

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	3
1.1	Users	3
1.2	Books	3
1.3	Reports	3
1.4	Event Notifications	3
2	System modules	3
2.1	Database	3
2.2	Backend	4
2.3	GUI	4
3	System design	4
3.1	Database	4
3.2	Backend	5
3.2.1	Business logic and data layer	5
3.2.2	RestAPI	5
3.3	Graphical user interface	5
4	Implementation	5
4.1	Database	5
4.2	Backend	5
4.2.1	Models	5
4.2.2	Services	6
4.2.3	RestAPI	7
4.3	Graphical user interface	9
4.3.1	Connection with backend	9
4.3.2	Sidebar	9
4.3.3	Translations	10
4.3.4	Forms	10
4.3.5	Screenshots of GUI	10
5	Developer version deployment	19
5.1	Database	19
5.2	Backend	19
5.3	GUI	19
5.4	Run project	19
6	Appendix A	19

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.

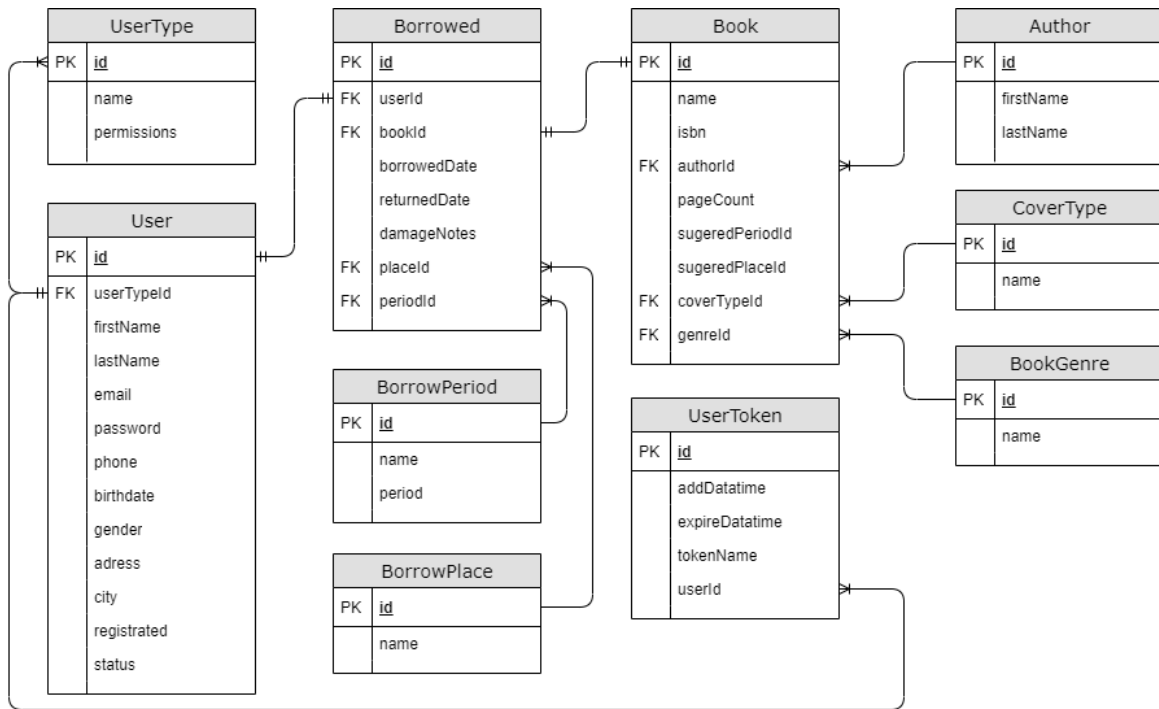


Figure 1: ERD schematic diagram

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 *userId* that reeffers 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.

- *BookGenre* 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

```

931 User :
932   type: object
933   properties:
934     id:
935       type: integer
936       format: int64
937     userId:
938       type: integer
939       format: int64
940     firstName:
```

```

941         type: string
942     lastName:
943         type: string
944     email:
945         type: string
946     password:
947         type: string
948     phone:
949         type: string
950     birthdate:
951         type: string
952         format: date-time
953     registrated:
954         type: string
955         format: date-time
956     gender:
957         type: string
958     enum:
959         - male
960         - female
961         - other
962     address:
963         type: string
964     city:
965         type: string
966     status:
967         type: string
968         description: User Status
969     enum:
970         - active
971         - suspended
972         - inactive
973         - to verification
974     xml:
975         name: User
976     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

```

1  @Entity
2  @Table(name = "book")
3  @JsonRootName("Book")
4  public class Book {
5      @Id
6      @GeneratedValue(strategy=GenerationType.AUTO)
7      @JsonProperty("id")
8      private Long id = null;
9
10     @JsonProperty("name")
11     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

```

1 public interface BookService {
2     public Void initBookValues();
3     public int countBooks();
4     public int countAvailableBooks();
5     public Book createBook(Book book);
6     public List<Book> getAllBooks();
7     public Book deleteBookById(Long id);
8     public Book getBookById(Long id);
9     public String getBookNameById(Long id);
10    public Book updateBookById(Long id, Book body);
11    public Boolean updateBookStatus(Long id, StatusEnum status);
12    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

```

1 @Override
2 public User createUser(User user) {
3     OffsetDateTime registrated = OffsetDateTime.now(ZoneId.of("UTC"));
4     user.setRegistrated(registrated);
5
6     Optional<User> od = userRepository.getUserByEmail(user.getEmail());
7     if(!od.isPresent()) {
8         log.info("Create_new_user:\n"+user.toString());
9         return userRepository.save(user);
10    } else {
11        log.error("Sorry, _user_with_given_email_exist._Email:_"+user.
12            getEmail());
13    }
14    return null;
15 }

```

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

```

25 paths:
26   /user:
27     post:
28       tags:
29         - user
30       summary: Create user
31       description: This can only be done by the logged in user.
32       operationId: createUser
33       consumes:
34         - application/json
35       parameters:
36         - in: body
37           name: body
38           description: Created user object
39           required: true

```

```

40     schema:
41         $ref: '#/definitions/User'
42     - name: token
43       in: query
44       required: false
45       type: string
46     responses:
47       200:
48         description: successful operation
49       400:
50         description: "Bad request"
51       401:
52         description: Unauthorized, no access
53       403:
54         description: "Forbidden"
55     security:
56     - 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

```

1  @ApiOperation(value = "Create_user", nickname = "createUser", notes = "
   This_can_only_be_done_by_the_logged_in_user.", authorizations = {
2  @Authorization(value = "bearerAuth")
3  }, tags={ "user", })
4  @ApiResponse(value = {
5  @ApiResponse(code = 200, message = "successful_operation"),
6  @ApiResponse(code = 400, message = "Bad_request"),
7  @ApiResponse(code = 401, message = "Unauthorized,no_access"),
8  @ApiResponse(code = 403, message = "Forbidden") })
9  @RequestMapping(value = "/user",
10 consumes = { "application/json" },
11 method = RequestMethod.POST)
12 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

```

1  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) {
2  String accept = request.getHeader("Accept");
3  if (accept != null && (accept.contains("application/json") || accept.
   contains("*/*"))) {
4      if(userService.isModifyPermittedForToken(body, token)){
5          body = userService.initNewUserAccordingToPermissions(body,
   token);
6          if(userService.createUser(body) != null){
7              return new ResponseEntity<Void>(HttpStatus.OK);
8          } else {
9              return new ResponseEntity<Void>(HttpStatus.FORBIDDEN);
10         }

```



```

11     } else {
12         return new ResponseEntity<Void>(HttpStatus.UNAUTHORIZED);
13     }
14 } else {
15     return new ResponseEntity<Void>(HttpStatus.BAD_REQUEST);
16 }
17 }

```

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

```

1 token: auth.authState.token
2 }
3 setSaveLoadingBookFullDetailsForm(true);
4 const { data } = credentials.id ? (
5     await fetchContext.authAxios.put(
6         '/book/' + credentials.id + '?' + qs.stringify(queryValues),
7         saveBookItem
8     )
9 ) : (
10     await fetchContext.authAxios.post(
11         '/book?' + qs.stringify(queryValues),
12         saveBookItem
13     )
14 );

```

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

```

1 const navItems = [
2     {
3         label: 'components.sidebar-component.link_label.find_books',
4         path: '/find-books',
5         icon: faBook,
6         allowedRoles: [reader, librarian]
7     },
8     {
9         label: 'components.sidebar-component.link_label.orders',
10        path: '/manage-orders',
11        icon: faBookOpen,
12        allowedRoles: [reader]
13    },
14    {
15        label: 'components.sidebar-component.link_label.manage_books',
16        path: '/manage-books',

```

```

17     icon: faEdit ,
18     allowedRoles: [ librarian ]
19   },
20   {
21     label: 'components.sidebar_component.link_label.manage_orders ',
22     path: '/manage-orders ',
23     icon: faTasks ,
24     allowedRoles: [ librarian ]
25   },
26   {
27     label: 'components.sidebar_component.link_label.all_users ',
28     path: '/users ',
29     icon: faUsersCog ,
30     allowedRoles: [ admin ]
31   },
32   {
33     label: 'components.sidebar_component.link_label.readers ',
34     path: '/readers ',
35     icon: faBookReader ,
36     allowedRoles: [ admin, librarian ]
37   },
38   {
39     label: 'components.sidebar_component.link_label.reports ',
40     path: '/reports ',
41     icon: faList ,
42     allowedRoles: [ admin ]
43   },
44   {
45     label: 'components.sidebar_component.link_label.account ',
46     path: '/account ',
47     icon: faAddressCard ,
48     allowedRoles: [ reader , librarian , admin ]
49   }
50 ];

```

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

Sign up for an account

Already have an account? [Log in now](#)

First Name

Last Name

Date of birth

Gender ☐ Male ☐ Female ☐ Other

Phone number

Address

City

Email address

Password

[Sign Up](#)

Figure 2: Sign up page

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

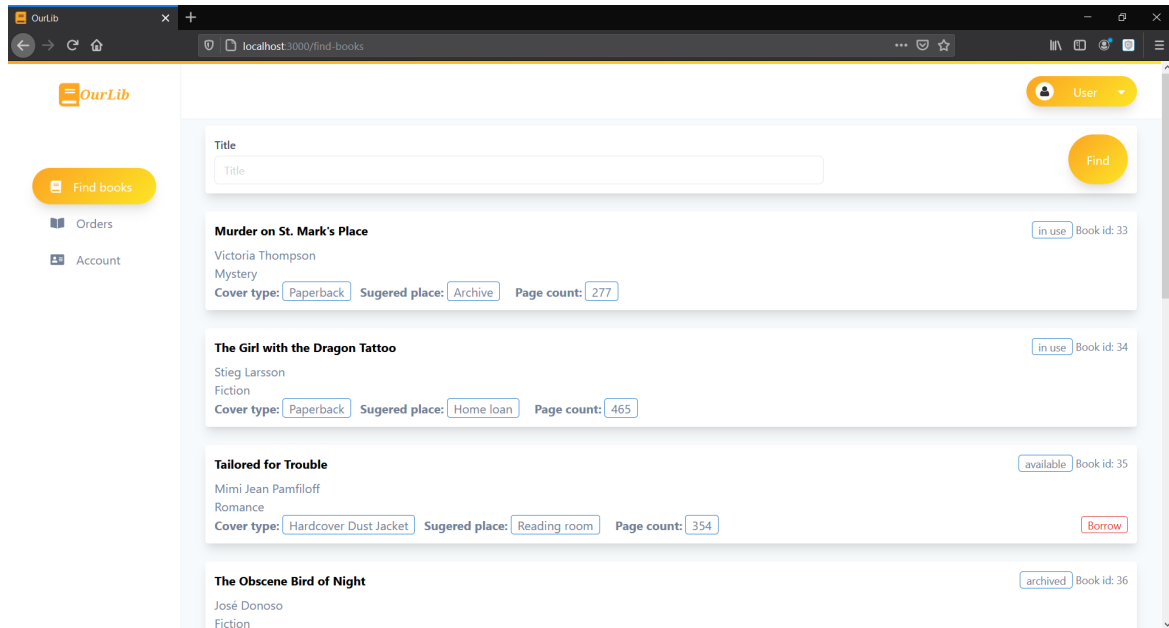


Figure 4: Find books from reader account

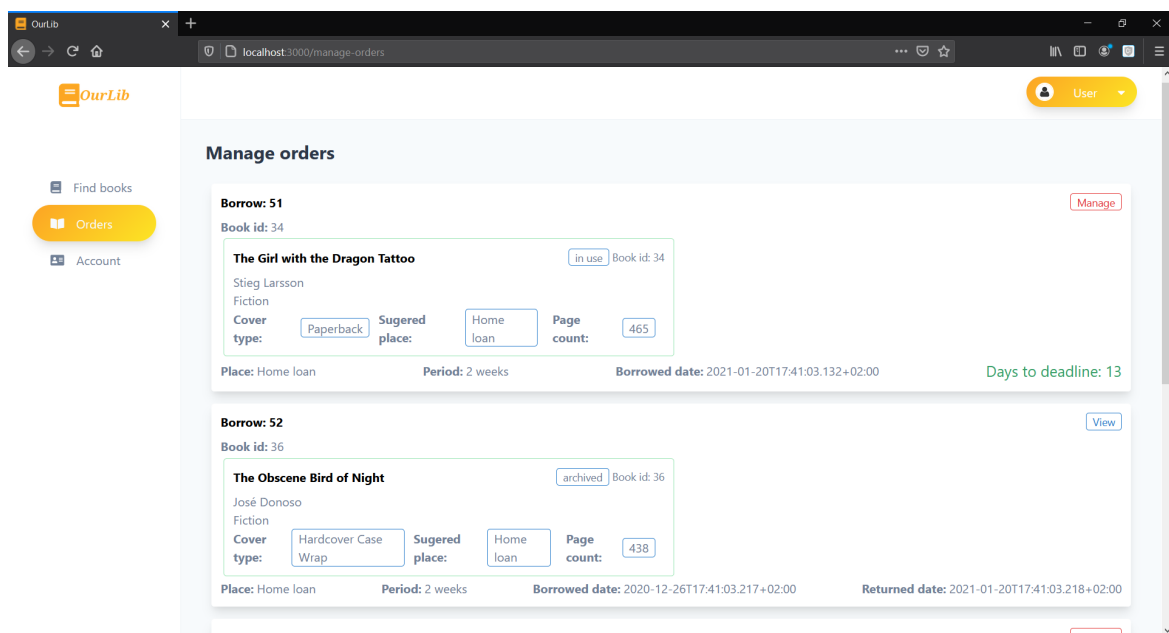


Figure 5: Manage current reader orders

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 Suggested 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

OurLib Librarian

Find books Find

Title

Set filters and click Find button to see results

Manage books
 Manage orders
 Readers
 Account

Figure 7: Find book page before find button was pressed

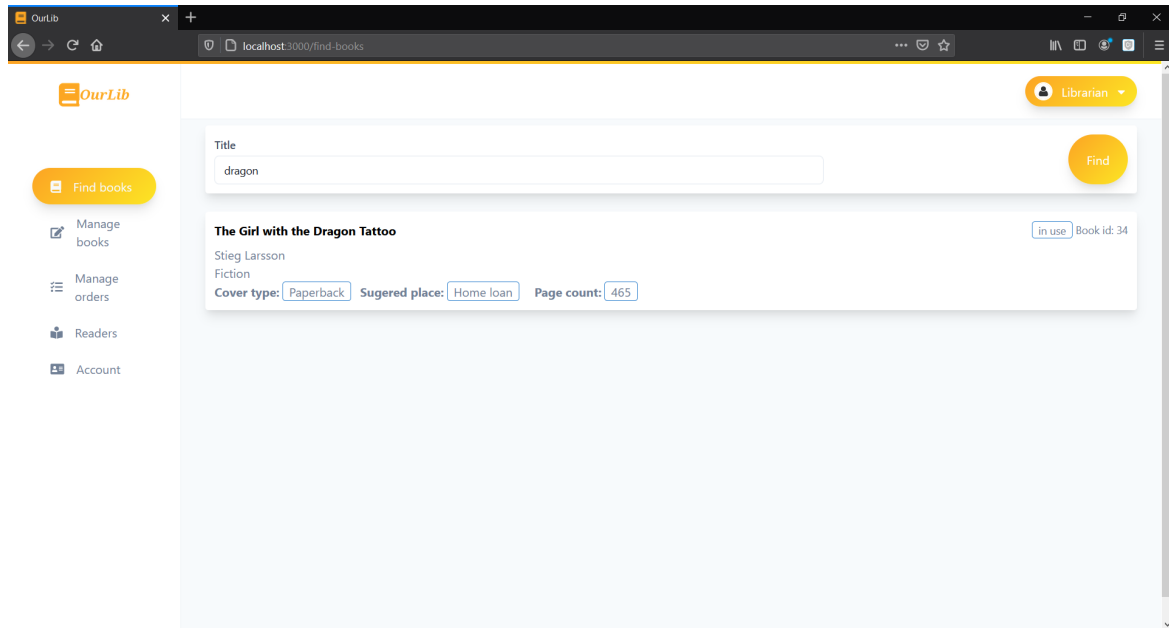


Figure 8: Searching phrase in search bar

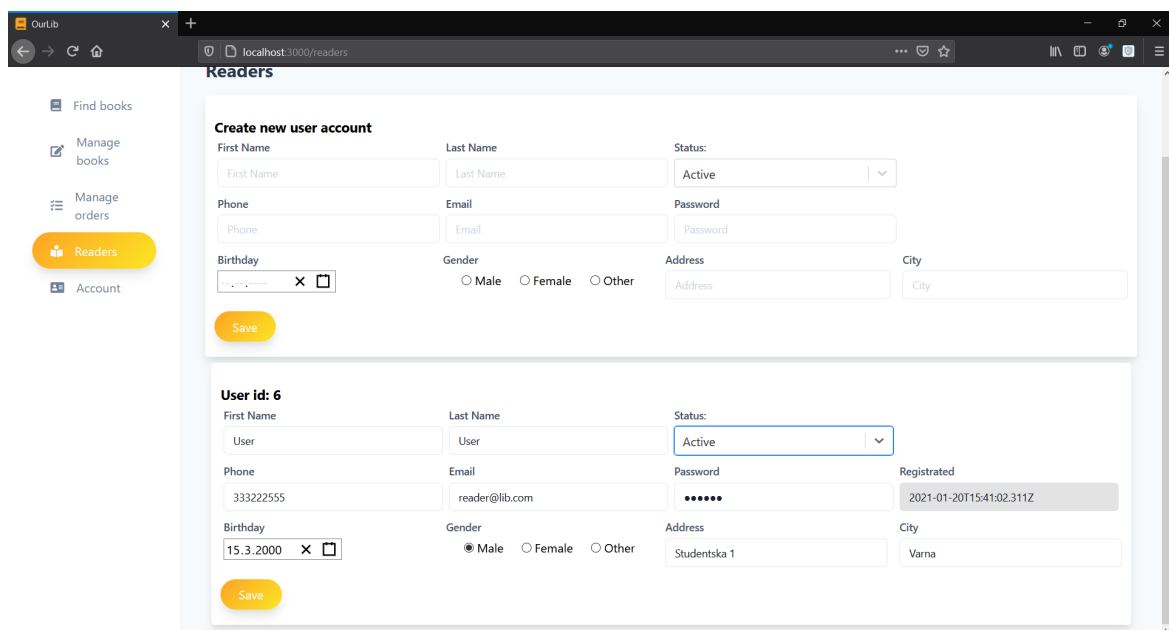


Figure 9: Manage all readers available in system

Order

User id: 5 Book id: 39

User overview
Librarian Librarian
 Phone: 111222333
 Birthdate: 1985-01-10T00:10:00Z City: Varna
 Email: librarian@lib.com Address: Studentska 1

Book overview
Stalk Me
 Jillian Dodd
 Young Adult
 Cover type: Hardcover Case Wrap Suger 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

Manage books

Create new book item

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

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

Save

Book id: 34 Status: in use

Figure 11: Manage all books in system

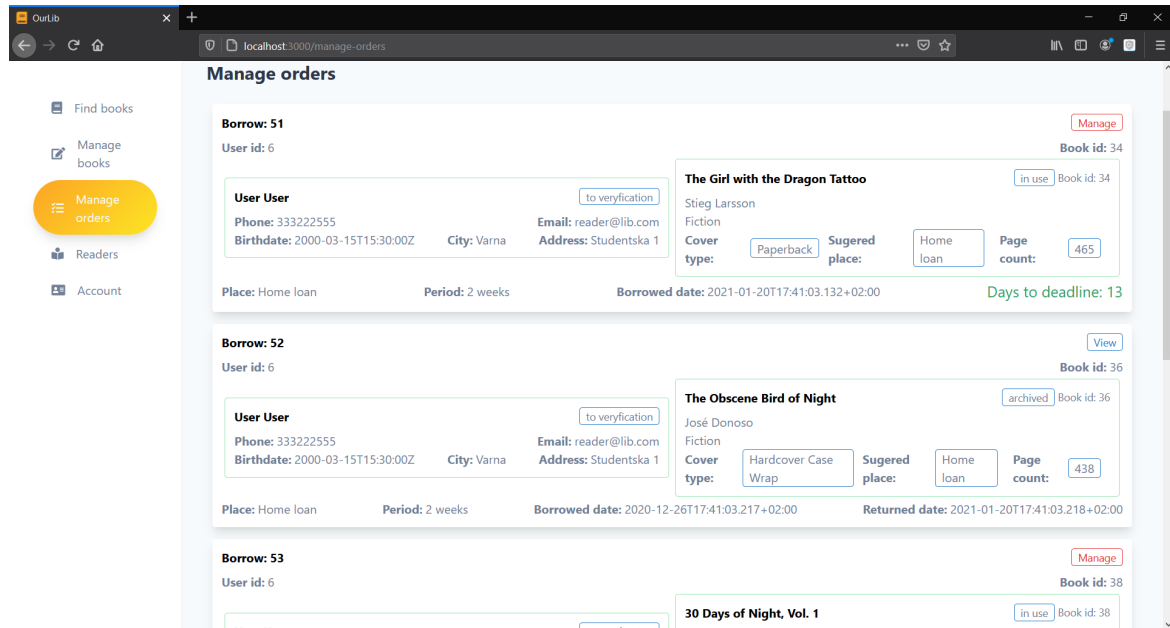


Figure 12: Manage all orders in system

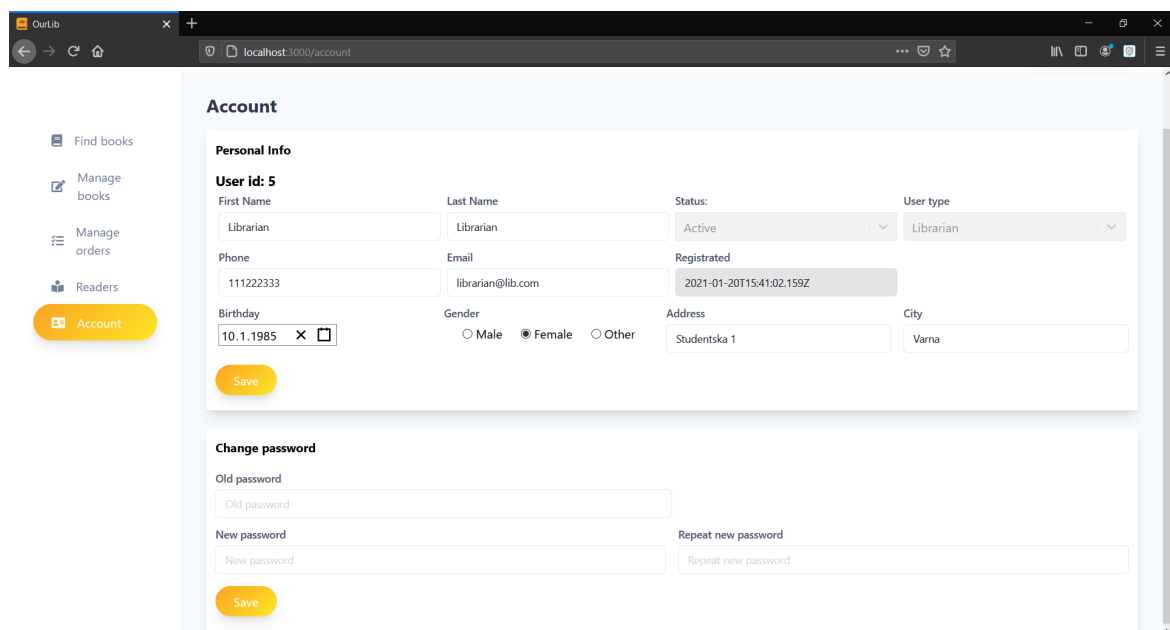


Figure 13: Change credentials of current user

OurLib

Admin

Users

Create new user account

First Name: Last Name: Status: User type:

Phone: Email: Password:

Birthday: Gender: ☐ Male ☐ Female ☐ Other Address: City:

Save

User id: 4

First Name: Last Name: Status: User type:

Phone: Email: Password: Registered:

Birthday: Gender: ☐ Male ☐ Female ☐ Other Address: City:

Figure 14: Manage all users in system

OurLib

Admin

Reports

Submitted forms
Get report

Books
Get report

Users
Get report

User rating
Get report

Figure 15: Reports main navigation page

Submitted report

User id: 6

First Name User	Last Name User	Status: To verification	User type Reader
Phone 333222555	Email reader@lib.com	Password *****	Registered 2021-01-20T15:41:02.311Z
Birthdate 15.3.2000	Gender <input checked="" type="radio"/> Male <input type="radio"/> Female <input type="radio"/> Other	Address Studentska 1	City Varna

[Save](#)

Figure 16: Manage new submitted readers

Books report

Number of books: 10 Number of available books: 5

35	Tailored for Trouble	available	Book id: 35
Mimi Jean Pamfiloff Romance			
Cover type:	Hardcover Dust Jacket	Sugered place:	Reading room
Page count:	354	Borrow	
37	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	Borrow	
39	Stalk Me	available	Book id: 39
Jillian Dodd Young Adult			
Cover type:	Hardcover Case Wrap	Sugered place:	Home loan
Page count:	327	Borrow	
41	Kiss of the Vampire	available	Book id: 41

Figure 17: Books in system report

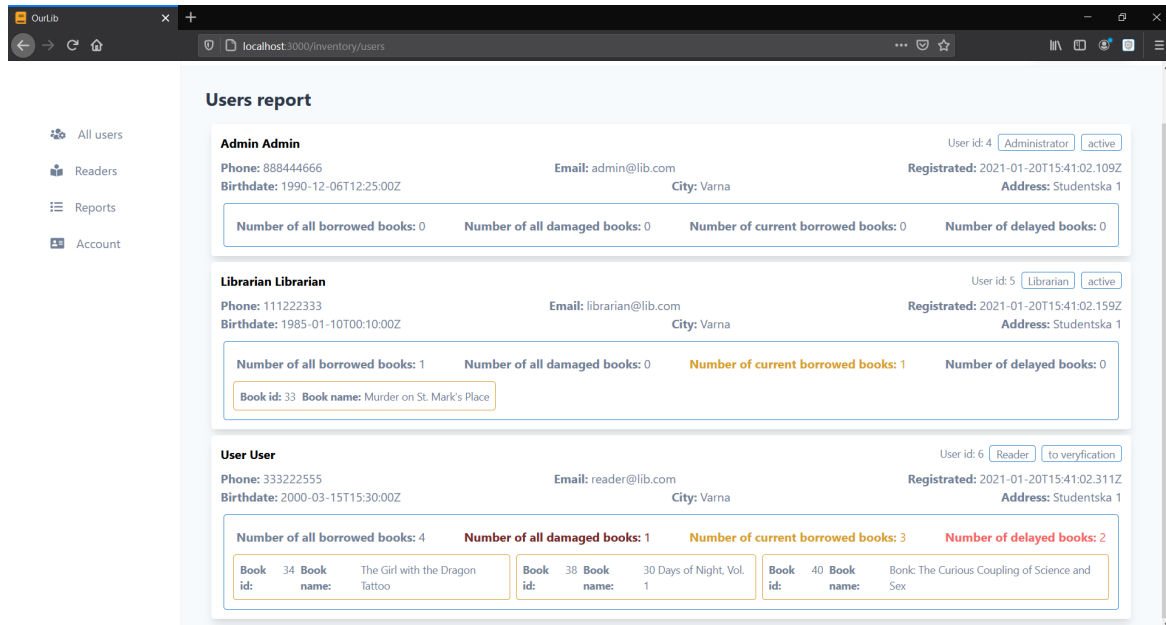


Figure 18: Users in system report

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

```

1  swagger: '2.0'
2  info:
3    description: Library API
4    version: 1.0.3
5    title: Library Varna
6    termsOfService: http://swagger.io/terms/
7    contact:
8      email: kornelstefanczyk@wp.pl
9    license:
10     name: Apache 2.0
11     url: "http://www.apache.org/licenses/LICENSE-2.0.html"
12  tags:
13    - name: user
14      description: Operations about user
15    - name: book
16      description: Everything about your Books
17    - name: library
18      description: Everything about Library
19    - name: dataset
20      description: Print datasets stored in database
21  host: localhost:8080
22  # basePath: /
23  schemes:
24  - http
25  paths:
26    /user:
27      post:
28        tags:
29          - user
30        summary: Create user
31        description: This can only be done by the logged in user.
32        operationId: createUser
33        consumes:
34          - application/json
35        parameters:
36          - in: body
37            name: body
38            description: Created user object
39            required: true
40            schema:
41              $ref: '#/definitions/User'
42          - name: token
43            in: query
44            required: false
45            type: string
46        responses:
47          200:
48            description: successful operation
49          400:
50            description: "Bad request"
51          401:
52            description: Unauthorized, no access
53          403:
54            description: "Forbidden"
55        security:
56          - bearerAuth: []
57  get:

```

```

58     tags:
59       - user
60     summary: Finds user items by params
61     description: Multiple status values can be provided with comma
62       separated strings
63     operationId: findUsersByStatus
64     produces:
65       - application/json
66     parameters:
67       - name: token
68         in: query
69         required: false
70         type: string
71       - name: firstName
72         in: query
73         description: User first name
74         required: false
75         type: string
76       - name: lastName
77         in: query
78         description: User last name
79         required: false
80         type: string
81       - name: email
82         in: query
83         description: User last name
84         required: false
85         type: string
86       - name: userId
87         in: query
88         description: User type id
89         required: false
90         type: integer
91         format: int64
92       - name: status
93         in: query
94         description: User status
95         required: false
96         type: string
97         enum:
98           - active
99           - suspended
100          - inactive
101          - to verification
102         default: available
103       - name: gender
104         in: query
105         description: User gender, not implemented
106         required: false
107         type: string
108     responses:
109       200:
110         description: successful operation
111         schema:
112           type: array
113           items:
114             $ref: "#/definitions/User"

```

```

114         400:
115             description: Invalid status value
116     security:
117         - bearerAuth: []
118
119 /user/login:
120     post:
121         tags:
122             - user
123         summary: Logs user into the system
124         operationId: loginUser
125         consumes:
126             - application/json
127         produces:
128             - application/json
129         parameters:
130             - in: body
131               name: body
132               description: Created user object
133               required: true
134             schema:
135                 $ref: "#/definitions/UserLoginBody"
136     security:
137         - bearerAuth: []
138     responses:
139         200:
140             description: successful operation
141             schema:
142                 $ref: "#/definitions/User"
143             headers:
144                 token:
145                     type: string
146                     description: Personal token generated for user
147                 X-Rate-Limit:
148                     type: integer
149                     format: int32
150                     description: calls per hour allowed by the user
151                 X-Expires-After:
152                     type: string
153                     format: date-time
154                     description: date in UTC when token expires
155         400:
156             description: Invalid username/password supplied
157 /user/logout:
158     get:
159         tags:
160             - user
161         summary: Logs out current logged in user session
162         operationId: logoutUser
163         parameters:
164             - name: token
165               in: query
166               required: false
167               type: string
168     responses:
169         200:
170             description: successful operation

```

```

171         401:
172             description: Unauthorized, no access
173         default:
174             description: successful operation
175     security:
176     - bearerAuth: []
177 "/user/{id}":
178     get:
179         tags:
180         - user
181         summary: Get user by id
182         operationId: getUserById
183         produces:
184         - application/json
185         parameters:
186         - name: id
187           in: path
188           type: integer
189           format: int64
190           description: The id that needs to be fetched.
191           required: true
192         - name: token
193           in: query
194           required: true
195           type: string
196         responses:
197             200:
198                 description: successful operation
199                 schema:
200                     $ref: '#/definitions/User'
201             400:
202                 description: Invalid id supplied
203             401:
204                 description: Unauthorized, no access
205             404:
206                 description: User not found
207         security:
208         - bearerAuth: []
209     put:
210         tags:
211         - user
212         summary: Updated user
213         description: This can only be done by the logged in user.
214         operationId: updateUserById
215         consumes:
216         - application/json
217         parameters:
218         - name: id
219           in: path
220           description: Id that need to be updated
221           required: true
222           type: integer
223           format: int64
224         - in: body
225           name: body
226           description: Updated user object
227           required: true

```

```

228     schema:
229         $ref: '#/definitions/User'
230 - name: token
231     in: query
232     required: true
233     type: string
234 responses:
235     200:
236         description: successful operation
237     400:
238         description: Invalid Id supplied
239     401:
240         description: Unauthorized, no access
241     404:
242         description: User not found
243 security:
244     - bearerAuth: []
245 delete:
246     tags:
247     - user
248     summary: Delete user
249     description: This can only be done by the logged in user.
250     operationId: deleteUserById
251     parameters:
252     - name: id
253       in: path
254       description: The Id that needs to be deleted
255       required: true
256       type: integer
257       format: int64
258     - name: token
259       in: query
260       required: true
261       type: string
262 responses:
263     200:
264         description: successful operation
265     400:
266         description: Invalid Id supplied
267     401:
268         description: Unauthorized, no access
269     404:
270         description: User not found
271 security:
272     - bearerAuth: []
273
274 "/user/{id}/stats":
275 get:
276     tags:
277     - user
278     summary: Get user statistics by user name
279     operationId: getStatsById
280     produces:
281     - application/json
282     parameters:
283     - name: id
284       in: path

```



```

285         required: true
286         type: integer
287         format: int64
288     - name: token
289       in: query
290       required: true
291       type: string
292     responses:
293       200:
294         description: successful operation
295         schema:
296           $ref: '#/definitions/UserStatus'
297       400:
298         description: Invalid Id supplied
299       401:
300         description: Unauthorized, no access
301       404:
302         description: User not found
303     security:
304       - bearerAuth: []
305 /book:
306   post:
307     tags:
308       - book
309     summary: Add a new book to the library
310     operationId: addBook
311     consumes:
312       - application/json
313     parameters:
314       - in: body
315         name: body
316         description: order placed for purchasing the book
317         required: true
318         schema:
319           $ref: '#/definitions/Book'
320     - name: token
321       in: query
322       required: true
323       type: string
324     responses:
325       200:
326         description: successful operation
327       400:
328         description: "Bad request"
329       401:
330         description: Unauthorized, no access
331     security:
332       - bearerAuth: []
333   get:
334     tags:
335       - book
336     summary: Get all books
337     description: Can be done only by librarian
338     operationId: getAllBooks
339     produces:
340       - application/json
341     parameters:

```

```

342     - name: token
343       in: query
344       required: true
345       type: string
346   responses:
347     200:
348       description: successful operation
349       schema:
350         type: array
351         items:
352           $ref: "#/definitions/Book"
353     400:
354       description: Invalid status value
355   security:
356     - bearerAuth: []
357 "/book/{id}":
358   # *****
359   get:
360     tags:
361     - book
362     summary: Get book by book id
363     operationId: getBookById
364     produces:
365     - application/json
366     parameters:
367       - name: id
368         in: path
369         description: The id that needs to be fetched.
370         required: true
371         type: integer
372         format: int64
373       - name: token
374         in: query
375         required: false
376         type: string
377     responses:
378       200:
379         description: successful operation
380         schema:
381           $ref: "#/definitions/Book"
382       400:
383         description: Invalid id supplied
384       401:
385         description: Unauthorized, no access
386       404:
387         description: Book not found
388     security:
389     - bearerAuth: []
390   put:
391     tags:
392     - book
393     summary: Updated book
394     description: This can only be done by the logged in user.
395     operationId: updateBookById
396     consumes:
397     - application/json
398     parameters:

```

```

399     - name: id
400       in: path
401       description: id of book that need to be updated
402       required: true
403       type: integer
404       format: int64
405     - in: body
406       name: body
407       description: Updated user object
408       required: true
409       schema:
410         $ref: '#/definitions/Book'
411     - name: token
412       in: query
413       required: true
414       type: string
415   responses:
416     200:
417       description: successful operation
418     400:
419       description: Invalid id supplied
420     401:
421       description: Unauthorized, no access
422     404:
423       description: Book not found
424   security:
425     - bearerAuth: []
426 delete:
427   tags:
428     - book
429   summary: Delete book
430   description: This can only be done by the logged in user.
431   operationId: deleteBookById
432   parameters:
433     - name: id
434       in: path
435       description: The id of the book that needs to be deleted
436       required: true
437       type: integer
438       format: int64
439     - name: token
440       in: query
441       required: true
442       type: string
443   responses:
444     200:
445       description: successful operation
446     400:
447       description: Invalid id supplied
448     401:
449       description: Unauthorized, no access
450     404:
451       description: Book not found
452   security:
453     - bearerAuth: []
454
455 /book/findByStatus:

```

```

456     get:
457         tags:
458             - book
459         summary: Finds Books by status
460         description: Multiple status values can be provided with comma
461                     separated strings
462         operationId: findBookByStatus
463         produces:
464             - application/json
465         parameters:
466             - name: status
467               in: query
468               description: Status values that need to be considered for
469                           filter
470               required: false
471               type: array
472               items:
473                 type: string
474                 enum:
475                     - available
476                     - all
477                 default: available
478             - name: author
479               in: query
480               description: Author of the book
481               required: false
482               type: string
483             - name: title
484               in: query
485               description: Title of the book
486               required: false
487               type: string
488             - name: genere
489               in: query
490               description: Genere of the book
491               required: false
492               type: string
493         responses:
494             200:
495                 description: successful operation
496                 schema:
497                     type: array
498                     items:
499                         $ref: "#/definitions/Book"
500             400:
501                 description: Invalid status value
502         security:
503             - bearerAuth: []
504 /dataset/author:
505     get:
506         tags:
507             - dataset
508         summary: Get authors dataset
509         description: Get authors dataset
510         operationId: getDatasetAuthors
511         produces:
512             - application/json

```

```

511     parameters:
512         - name: id
513           in: query
514           description: The id of the author. If not entered then all
                    authors are listed
515           required: false
516           type: integer
517           format: int64
518     responses:
519         200:
520             description: successful operation
521             schema:
522                 type: array
523                 items:
524                     $ref: "#/definitions/Author"
525     security:
526         - bearerAuth: []
527 /dataset/coverType:
528     get:
529         tags:
530             - dataset
531         summary: Get cover type dataset
532         description: Get cover type dataset
533         operationId: getDatasetCoverType
534         produces:
535             - application/json
536         parameters:
537             - name: id
538               in: query
539               description: The id of the cover type. If not entered then all
                    cover types are listed
540               required: false
541               type: integer
542               format: int64
543     responses:
544         200:
545             description: successful operation
546             schema:
547                 type: array
548                 items:
549                     $ref: "#/definitions/CoverType"
550     security:
551         - bearerAuth: []
552 /dataset/bookGenre:
553     get:
554         tags:
555             - dataset
556         summary: Get book genre dataset
557         description: Get book genre dataset
558         operationId: getDatasetBookGenre
559         produces:
560             - application/json
561         parameters:
562             - name: id
563               in: query
564               description: The id of the book genre. If not entered then all
                    book genre are listed

```

```

565         required: false
566         type: integer
567         format: int64
568     responses:
569         200:
570             description: successful operation
571             schema:
572                 type: array
573                 items:
574                     $ref: "#/definitions/BookGenre"
575     security:
576         - bearerAuth: []
577
578 /dataset/borrowPeriod:
579     get:
580         tags:
581             - dataset
582         summary: Get borrow period dataset
583         description: Get borrow period dataset
584         operationId: getDatasetBorrowPeriod
585         produces:
586             - application/json
587         parameters:
588             - name: id
589               in: query
590               description: The id of the borrow period. If not entered then
591                 all borrow period are listed
592               required: false
593               type: integer
594               format: int64
595         responses:
596             200:
597                 description: successful operation
598                 schema:
599                     type: array
600                     items:
601                         $ref: "#/definitions/BorrowPeriod"
602         security:
603             - bearerAuth: []
604
605 /dataset/borrowPlace:
606     get:
607         tags:
608             - dataset
609         summary: Get borrow place dataset
610         description: Get borrow place dataset
611         operationId: getDatasetBorrowPlace
612         produces:
613             - application/json
614         parameters:
615             - name: id
616               in: query
617               description: The id of the borrow place. If not entered then
618                 all borrow place are listed
619               required: false
620               type: integer
621               format: int64

```

```

620     responses:
621       200:
622         description: successful operation
623         schema:
624           type: array
625           items:
626             $ref: "#/definitions/BorrowPlace"
627     security:
628       - bearerAuth: []
629
630 /library/inventory/books:
631   get:
632     tags:
633       - library
634     summary: Returns library books
635     description: Returns a stats of the library inventory
636     operationId: getLibraryInventoryBooks
637     parameters:
638       - name: token
639         in: query
640         required: true
641         type: string
642     produces:
643       - application/json
644     responses:
645       200:
646         description: successful operation
647         schema:
648           $ref: "#/definitions/LibraryBooksReport"
649       401:
650         description: Unauthorized, no access
651     security:
652       - bearerAuth: []
653 /library/inventory/users:
654   get:
655     tags:
656       - library
657     summary: Returns library users
658     description: Returns a stats of the library users
659     operationId: getLibraryInventoryUsers
660     parameters:
661       - name: token
662         in: query
663         required: true
664         type: string
665     produces:
666       - application/json
667     responses:
668       200:
669         description: successful operation
670         schema:
671           type: array
672           items:
673             $ref: "#/definitions/UserStatusReport"
674       401:
675         description: Unauthorized, no access
676     security:

```

```

677         - bearerAuth: []
678 /library/inventory/submitted:
679   get:
680     tags:
681     - library
682     summary: Returns library submitted users
683     description: Returns a stats of the library submitted users
684     operationId: getLibraryInventorySubmittedUsers
685     parameters:
686     - name: token
687       in: query
688       required: true
689       type: string
690     produces:
691     - application/json
692     responses:
693       200:
694         description: successful operation
695         schema:
696           type: array
697           items:
698             $ref: "#/definitions/SubmitUserReport"
699       401:
700         description: Unauthorized, no access
701     security:
702     - bearerAuth: []
703 /library/inventory/userrating:
704   get:
705     tags:
706     - library
707     summary: Returns library users rating
708     description: Returns a stats of the library users rating
709     operationId: getLibraryInventoryUsersRating
710     parameters:
711     - name: token
712       in: query
713       required: true
714       type: string
715     produces:
716     - application/json
717     responses:
718       200:
719         description: successful operation
720         schema:
721           type: array
722           items:
723             $ref: "#/definitions/UsersRatingReport"
724       401:
725         description: Unauthorized, no access
726     security:
727     - bearerAuth: []
728 /library/order:
729   post:
730     tags:
731     - library
732     summary: Place an order for a book rent
733     operationId: placeOrder

```



```

734     consumes:
735     - application/json
736     produces:
737     - application/json
738     parameters:
739     - in: body
740       name: body
741       description: order placed for borrow the book
742       required: true
743       schema:
744         $ref: "#/definitions/Borrowed"
745     - name: token
746       in: query
747       required: true
748       type: string
749     responses:
750       200:
751         description: successful operation
752         schema:
753           $ref: "#/definitions/Borrowed"
754       401:
755         description: Unauthorized, no access
756       400:
757         description: Invalid Order
758     security:
759     - bearerAuth: []
760 get:
761     tags:
762     - library
763     summary: Get orders
764     description: Librarian get all orders, Reader takes only owned
765       orders
766     operationId: getAllOrders
767     produces:
768     - application/json
769     parameters:
770     - name: token
771       in: query
772       required: true
773       type: string
774     responses:
775       200:
776         description: successful operation
777         schema:
778           type: array
779           items:
780             $ref: "#/definitions/Borrowed"
781       400:
782         description: Invalid status value
783     security:
784     - bearerAuth: []
785 "/library/order/{orderId}":
786 get:
787     tags:
788     - library
789     summary: Find book order by ID
790     operationId: getOrderById

```

```

790     produces:
791     - application/json
792     parameters:
793     - name: orderId
794       in: path
795       description: ID of borrow form that needs to be fetched
796       required: true
797       type: integer
798       format: int64
799     - name: token
800       in: query
801       required: true
802       type: string
803     responses:
804       200:
805         description: successful operation
806         schema:
807           $ref: "#/definitions/Borrowed"
808       400:
809         description: Invalid ID supplied
810       401:
811         description: Unauthorized, no access
812       404:
813         description: Order not found
814     security:
815     - bearerAuth: []
816 put:
817     tags:
818     - library
819     summary: Find purchase order by ID
820     operationId: putOrderById
821     consumes:
822     - application/json
823     produces:
824     - application/json
825     parameters:
826     - name: orderId
827       in: path
828       description: ID of pet that needs to be fetched
829       required: true
830       type: integer
831       format: int64
832     - in: body
833       name: body
834       description: form of the borrowed book that needs to update
835       required: true
836       schema:
837         $ref: "#/definitions/Borrowed"
838     - name: token
839       in: query
840       required: true
841       type: string
842     responses:
843       200:
844         description: successful operation
845         schema:
846           $ref: "#/definitions/Borrowed"

```

```

847         400:
848             description: Invalid ID supplied
849         401:
850             description: Unauthorized, no access
851         404:
852             description: Order not found
853     security:
854     - bearerAuth: []
855 delete:
856     tags:
857     - library
858     summary: Delete purchase order by ID
859     operationId: deleteOrder
860     consumes:
861     - application/json
862     produces:
863     - application/json
864     parameters:
865     - name: orderId
866       in: path
867       description: ID of the order that needs to be deleted
868       required: true
869       type: integer
870       format: int64
871     - name: token
872       in: query
873       required: true
874       type: string
875     responses:
876     200:
877         description: successful operation
878     400:
879         description: Invalid ID supplied
880     401:
881         description: Unauthorized, no access
882     404:
883         description: Order not found
884     security:
885     - bearerAuth: []
886
887 /library/notification:
888 get:
889     tags:
890     - library
891     summary: Returns notifications of library
892     description: Returns notifications of library
893     operationId: getLibraryNotifications
894     produces:
895     - application/json
896     parameters:
897     - name: token
898       in: query
899       required: true
900       type: string
901     responses:
902     200:
903         description: successful operation

```

```

904         schema:
905             type: array
906             format: chunked
907             items:
908                 $ref: "#/definitions/NotificationForm"
909     401:
910         description: Unauthorized, no access
911     security:
912     - bearerAuth: []
913
914
915 securityDefinitions:
916     bearerAuth:
917         type: apiKey
918         name: api_key
919         in: header
920 definitions:
921     UserLoginBody:
922         type: object
923         properties:
924             email:
925                 description: The email for login
926                 type: string
927                 format: email
928             password:
929                 description: The password for login in clear text
930                 type: string
931     User:
932         type: object
933         properties:
934             id:
935                 type: integer
936                 format: int64
937             userTypeId:
938                 type: integer
939                 format: int64
940             firstName:
941                 type: string
942             lastName:
943                 type: string
944             email:
945                 type: string
946             password:
947                 type: string
948             phone:
949                 type: string
950             birthdate:
951                 type: string
952                 format: date-time
953             registrated:
954                 type: string
955                 format: date-time
956             gender:
957                 type: string
958                 enum:
959                     - male
960                     - female

```

```

961         - other
962     address:
963         type: string
964     city:
965         type: string
966     status:
967         type: string
968         description: User Status
969     enum:
970         - active
971         - suspended
972         - inactive
973         - to verification
974     xml:
975         name: User
976 UserType:
977     type: object
978     properties:
979         id:
980             type: integer
981             format: int64
982         name:
983             type: string
984         permissions:
985             type: string
986     xml:
987         name: UserType
988 SubmitUserReport:
989     allOf:
990         - $ref: '#/definitions/User'
991         - type: object
992         properties:
993             userTypeName:
994                 type: string
995 UserStatusReport:
996     allOf:
997         - $ref: '#/definitions/User'
998         - $ref: '#/definitions/UserStatus'
999         - type: object
1000     properties:
1001         userTypeName:
1002             type: string
1003         currentBorrowed:
1004             type: array
1005             items:
1006                 type: object
1007                 properties:
1008                     bookId:
1009                         type: integer
1010                         format: int64
1011                     bookName:
1012                         type: string
1013
1014 UsersRatingReport:
1015     type: object
1016     properties:
1017         loyal:

```

```

1018         type: array
1019         items:
1020             type: object
1021             properties:
1022                 userId:
1023                     type: integer
1024                     format: int64
1025                 userName:
1026                     type: string
1027         disloyal:
1028             type: array
1029             items:
1030                 type: object
1031                 properties:
1032                     userId:
1033                         type: integer
1034                         format: int64
1035                     userName:
1036                         type: string
1037
1038     Author:
1039         type: object
1040         properties:
1041             id:
1042                 type: integer
1043                 format: int64
1044             firstName:
1045                 type: string
1046             lastName:
1047                 type: string
1048         xml:
1049             name: Author
1050     CoverType:
1051         type: object
1052         properties:
1053             id:
1054                 type: integer
1055                 format: int64
1056             name:
1057                 type: string
1058         xml:
1059             name: CoverType
1060     Borrowed:
1061         type: object
1062         properties:
1063             id:
1064                 type: integer
1065                 format: int64
1066             userId:
1067                 type: integer
1068                 format: int64
1069             bookId:
1070                 type: integer
1071                 format: int64
1072             borrowedDate:
1073                 type: string
1074                 format: date-time

```

```

1075         returnedDate:
1076             type: string
1077             format: date-time
1078         damageNotes:
1079             type: string
1080         placeId:
1081             type: integer
1082             format: int64
1083         periodId:
1084             type: integer
1085             format: int64
1086     xml:
1087         name: Borrowed
1088 Book:
1089     type: object
1090     properties:
1091         id:
1092             type: integer
1093             format: int64
1094         name:
1095             type: string
1096         isbn:
1097             type: string
1098         authorId:
1099             type: integer
1100             format: int64
1101         pageCount:
1102             type: integer
1103         coverTypeId:
1104             type: integer
1105             format: int64
1106         genreId:
1107             type: integer
1108             format: int64
1109         sugeredPeriodId:
1110             type: integer
1111             format: int64
1112         sugeredPlaceId:
1113             type: integer
1114             format: int64
1115         status:
1116             type: string
1117             description: Book Status
1118         enum:
1119             - available
1120             - in use
1121             - archived
1122     xml:
1123         name: Book
1124 BookReport:
1125     allOf:
1126         - $ref: '#/definitions/Book'
1127         - type: object
1128         properties:
1129             authorTypeName:
1130                 type: string
1131             coverTypeName:

```

```

1132         type: string
1133     genreName:
1134         type: string
1135     sugeredPeriodName:
1136         type: string
1137     sugeredPlaceName:
1138         type: string
1139     statusName:
1140         type: string
1141
1142     BookGenre:
1143         type: object
1144         properties:
1145             id:
1146                 type: integer
1147                 format: int64
1148             name:
1149                 type: string
1150         xml:
1151             name: BookGenre
1152     BorrowPeriod:
1153         type: object
1154         properties:
1155             id:
1156                 type: integer
1157                 format: int64
1158             name:
1159                 type: string
1160             period:
1161                 type: integer
1162         xml:
1163             name: BorrowPeriod
1164     BorrowPlace:
1165         type: object
1166         properties:
1167             id:
1168                 type: integer
1169                 format: int64
1170             name:
1171                 type: string
1172         xml:
1173             name: BorrowPlace
1174     LibraryBooksReport:
1175         type: object
1176         properties:
1177             numberOfBooks:
1178                 type: integer
1179             numberOfAvailableBooks:
1180                 type: integer
1181             books:
1182                 type: array
1183             items:
1184                 $ref: '#/definitions/BookReport'
1185         xml:
1186             name: LibraryBooksReport
1187     UserStatus:
1188         type: object

```



```

1189     properties:
1190         numberOfAllBorrowedBooks:
1191             type: integer
1192         numberOfAllDamagedBooks:
1193             type: integer
1194         numberOfCurrentBorrowedBooks:
1195             type: integer
1196         numberOfDelayedBooks:
1197             type: integer
1198     xml:
1199         name: UserStatus
1200 NotificationForm:
1201     type: object
1202     properties: # also possible to model with oneOf+discriminator
1203         id:
1204             type: integer
1205             description: my event id
1206         event:
1207             type: string
1208             description: my event type
1209         data:
1210             type: object # object in case we have a complex structure,
1211                           otherwise string, integer, ..., as usual
1211     xml:
1212         name: UserStatus
1213 UserToken:
1214     type: object
1215     properties:
1216         id:
1217             type: integer
1218             format: int64
1219         addDatetime:
1220             type: string
1221             format: date-time
1222         tokenName:
1223             type: string
1224         userId:
1225             type: integer
1226             format: int64
1227         expireDatetime:
1228             type: string
1229             format: date-time
1230
1231 externalDocs:
1232     description: Find out more about Swagger
1233     url: "http://swagger.io"

```