: "#/definitions/Author"
      security:
        - bearerAuth: []
  /dataset/coverType:
    get:
      tags:
        - dataset
      summary: Get cover type dataset
      description: Get cover type dataset
      operationId: getDatasetCoverType
      produces:
      - application/json
      parameters:
        - name: id
          in: query
          description: The id of the cover type. If not entered then all cover types are listed
          required: false
          type: integer
          format: int64
      responses:
        200:
          description: successful operation
          schema:
            type: array
            items:
              $ref: "#/definitions/CoverType"
      security:
        - bearerAuth: []
  /dataset/bookGenre:
    get:
      tags:
        - dataset
      summary: Get book genre dataset
      description: Get book genre dataset
      operationId: getDatasetBookGenre
      produces:
      - application/json
      parameters:
        - name: id
          in: query
          description: The id of the book genre. If not entered then all book genre are listed
          required: false
          type: integer
          format: int64
      responses:
        200:
          description: successful operation
          schema:
            type: array
            items:
              $ref: "#/definitions/BookGenre"
      security:
        - bearerAuth: []

  /dataset/borrowPeriod:
    get:
      tags:
        - dataset
      summary: Get borrow period dataset
      description: Get borrow period dataset
      operationId: getDatasetBorrowPeriod
      produces:
      - application/json
      parameters:
        - name: id
          in: query
          description: The id of the borrow period. If not entered then
              all borrow period are listed
          required: false
          type: integer
          format: int64
      responses:
        200:
          description: successful operation
          schema:
            type: array
            items:
              $ref: "#/definitions/BorrowPeriod"
      security:
        - bearerAuth: []

  /dataset/borrowPlace:
    get:
      tags:
        - dataset
      summary: Get borrow place dataset
      description: Get borrow place dataset
      operationId: getDatasetBorrowPlace
      produces:
      - application/json
      parameters:
        - name: id
          in: query
          description: The id of the borrow place. If not entered then
              all borrow place are listed
          required: false
          type: integer
          format: int64
      responses:
        200:
          description: successful operation
          schema:
            type: array
            items:
              $ref: "#/definitions/BorrowPlace"
      security:
        - bearerAuth: []

  /library/inventory/books:
    get:
      tags:
      - library
      summary: Returns library books
      description: Returns a stats of the library inventory
      operationId: getLibraryInventoryBooks
      parameters:
      - name: token
        in: query
        required: true
        type: string
      produces:
      - application/json
      responses:
        200:
          description: successful operation
          schema:
            $ref: "#/definitions/LibraryBooksReport"
        401:
          description: Unauthorized, no access
      security:
        - bearerAuth: []
  /library/inventory/users:
    get:
      tags:
      - library
      summary: Returns library users
      description: Returns a stats of the library users
      operationId: getLibraryInventoryUsers
      parameters:
      - name: token
        in: query
        required: true
        type: string
      produces:
      - application/json
      responses:
        200:
          description: successful operation
          schema:
            type: array
            items:
              $ref: "#/definitions/UserStatusReport"
        401:
          description: Unauthorized, no access
      security:
        - bearerAuth: []
  /library/inventory/submitted:
    get:
      tags:
      - library
      summary: Returns library submitted users
      description: Returns a stats of the library submitted users
      operationId: getLibraryInventorySubmittedUsers
      parameters:
      - name: token
        in: query
        required: true
        type: string
      produces:
      - application/json
      responses:
        200:
          description: successful operation
          schema:
            type: array
            items:
              $ref: "#/definitions/SubmitUserReport"
        401:
          description: Unauthorized, no access
      security:
        - bearerAuth: []
  /library/inventory/userrating:
    get:
      tags:
      - library
      summary: Returns library users rating
      description: Returns a stats of the library users rating
      operationId: getLibraryInventoryUsersRating
      parameters:
      - name: token
        in: query
        required: true
        type: string
      produces:
      - application/json
      responses:
        200:
          description: successful operation
          schema:
            type: array
            items:
              $ref: "#/definitions/UsersRatingReport"
        401:
          description: Unauthorized, no access
      security:
        - bearerAuth: []
  /library/order:
    post:
      tags:
      - library
      summary: Place an order for a book rent
      operationId: placeOrder
      consumes:
      - application/json
      produces:
      - application/json
      parameters:
      - in: body
        name: body
        description: order placed for borrow the book
        required: true
        schema:
          $ref: "#/definitions/Borrowed"
      - name: token
        in: query
        required: true
        type: string
      responses:
        200:
          description: successful operation
          schema:
            $ref: "#/definitions/Borrowed"
        401:
          description: Unauthorized, no access
        400:
          description: Invalid Order
      security:
      - bearerAuth: []
    get:
      tags:
        - library
      summary: Get orders
      description: Librarian get all orders, Reader takes only owned
          orders
      operationId: getAllOrders
      produces:
      - application/json
      parameters:
      - name: token
        in: query
        required: true
        type: string
      responses:
        200:
          description: successful operation
          schema:
            type: array
            items:
              $ref: "#/definitions/Borrowed"
        400:
          description: Invalid status value
      security:
        - bearerAuth: []
  "/library/order/{orderId}":
    get:
      tags:
      - library
      summary: Find book order by ID
      operationId: getOrderById
      produces:
      - application/json
      parameters:
      - name: orderId
        in: path
        description: ID of borrow form that needs to be fetched
        required: true
        type: integer
        format: int64
      - name: token
        in: query
        required: true
        type: string
      responses:
        200:
          description: successful operation
          schema:
            $ref: "#/definitions/Borrowed"
        400:
          description: Invalid ID supplied
        401:
          description: Unauthorized, no access
        404:
          description: Order not found
      security:
      - bearerAuth: []
    put:
      tags:
      - library
      summary: Find purchase order by ID
      operationId: putOrderById
      consumes:
      - application/json
      produces:
      - application/json
      parameters:
      - name: orderId
        in: path
        description: ID of pet that needs to be fetched
        required: true
        type: integer
        format: int64
      - in: body
        name: body
        description: form ot the borrowed book that needs to update
        required: true
        schema:
          $ref: "#/definitions/Borrowed"
      - name: token
        in: query
        required: true
        type: string
      responses:
        200:
          description: successful operation
          schema:
            $ref: "#/definitions/Borrowed"
        400:
          description: Invalid ID supplied
        401:
          description: Unauthorized, no access
        404:
          description: Order not found
      security:
      - bearerAuth: []
    delete:
      tags:
      - library
      summary: Delete purchase order by ID
      operationId: deleteOrder
      consumes:
      - application/json
      produces:
      - application/json
      parameters:
        - name: orderId
          in: path
          description: ID of the order that needs to be deleted
          required: true
          type: integer
          format: int64
        - name: token
          in: query
          required: true
          type: string
      responses:
        200:
          description: successful operation
        400:
          description: Invalid ID supplied
        401:
          description: Unauthorized, no access
        404:
          description: Order not found
      security:
      - bearerAuth: []

  /library/notyfication:
    get:
      tags:
      - library
      summary: Returns notyfications of library
      description: Returns notyfications of library
      operationId: getLibraryNotyfications
      produces:
      - application/json
      parameters:
      - name: token
        in: query
        required: true
        type: string
      responses:
        200:
          description: successful operation
        schema:
          type: array
          format: chunked
          items:
            $ref: "#/definitions/NotificationForm"
      401:
        description: Unauthorized, no access
    security:
    - bearerAuth: []


securityDefinitions:
bearerAuth:
  type: apiKey
  name: api_key
  in: header
definitions:
UserLoginBody:
  type: object
  properties:
    email:
      description: The email for login
      type: string
      format: email
    password:
      description: The password for login in clear text
      type: string
User:
  type: object
  properties:
    id:
      type: integer
      format: int64
    userTypeId:
      type: integer
      format: int64
    firstName:
      type: string
    lastName:
      type: string
    email:
      type: string
    password:
      type: string
    phone:
      type: string
    birthdate:
      type: string
      format: date-time
    registrated:
      type: string
      format: date-time
    gender:
      type: string
      enum:
        - male
        - female
```

```
            - other
        address:
          type: string
        city:
          type: string
        status:
          type: string
          description: User Status
          enum:
            - active
            - suspended
            - inactive
            - to veryfication
      xml:
        name: User
    UserType:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
        permissions:
          type: string
      xml:
        name: UserType
    SubmitUserReport:
      allOf:
        - $ref: '#/definitions/User'
        - type: object
          properties:
            userTypeName:
              type: string
    UserStatusReport:
      allOf:
        - $ref: '#/definitions/User'
        - $ref: '#/definitions/UserStatus'
        - type: object
          properties:
            userTypeName:
              type: string
            currentBorrowed:
              type: array
              items:
                type: object
                properties:
                  bookId:
                    type: integer
                    format: int64
                  bookName:
                    type: string

    UsersRatingReport:
      type: object
      properties:
        loyal:
          type: array
          items:
            type: object
            properties:
              userId:
                type: integer
                format: int64
              userName:
                type: string
        disloyal:
          type: array
          items:
            type: object
            properties:
              userId:
                type: integer
                format: int64
              userName:
                type: string

    Author:
      type: object
      properties:
        id:
          type: integer
          format: int64
        firstName:
          type: string
        lastName:
          type: string
      xml:
        name: Author
    CoverType:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
      xml:
        name: CoverType
    Borrowed:
      type: object
      properties:
        id:
          type: integer
          format: int64
        userId:
          type: integer
          format: int64
        bookId:
          type: integer
          format: int64
        borrowedDate:
          type: string
          format: date-time
        returnedDate:
          type: string
          format: date-time
        damageNotes:
          type: string
        placeId:
          type: integer
          format: int64
        periodId:
          type: integer
          format: int64
      xml:
        name: Borrowed
    Book:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
        isbn:
          type: string
        authorId:
          type: integer
          format: int64
        pageCount:
          type: integer
        coverTypeId:
          type: integer
          format: int64
        genreId:
          type: integer
          format: int64
        sugeredPeriodId:
          type: integer
          format: int64
        sugeredPlaceId:
          type: integer
          format: int64
        status:
          type: string
          description: Book Status
          enum:
            - available
            - in use
            - archived
      xml:
        name: Book
    BookReport:
      allOf:
        - $ref: '#/definitions/Book'
        - type: object
          properties:
            authorTypeName:
              type: string
            coverTypeName:
              type: string
            genreName:
              type: string
            sugeredPeriodName:
              type: string
            sugeredPlaceName:
              type: string
            statusName:
              type: string

    BookGenre:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
      xml:
        name: BookGenre
    BorrowPeriod:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
        period:
          type: integer
      xml:
        name: BorrowPeriod
    BorrowPlace:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
      xml:
        name: BorrowPlace
    LibraryBooksReport:
      type: object
      properties:
        numberOfBooks:
          type: integer
        numberOfAvailableBooks:
          type: integer
        books:
          type: array
          items:
            $ref: '#/definitions/BookReport'
      xml:
        name: LibraryBooksReport
    UserStatus:
      type: object
      properties:
        numberOfAllBorrowedBooks:
          type: integer
        numberOfAllDamagedBooks:
          type: integer
        numberOfCurrentBorrowedBooks:
          type: integer
        numberOfDelayedBooks:
          type: integer
      xml:
        name: UserStatus
    NotificationForm:
      type: object
      properties: # also possible to model with oneOf+discriminator
        id:
          type: integer
          description: my event id
        event:
          type: string
          description: my event type
        data:
          type: object # object in case we have a complex structure ,
              otherwise string , integer , ... , as usual
      xml:
        name: UserStatus
    UserToken:
      type: object
      properties:
        id:
          type: integer
          format: int64
        addDatatime:
          type: string
          format: date-time
        tokenName:
          type: string
        userId:
          type: integer
          format: int64
        expireDatatime:
          type: string
          format: date-time

  externalDocs:
    description: Find out more about Swagger
    url: "http://swagger.io"
```