MyShelf API

Expected Functionality

Get all books

Request: `GET /api/books/`

- A book can only have one course at the moment.
- Images are not implemented

Response:

```
{
  "success": True,
  "data": [
            {
              "id" : 1,
              "title" : "Introduction to Calculus",
              "course" : "MATH 1110",
              "image" : None,
              "listings" : [0, 3, 43, 192]
            },
            {
              "id" : 2,
              "title" : "Algorithm Design",
              "course" : "CS 4820",
              "image" : None,
              "listings" : [29, 323, 4, 13]
            }
          ]
}
```

Example iOS Response model:

(Note for Backend: you don't need to make these for iOS, the following is a guide for iOS)

```
struct Class {
  var id: Int,
  var code: String,
  var name: String,
  var assignments: [Assignment],
  var students: [Student],
  var instructors: [Instructor]
  // or, if students and instructors are both Users, make them [User], up to you
}

struct ClassesResponse {
  var success: Bool
  var data: [Class]
}
```

Example iOS Alamofire request:

```
static func getClasses(completion: @escaping ([Class]) -> Void) {
  let endpoint = "\(endpointVariable)/api/classes/"
  Alamofire.request(endpoint, method: .get).validate().responseData { response in
    switch response.result {
    case .success(let data):
        let jsonDecoder = JSONDecoder()
        if let classesResponse = try? jsonDecoder.decode(ClassesResponse.self, from: data) {
            completion(classesResponse.data)
        } else {
            print("Invalid Response Data")
```

```
        }
    case .failure(let error):
        print(error.localizedDescription)
    }
  }
}
```

Get books by course

Request: `GET /api/books/course/{string:course name}/`

- By course name, the method expects “CS%201110”, not “Introduction to Computing Using Python”. The &20 stands in for a space.
- If there are no books for that course, the method returns an empty list rather than an error.

Response:

```
{
  "success": True,
  "data": <List of dict representations of books, as above>
}
```

Get book by title

Request: `GET /api/books/book/{string: book title}/`

- Here the title would be "Introductory%20Calculus”

Response:

```
{
  "success": True,
  "data": {
      "id" : 1,
      "title" : "Introductory Calculus",
      "course" : "MATH 1110",
      "image" : None,
      "listings" : [12, 224, 23]
```

```
    }
}
```

Get book by ID

Request: `GET /api/books/book/id/{int: book id}/`

- Here the id would be 0.

Response:

```
{
  "success": True,
  "data": {
      "id" : 1,
      "title" : "Introductory Calculus",
      "course" : "MATH 1110",
      "image" : None,
      "listings" : [12, 224, 23]
    }
}
```

Get user by net ID

Request: `GET /api/user/{string: net ID}/`

- Profile picture not currently implemented
- Note that the listings for a user are stored by their ID, not their full dict representation

Response:

```
{
  "success": True,
  "data": {
            "id" : 1,
            "name" : Hartek Sabharwal,
            "netid" : hs786,
```

```
            "pfp" : None,
            "listings" : [0, 10, 283, 392]
          }
}
```

Get listing by ID

Request: `GET /api/listing/{int: listing ID}/`

- Error if the listing does not exist.

Response:

```
{
  "success": True,
  "data":
          {
            "id" : 3,
            "title" : "Introduction to Statistics",
            "price" : "27.83",
            "condition": "good",
            "notes" : "My dog ate the front cover.",
            "image" : None,
            "course" : "STSCI 2100",
            "seller" : 10,
            "book" : 273
          }
}
```

Get listings by user

Request: `GET /api/listings/user/{string: net ID}/`

- Error if the user does not exist. Empty list if the user is not selling anything.

Response:

```
{
```

```
  "success": True,
  "data": [
            {
              "id" : 1,
              "title" : "Introduction to Statistics",
              "price" : "27.83",
              "condition": "good",
              "notes" : "My dog ate the front cover.",
              "image" : None,
              "course" : "STSCI 2100",
              "seller" : 10,
              "book" : 273
            },
            {
              "id" : 29,
              "title" : "Algorithm Design",
              "price" : "3.23",
              "condition": "okay",
              "notes" : "My friend drew a thing on a lot of th
e pages.",
              "image" : None,
              "course" : "CS 4820",
              "seller" : 10,
              "book" : 932
              }
          ]
}
```


Create a user

Request: `POST /api/users/`

- The `pfp` argument in the body is optional

Body:

```
{
  "name": "Hartek Sabharwal",
  "netid": "hs786",
  "pfp" : "/images/users/hs786.png"
}
```

Response:

```
{
  "success": True,
  "data": {
            "id" : 1,
            "name" : Hartek Sabharwal,
            "netid" : hs786,
            "pfp" : "/images/users/hs786.png",
            "listings" : []
          }
}
```

Example iOS Response model:

```
struct PostResponse {
  var success: Bool
  // Note: you don"t need data here because you"re POST-ing, n
ot getting data back.
  // You can choose whether or not you want to have the data v
ariable here.
  // You need the "success" variable here because you want to
know if you successfully
  // sent the information to the backend.
}
```

Example iOS Alamofire request:

```
static func createClass(code: String, name: String, completio
n: @escaping (Bool) -> Void) {
  let postEndpoint = "\(endpointVariable)/api/classes/"
  let parameters: [String: Any] = [
                  "code": code,
                  "name": name
  ]
  Alamofire.request(postEndpoint, method: .post, parameters: p
arameters, encoding: URLEncoding.default, headers: [:]).valida
te().responseData { response in
      switch response.result {
      case .success(let data):
          let jsonDecoder = JSONDecoder()
          if let postResponse = try? jsonDecoder.decode(PostRe
sponse.self, from: data) {
              completion(postResponse.success)
          } else {
              print("Invalid Response Data")
          }
      case .failure(let error):
          print(error.localizedDescription)
    }
  }
}
```

Add a listing

Request: `POST /api/listings/`

- The condition, image, and notes fields are optional.
- Might add author and edition field at some point?
- Error if the user does not exist.

Body:

```
{
  "title" : "Algorithm Design",
  "netid" : "hs786",
  "course" : "CS 4820",
  "price" : "27.29",
  "condition" : "ehh",
  "notes" : "My friend drew a thing on some of the pages.",
  "image" : "/images/books/algorithm_design.png"
}
```

Response:

```
{
  "success": True,
  "data": {
      "id" : 1,
      "title" : "Algorithm Design",
      "course" : "CS 4820",
      "price" : "27.29",
      "condition": "ehh",
      "notes" : "My friend drew a thing on some of the page
s.",
      "image" : "/images/books/algorithm_design.png",
      "seller" : 1,
      "book" : 2
    }
}
```

Delete listing by ID

Request: `DELETE /api/listing/{int: listing ID}/`

- Error if the listing doesn’t exist to begin with.

Response:

```
{
  "success": True,
  "data": {
      "id" : 29,
      "title" : "Algorithm Design",
      "course" : "CS 4820",
      "price" : "27.29",
      "condition": "ehh",
      "notes" : "My friend drew a thing on some of the page
s.",
      "image" : "/images/books/algorithm_design.png",
      "seller" : 1,
      "book" : 2
    }
}
```