MiniBid Restful API

**Table of Contents**

# MiniBid microservices technical report ............................................................................... 3

**Description ................................................................................................................................ 3**

**MiniBid key functionalities: ....................................................................................................... 3** ***MiniBid microservices architecture .................................................................................... 3***

# Software setup and libraries .............................................................................................. 4

**Setup ......................................................................................................................................... 4**

**Libraries ..................................................................................................................................... 4**

# MiniBid Development ........................................................................................................ 5

**Software files and folder Structure ............................................................................................ 5** **app.js ......................................................................................................................................... 6**

**Models and database design ...................................................................................................... 7**

Database design: ........................................................................................................................................... 7

Models .......................................................................................................................................................... 7

**Routes ...................................................................................................................................... 10**

Auth route ................................................................................................................................................... 10

Item route ................................................................................................................................................... 11

Auction route .............................................................................................................................................. 12

Bid route ..................................................................................................................................................... 15

**Verification and Joi validation ................................................................................................... 17**

User ............................................................................................................................................................. 17

Item ............................................................................................................................................................. 18

Auction ........................................................................................................................................................ 18

Bid ............................................................................................................................................................... 18

**JWT Authentication .................................................................................................................. 19**

**API Endpoints and application flow ........................................................................................... 20**

MiniBid application flow: ............................................................................................................................ 20

Endpoints: ................................................................................................................................................... 20

**Testing ...................................................................................................................................... 21**

**Deploy your MiniBid project into a GCP VM using Docker ......................................................... 35**

**Future improvements ............................................................................................................... 38**

**Resources ................................................................................................................................. 39**

**Reference: ................................................................................................................................ 39**

**Acknowledgments: ................................................................................................................... 39**

# MiniBid microservices technical report

## Description

The aim of this project is to develop, test and host on a Google Cloud platform a MiniBid RESTful SaaS for auctioning system where users sell items and other users can bid for items in auction. The software supports following actions:

* Action 1: Authorise users to access the auctioning API using the oAuth v2 protocol.
* Action2. Authorised users could create items.
* Action 3: Authorised users could post items for selling in the auctioning API.
* Action 4: Authorised users could browse all items for sale in the auctioning API.
* Action 5: Authorised users could bid for an item while it is in an auction.
* Action 6: Highest bid authorised user wins the item after the end of the auction.
* Action 7: Authorised users could browse the auction bidding history of a sold item.

MiniBid key functionalities:

* User data validation
* User authentication
* Last highest bidder is a winner after auction is closed
* Portable
* Users can see if there are any bids for the auctions that are open

## MiniBid microservices architecture

To meet project requirements following software tools were used to develop MiniBid RESTful API:

* Postman
* Mongo DB Atlas Cloud
* NodeJS

The following figure demonstrates the MiniBid microservice architecture.

|  |  |  |  |
| --- | --- | --- | --- |
| Front  -  end    Postman | |  | | --- | | NodeJS | | User    Bid    Auction    Item    Mongo  DB |
|  |
| |  | | --- | | Web  Server | |

## Software setup and libraries

Setup

Since NodeJS framework is used to develop MiniBid Microservice, node.js should be installed locally on computer. Link to read official documentation of Node https://nodejs.org/en/

Once Node is successfully installed, create folder and choose appropriate name. Go inside folder (cd folder name) and run command in terminal npm init

This command will initialise Node JS application.

Libraries

Next, step is to install libraries for this project using npm install in terminal:

npm install express nodemon mongoose body-parser dotenv joi bcryptjs jsonwebtoken morgan moment @joi/date joi-oid

Below are short descriptions of libraries [source Google].

* @joi/date, joi, joi\_oid – node.js middleware for user input validation (date, object id)
* Bcryptjs - password encryption tool, stores hashed passwords
* dotnev - Loads environment variables from .env file
* Body-parser - body parsing middleware
* Express – framework to develop APIs
* Jsonwebtoken - open standard used to share security information between two parties - a client and a server
* moment -date parser
* nodemon – CLI interface utility, that doesn’t require restart server after changes to files are made.
* Morgan – HTTP request logger
* Mongoose – JavaScript object-oriented programming library that creates connection between MongoDb and Express web application framework

After packages has been successfully installed and

"scripts": {

"start": "nodemon app.js"

},

has been updated to use nodemon package, package.json file should look like code bellow:

{

"name": "auctionapp",

"version": "1.0.0",

"description": "Auction app to bid for items",

"main": "app.js",

"scripts": {

"start": "nodemon app.js"

},

"author": "Mindaugas Pranaitis",

"license": "ISC",

"dependencies": {

"@joi/date": "^2.1.0",

"bcryptjs": "^2.4.3",

"body-parser": "^1.19.1",

"dotenv": "^16.0.0",

"express": "^4.17.2",

"joi": "^17.6.0",

"joi-oid": "^1.1.2",

"jsonwebtoken": "^8.5.1",

"moment": "^2.29.1",

"mongoose": "^6.2.0",

"morgan": "^1.10.0",

"nodemon": "^2.0.15"

}

}

## MiniBid Development

This section briefly discusses project folder structure, database design, routing, and validation/verification.

Software files and folder Structure

Following picture shows project file and folder structure:

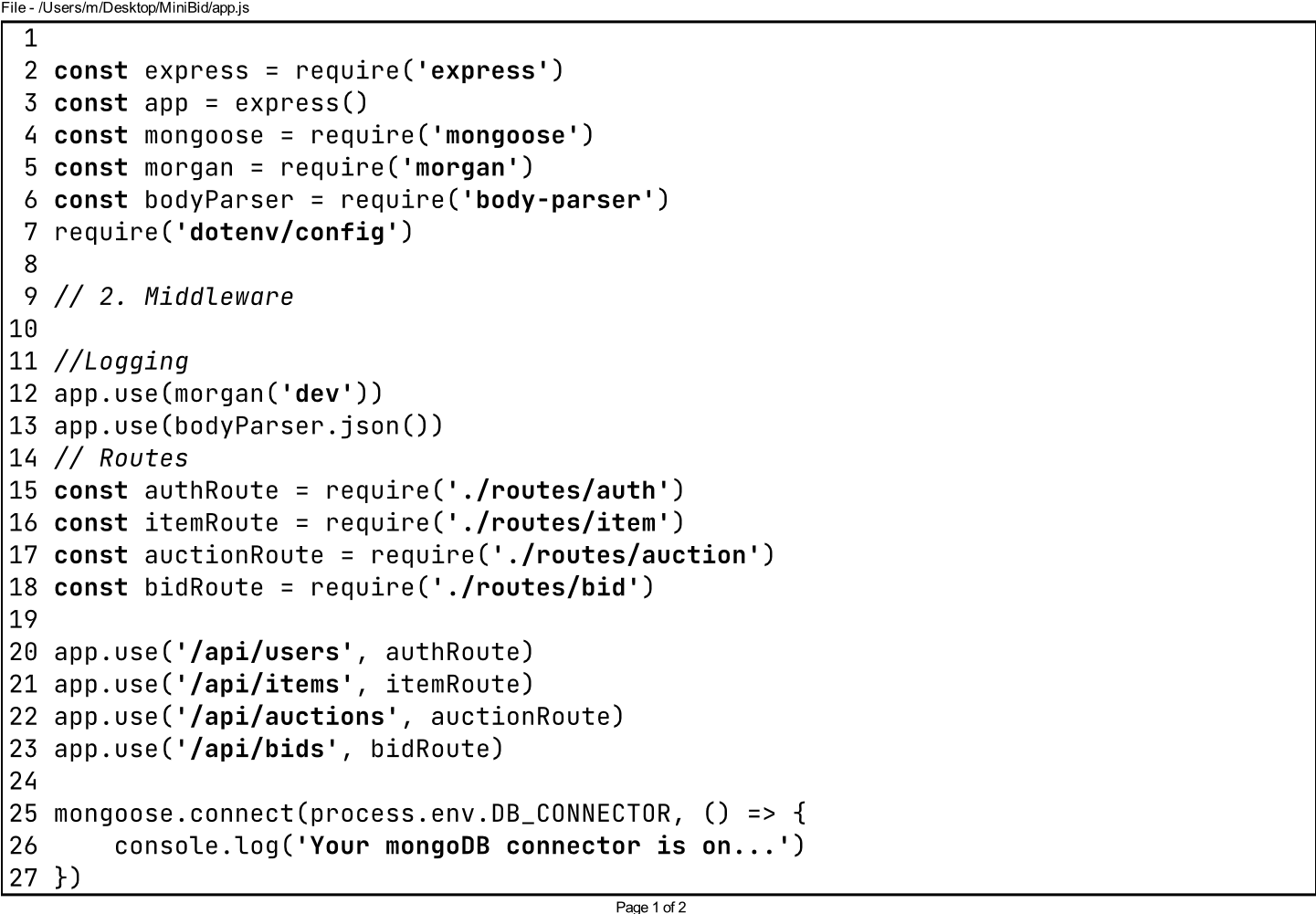


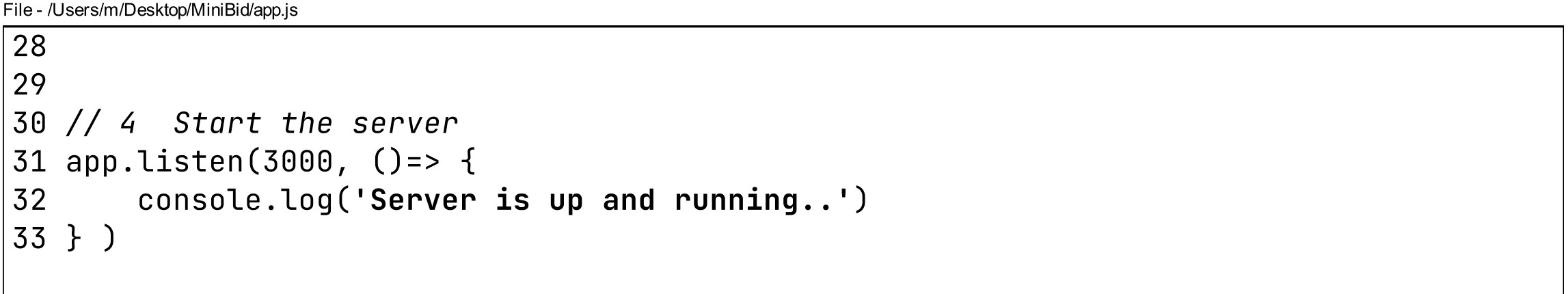
1. Models – folder contains database schemas
2. node\_modules – folder contains node libraries
3. Routes – route handlers
4. Validations – user data validations with joi
5. .env – environment variables
6. app.js – server configuration file
7. package.json - project dependencies
8. verifyToken – logic for verifying user token and allowing to access resources

app.js

App.js is web server configuration file in which all required libraries, routes are imported.

All code, see picture, is read from top to bottom. First libraries are loaded, then routes, after routes loaded, line 20-23 sets paths to resources, then mongoose connects to Cloud-hosted MongoDB instance (link to database is stored in .env file) and last app. listen tells server which port to listen to.





Models and database design

Database design:

Looking at the project requirements, there are 4 possible entities:

User, Auction, Item, Bid.

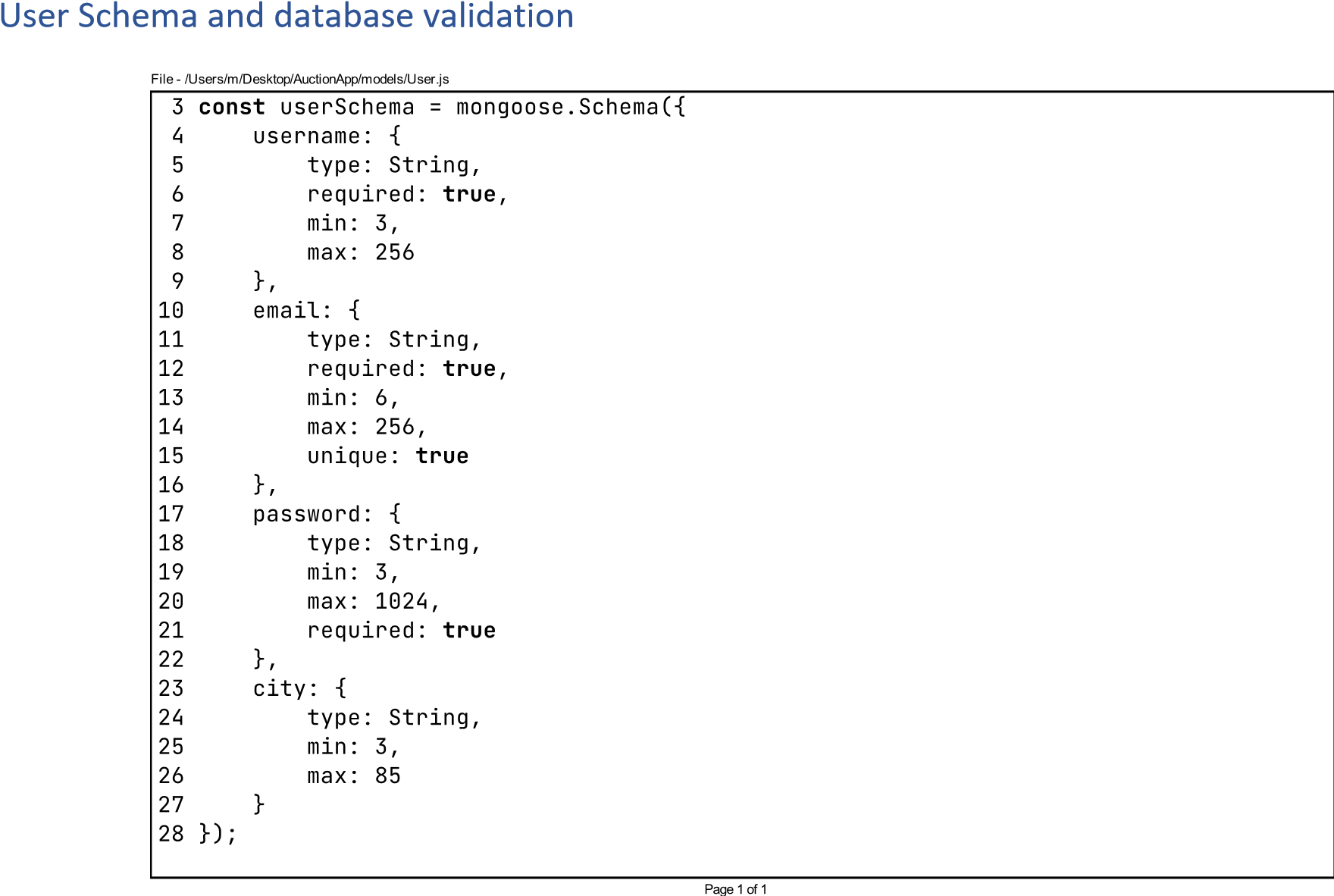
While Auction collection could contain nested objects such as Item and Bid or subdocuments, Mongo DB gives an opportunity to reference collections and perform SQL like left table joins. This approach I found most suitable for MiniBid database design considering data normalization and flexibility when performing queries [1]

### Models

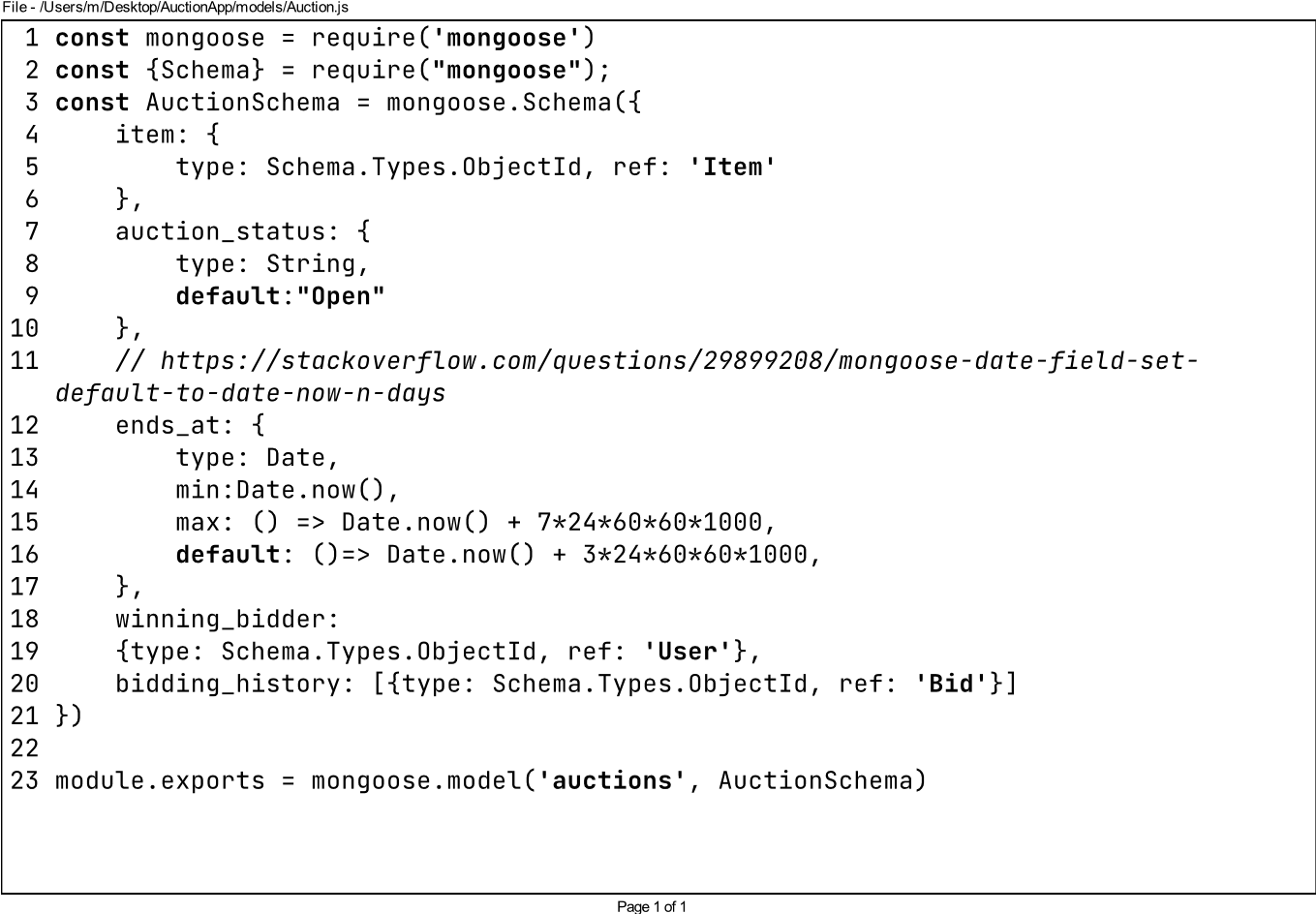
Models are responsible for creating and reading documents from MongoDb. MiniBid has 4 models:

User, Auction, Item, Bid.

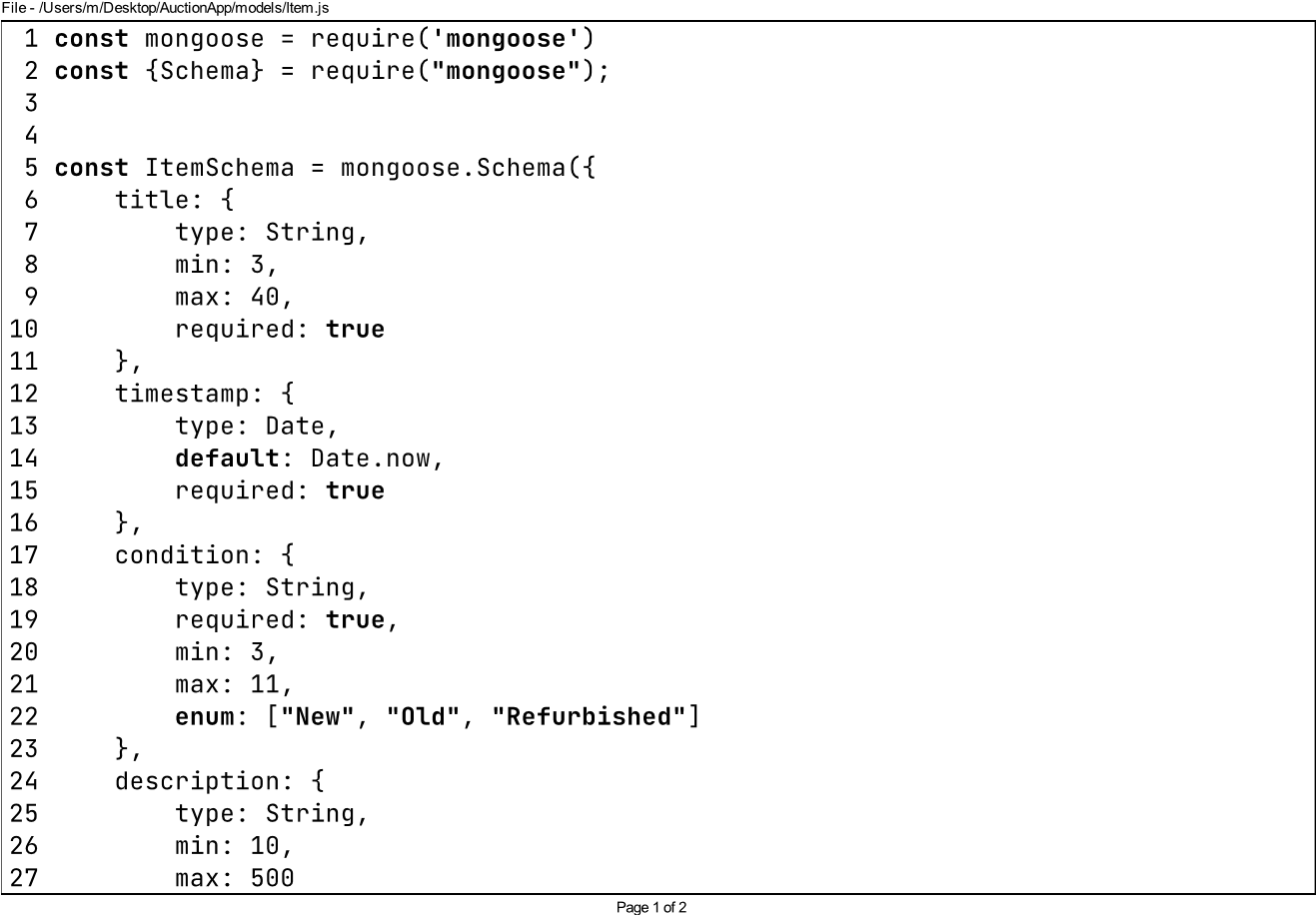
Before user data is saved to database, constrains (min, max, type, unique) are applied for all models.

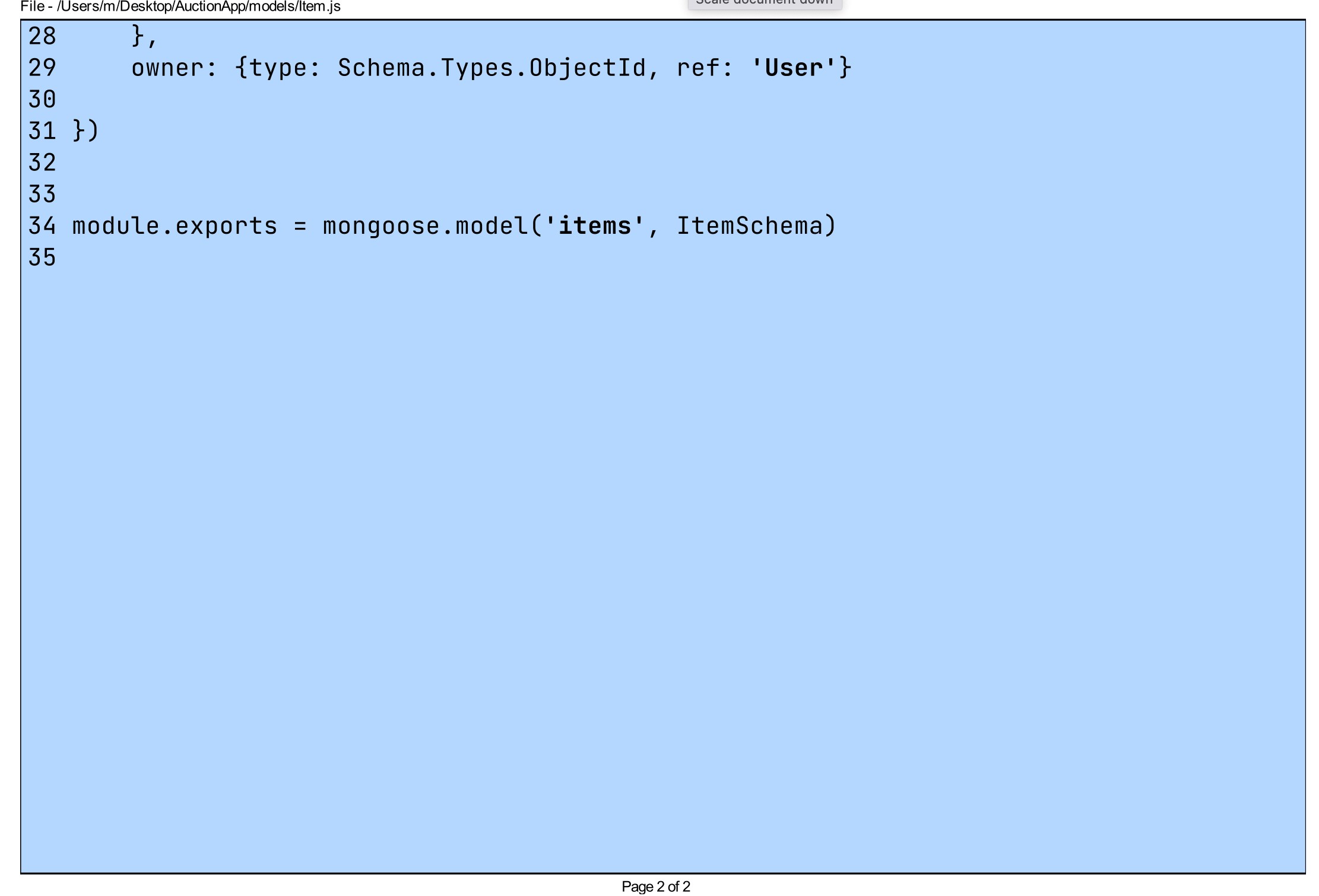


Auction schema and database validation



Item schema and database validation





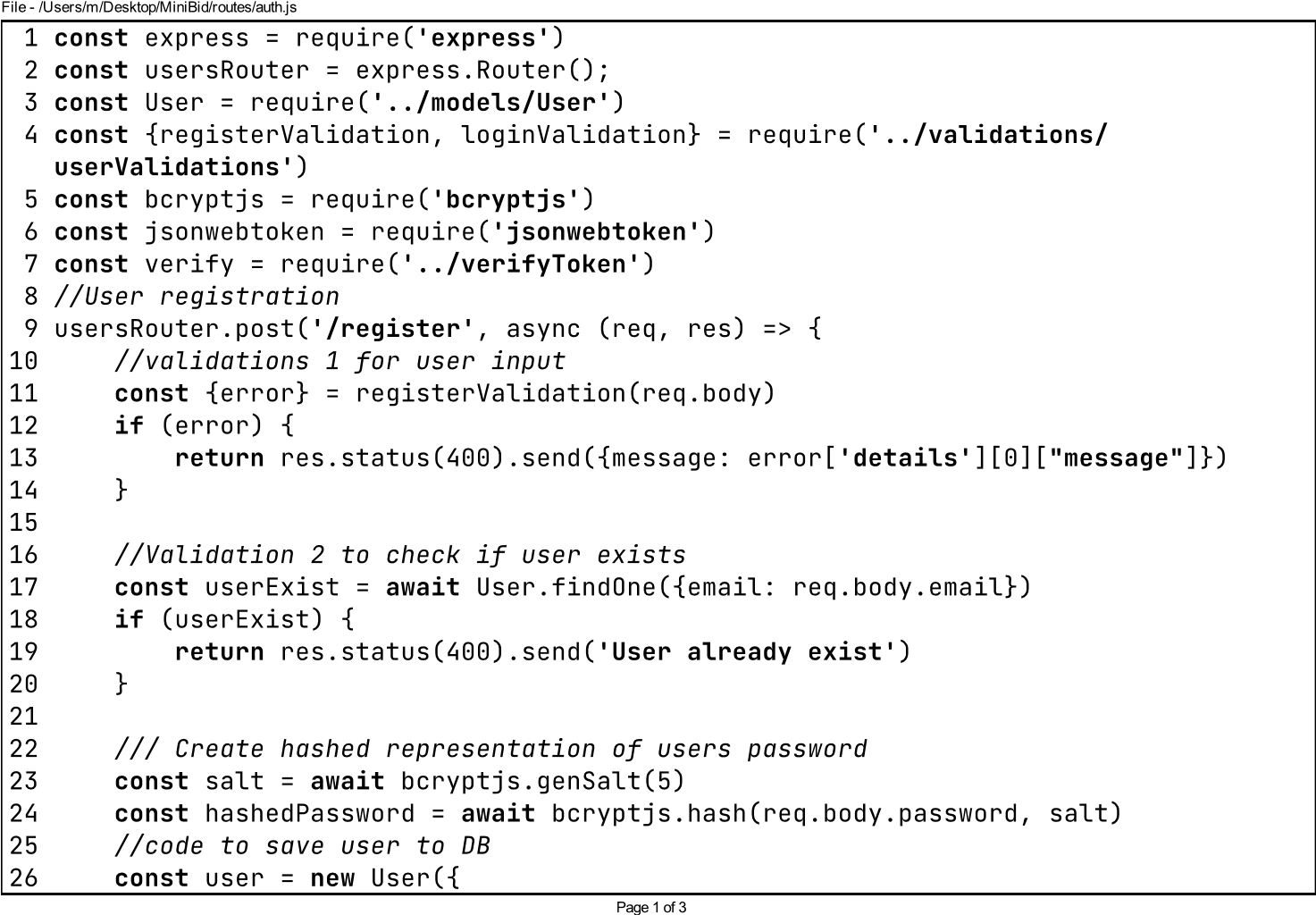
Bid schema and database validation

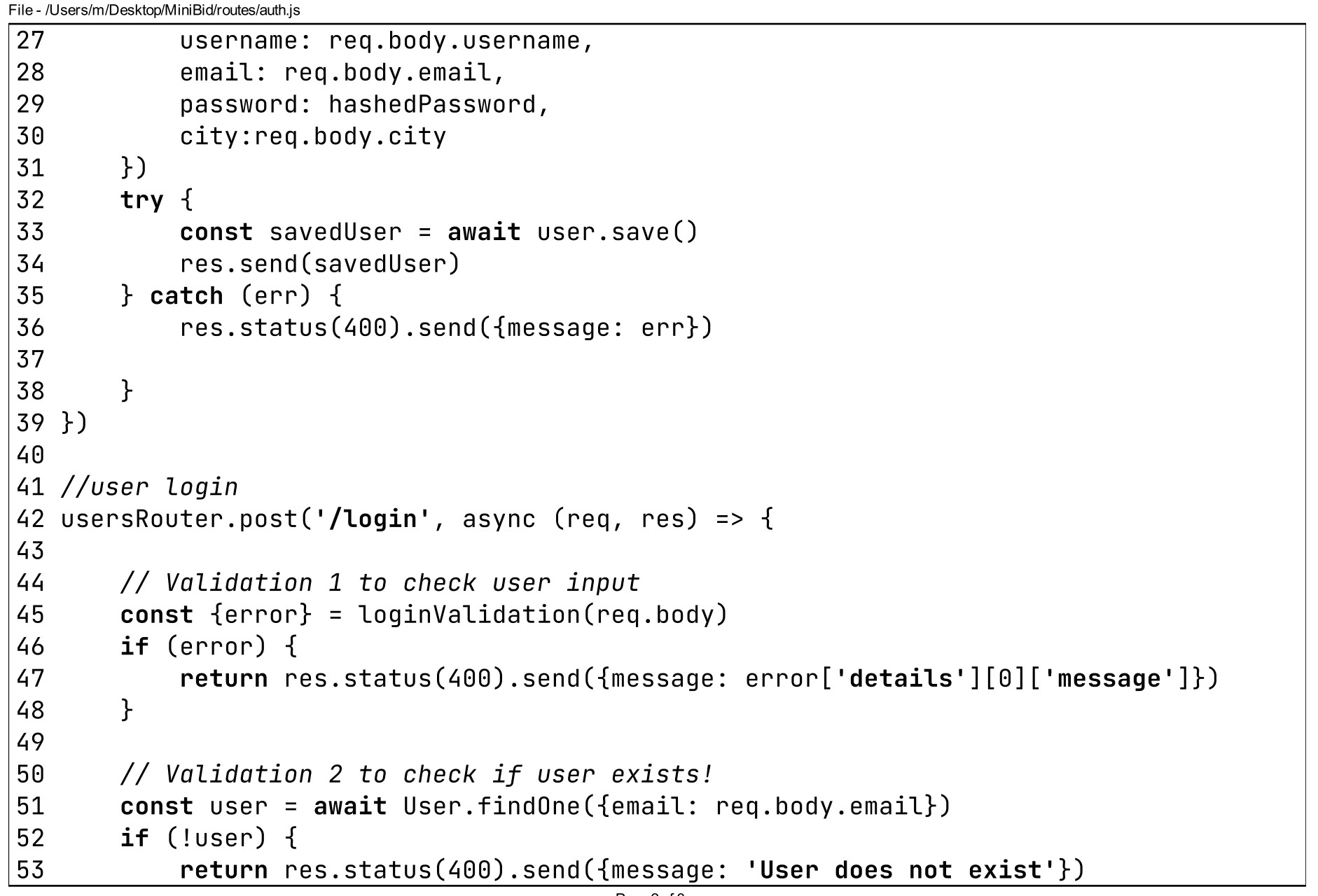
|  |  |
| --- | --- |
|  |  |
|

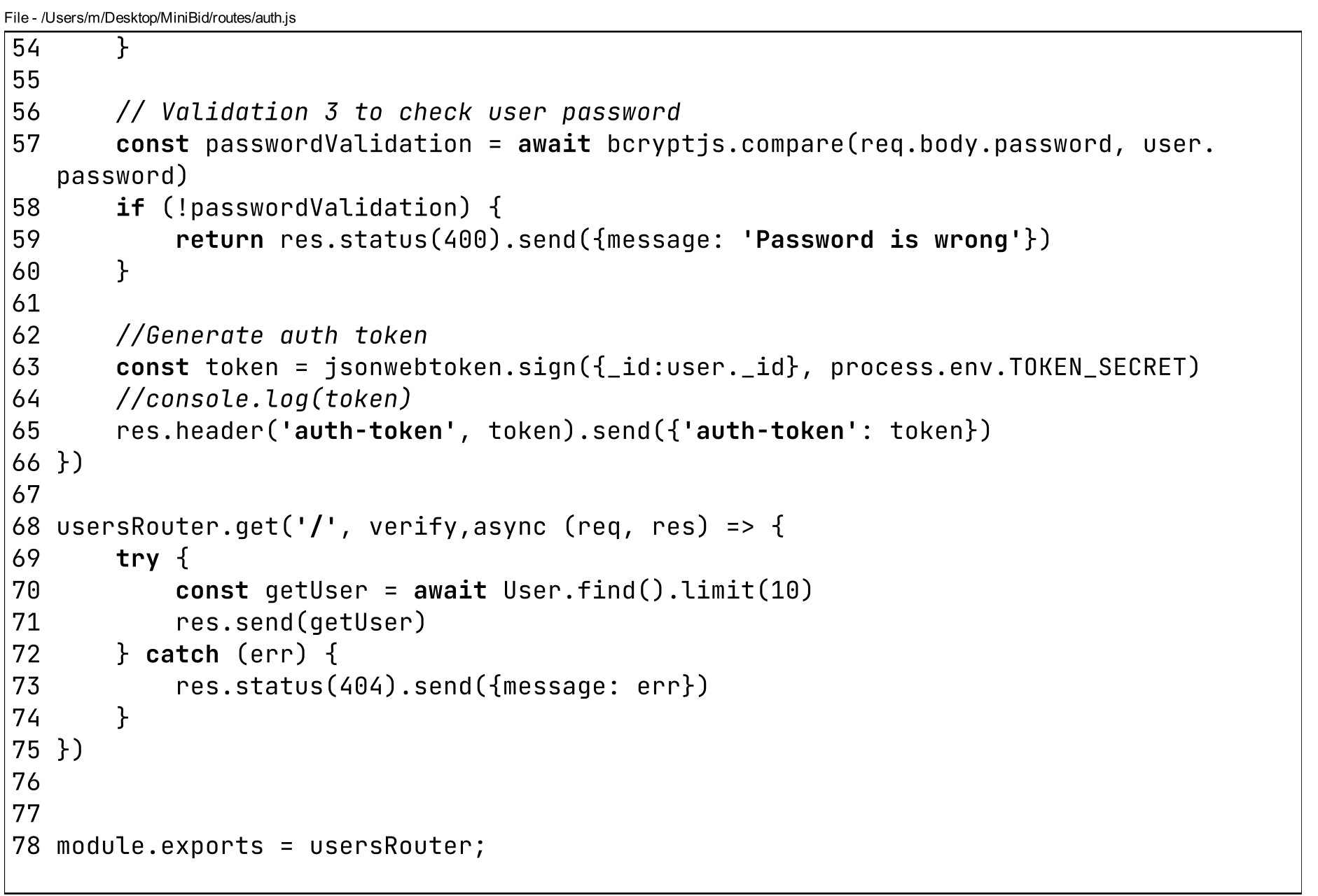
## Routes

There are 4 main routes for MiniBid API:

Auth route – is responsible for handling user registration and login requests, hashing user passwords, issuing auth-token to the logged in users and verifying user data shown in pictures:

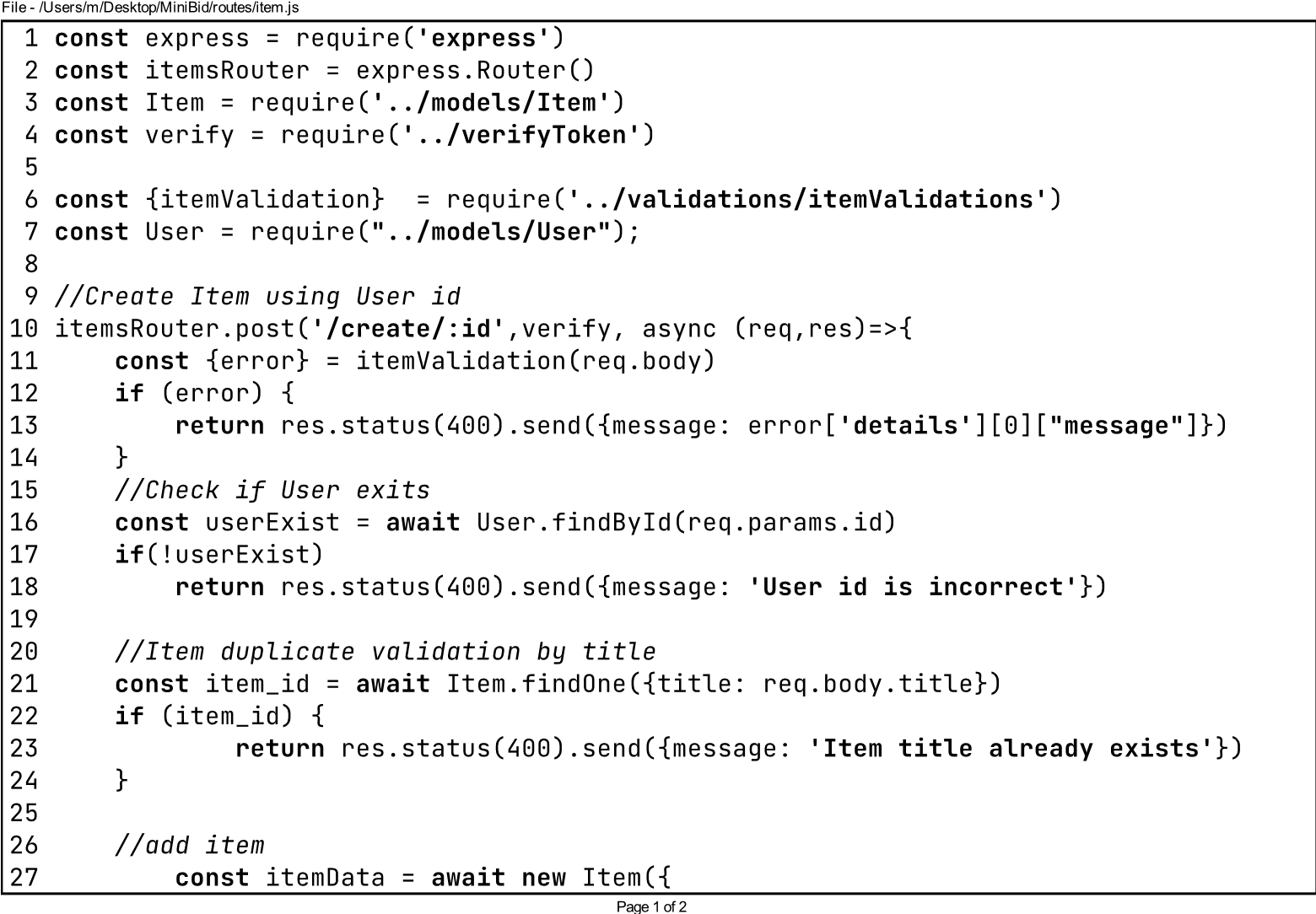






### Item route

Item routes handle user request for creating and reading items data from database. Additionally, there is item validation which enforces unique items (by title) to be created.

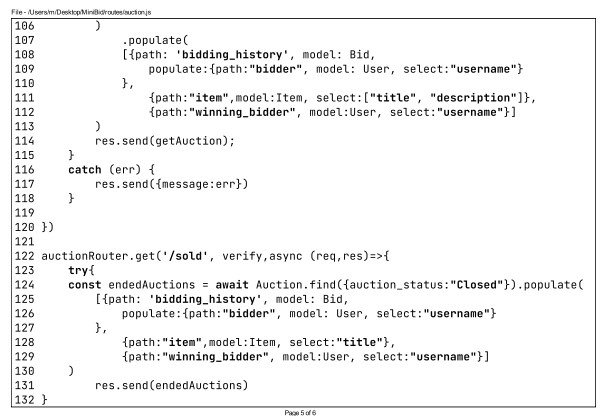
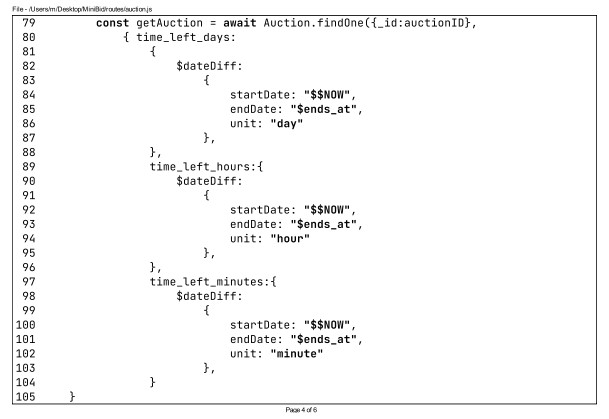
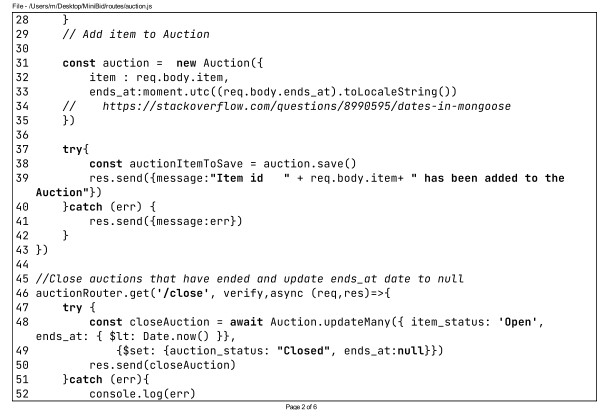




### Auction route

Auction route handles user request for adding items to the auction, displaying active and closed auctions with items sold, auction item bidding history.

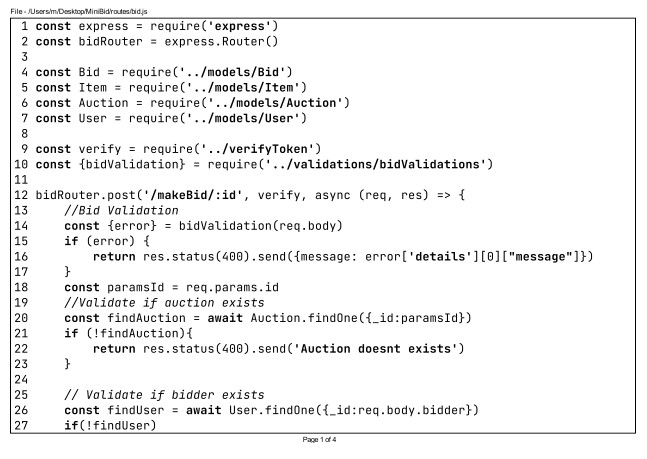


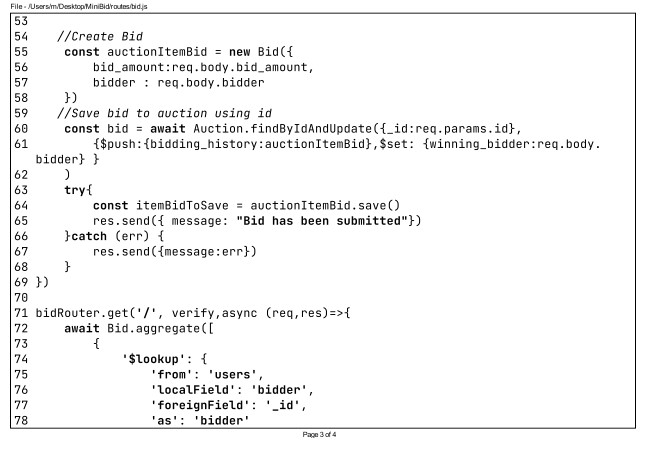
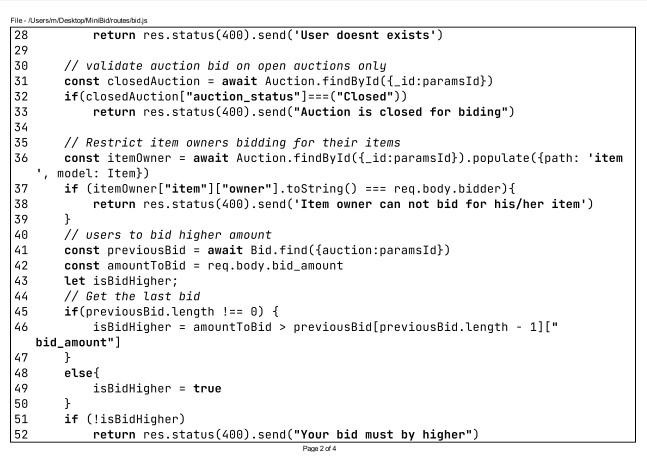


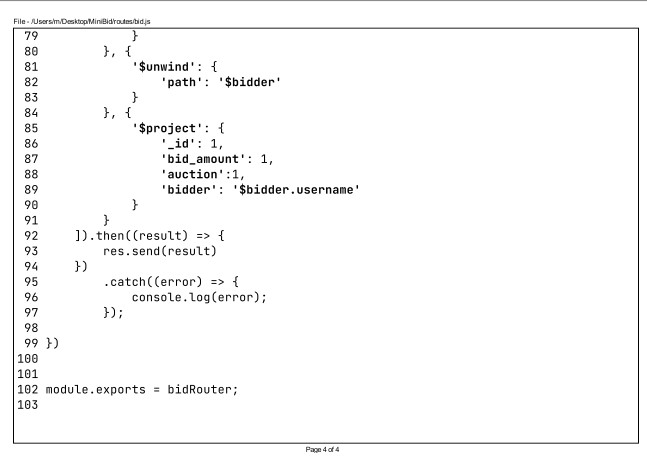


### Bid route

Bid route handles user request for bidding for an active auction which has item that user wants to bid for.







## Verification and Joi validation

Verification is process where data existence in database is checked. Combining verification and validation we ensure data accuracy and existence. For this project validation is done using Joi library. User

Verification

* Validation 2 to check if user exists
* Validation 2 to check if user exists
* Validation 3 to check user password

Joi validation

|  |
| --- |
| const registerValidation = (data) => { const schemaValidation = joi.object({ username: joi.string().required().min(3).max(256), email: joi.string().required().min(6).max(256).email(), password: joi.string().required().min(6).max(1024), city: joi.string().min(3).max(85)  })  return schemaValidation.validate(data)  }  const loginValidation = (data) => { const schemaValidation = joi.object({ |

email: joi.string().required().min(6).max(256).email(), password: joi.string().required().min(6).max(1024),

})

return schemaValidation.validate(data)

}

### Item

Verification

* Check if User exits
* Item duplicate validation by title

Joi Validation

const itemValidation = (data) => { const schemaValidation = joi.object({ title: joi.string().required().min(3).max(40), condition: joi.string().valid("New", "Old", "Refurbished").required().min(3).max(4), description: joi.string().required().min(10) .max(500)

})

return schemaValidation.validate(data)

}

### Auction

Verification

* Check if item exists in Auction
* Check if item exists. Item has been created.

Joi Validation

|  |
| --- |
| const auctionValidation = (data) => { const schemaValidation = joi.object({ item:***Joi***.objectId().required(),  ends\_at: joi.date().format(['YYYY/MM/DD', 'DD-MM-YYYY']).greater("now").max(Date.now() + 7  \* 24 \* 60 \* 60 \* 1000)  })  return schemaValidation.validate(data)  } |

### Bid

Verification

* Validate if auction exists
* Validate if bidder exists
* Validate auction bid on open auctions only
* Restrict item owners bidding for their items
* Users to bid higher amount

Joi Validation

const bidValidation = (data) => { const schemaValidation = joi.object({ bidder:***Joi***.objectId().required(), bid\_amount: joi.number().required().min(1).max(1000)

})

return schemaValidation.validate(data)

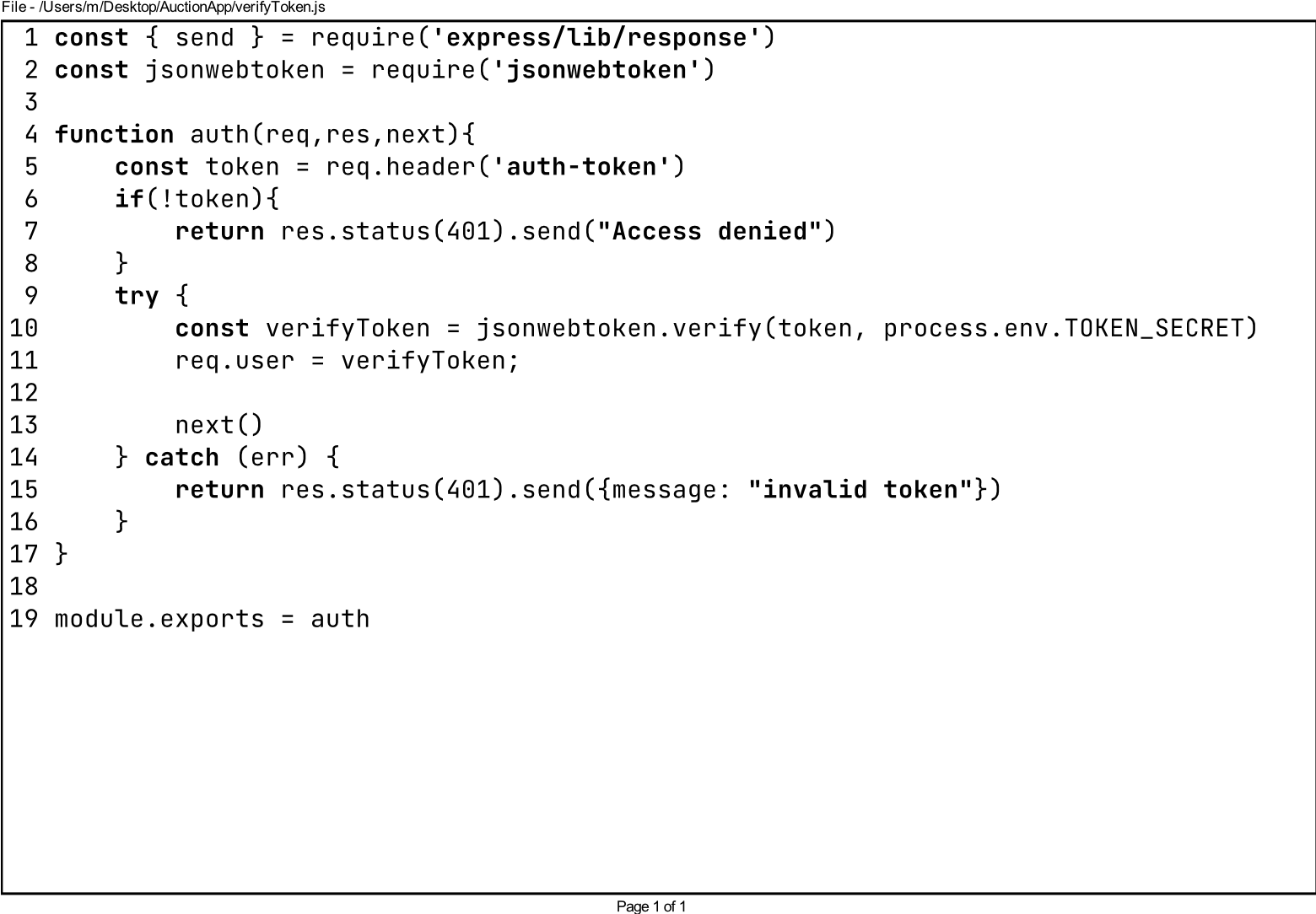
}

## JWT Authentication

To allow only registered users to interact with MiniBid API JWT (jsonwebtoken) is used for MiniBid API.

To obtain token user must be registered and logged in. User auth-token is issued when user sends login details, see routes/auth.js line 62-66.

Code in verifyToken.js will read HTTP header and looks for auth-token. If token is present(u) user is authorised to post or consume data, otherwise error message will be displayed.



API Endpoints and application flow MiniBid application flow:

Register

Login

Access MiniBid

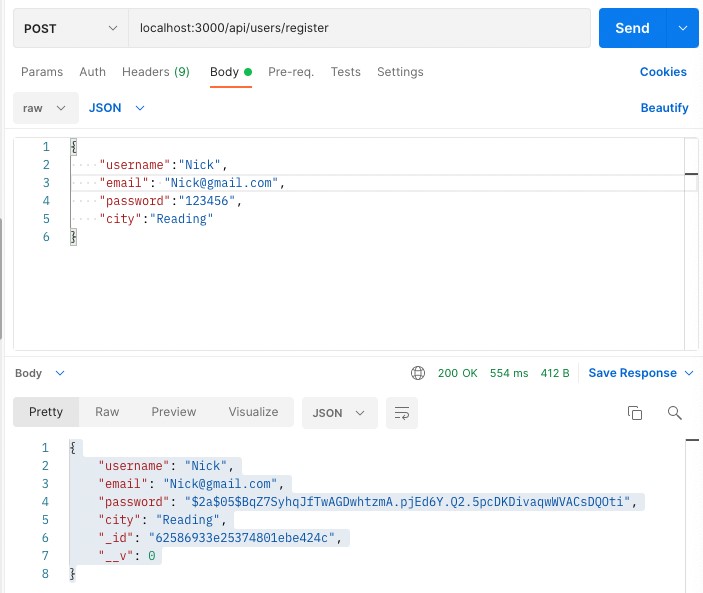
resources

Endpoints:

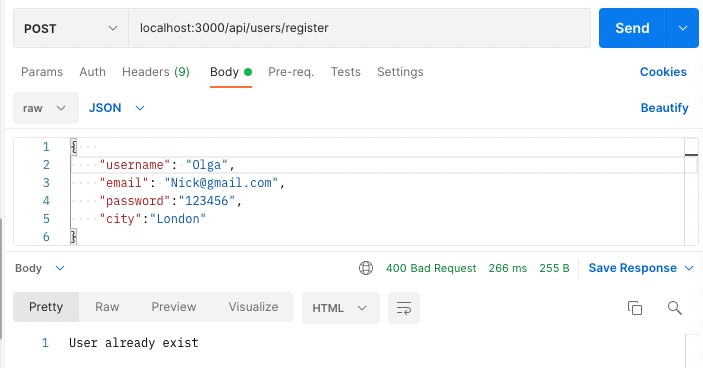
1. **POST/ User registration - localhost:3000/api/users/register**
2. **POST/ User Login - localhost:3000/api/users/login**
3. **GET/ All registered users- localhost:3000/api/users/login**
4. **POST/ Create item using user id- localhost:3000/api/items/:id**
5. **GET / All Items - localhost:3000/api/items**
6. **GET/ User Items using user id - localhost:3000/api/items/:id**
7. **POST/ Add item to the auction - localhost:3000/api/auctions/add**
8. **GET/ Close expired auctions - localhost:3000/api/auctions/close**
9. **GET/ Active auctions - localhost:3000/api/auctions/active**
10. **GET/ Active auction by auction id - localhost:3000/api/auctions/active/:id 11. GET/ Closed auctions with sold items- localhost:3000/api/auctions/sold**
11. **GET/ Closed auction with sold item - localhost:3000/api/auctions/sold/:id**
12. **POST /Make a bid using auction id - localhost:3000/api/bids/makeBid/:id**
13. **GET / All bids - localhost:3000/api/bids**

## Testing

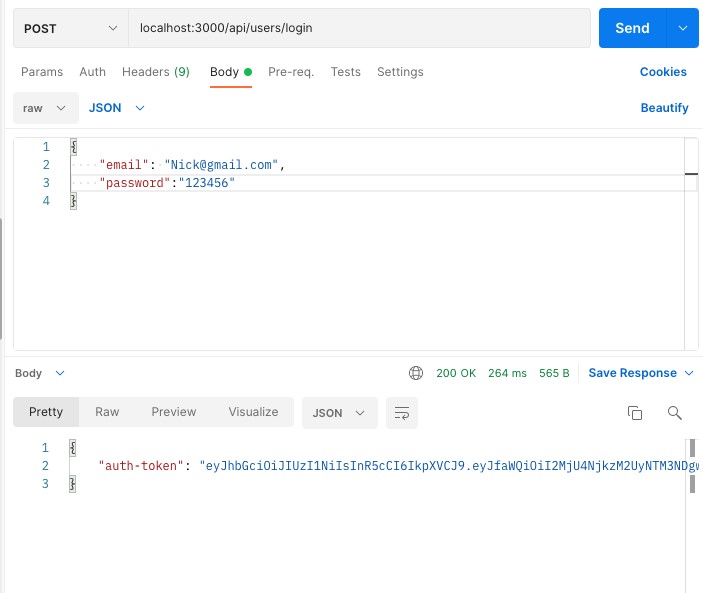
1. User registration



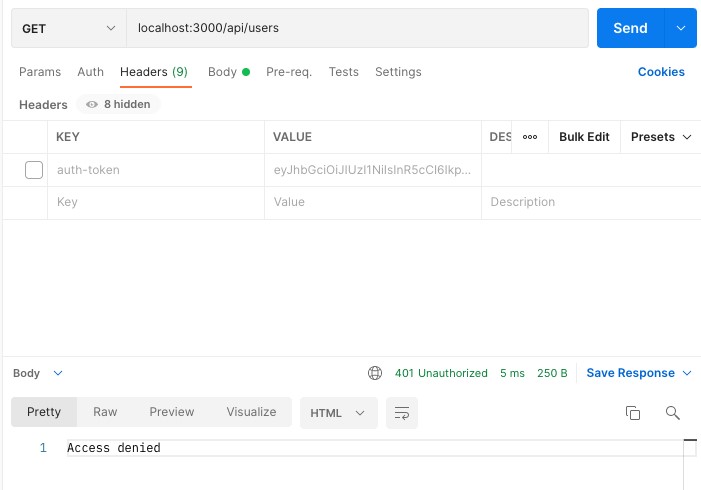
1. User register Validation

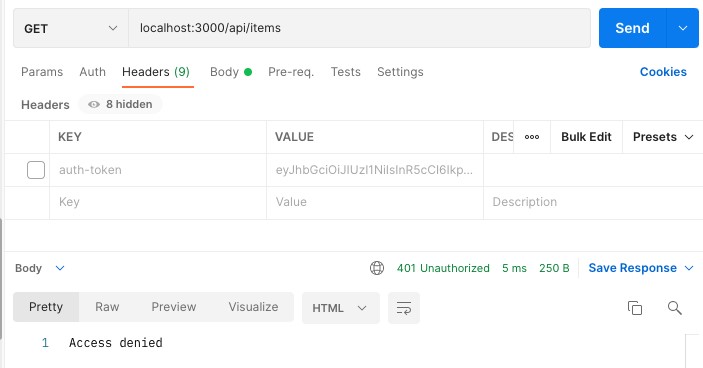


1. User Login

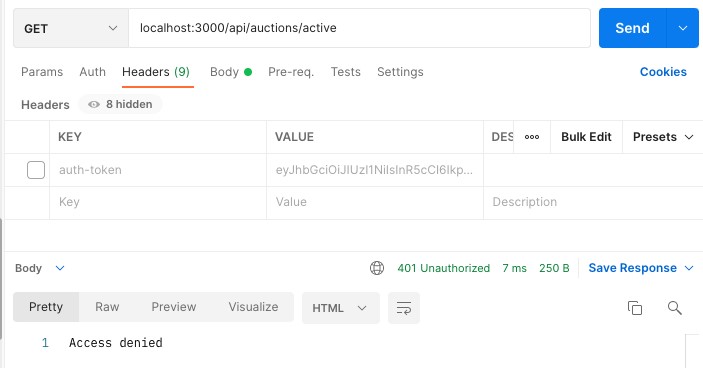


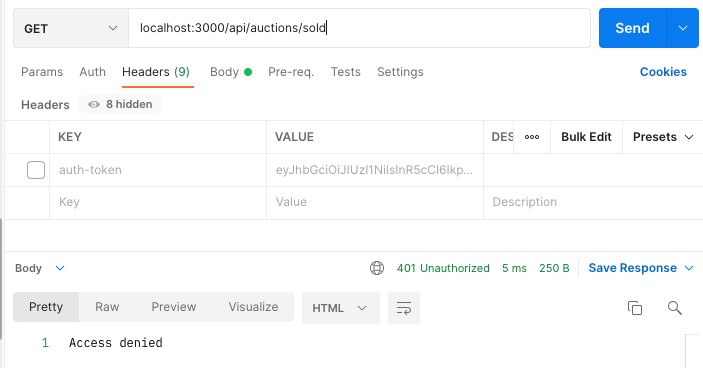
1. User tries accessing resources without token

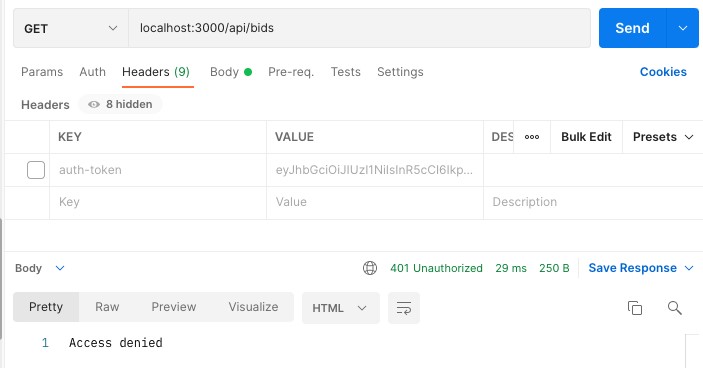




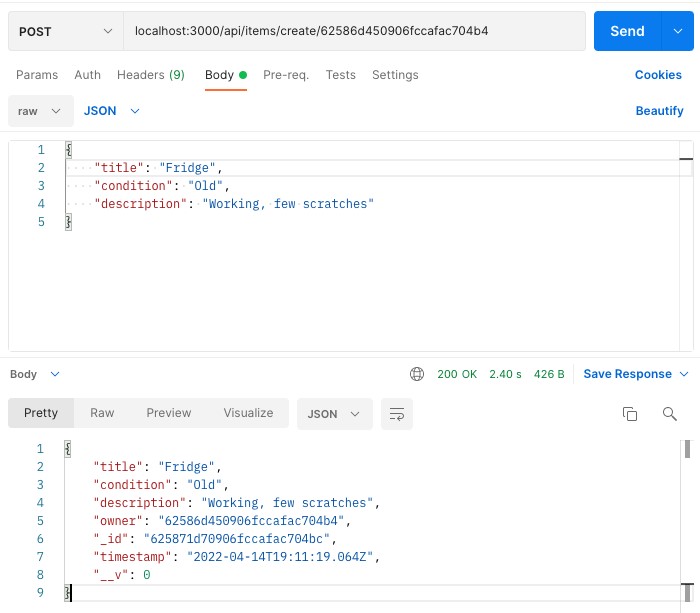




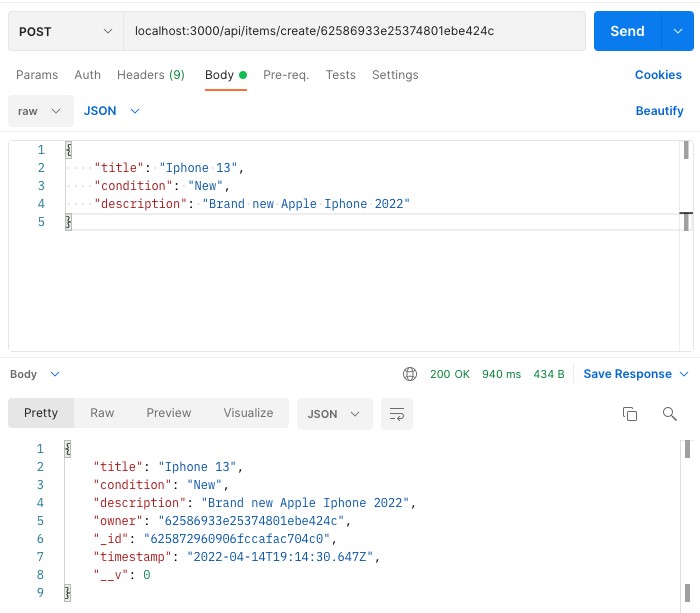




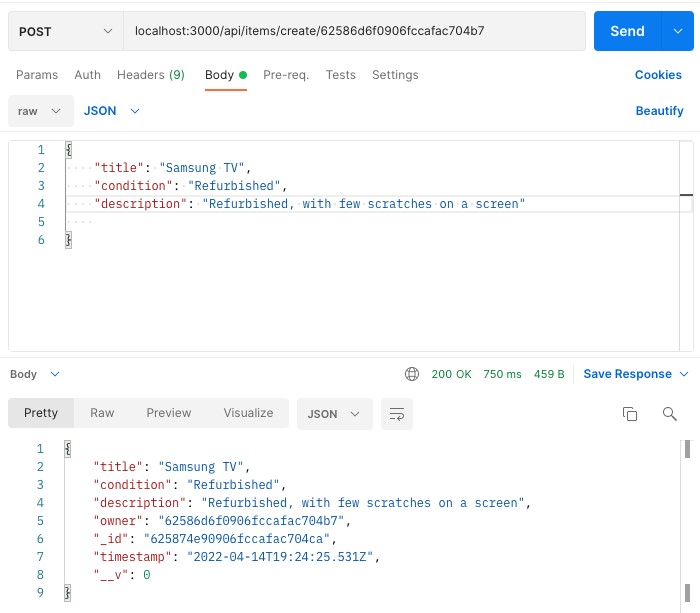
1. Olga creates item using her ID



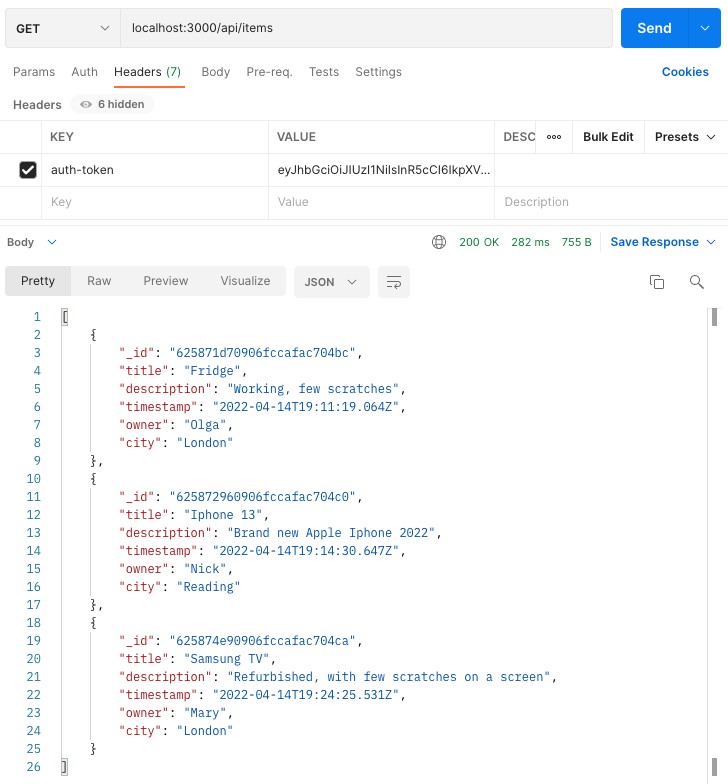
1. Nick creates Item using her ID



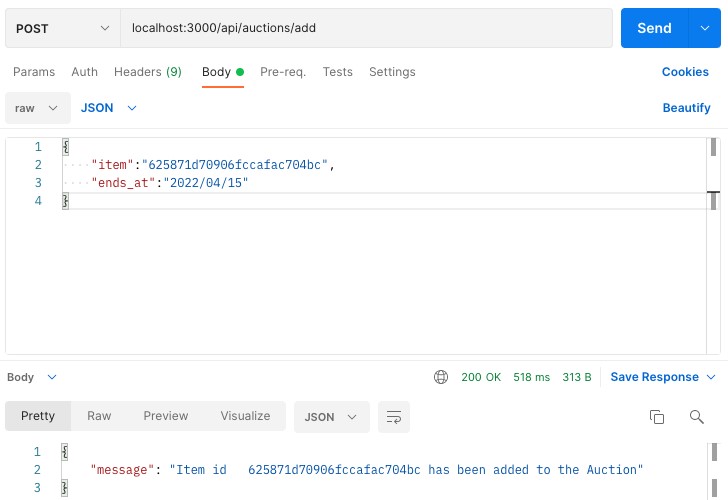
Mary creates item using her ID



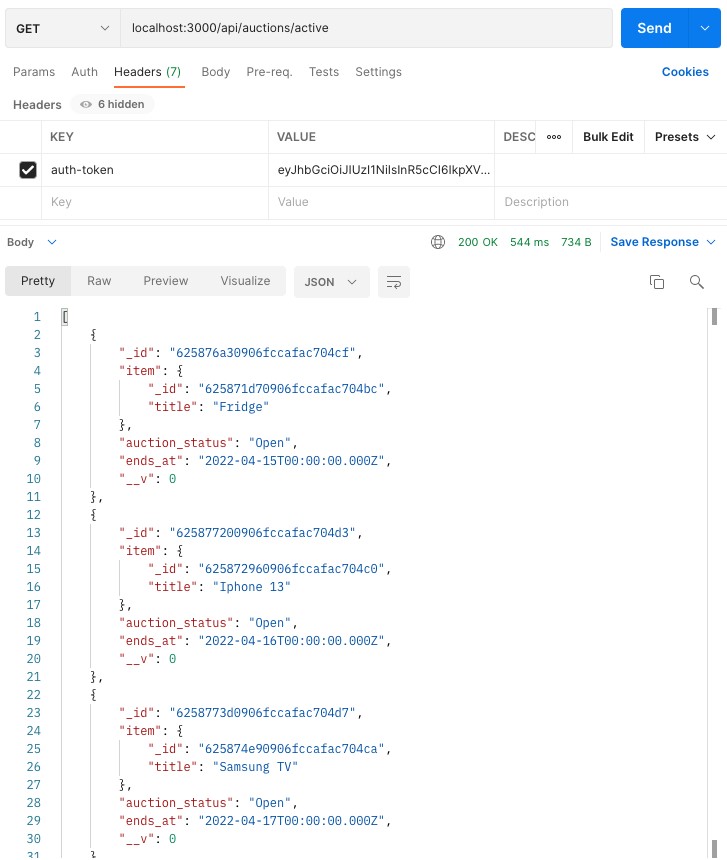
8. There are three items created



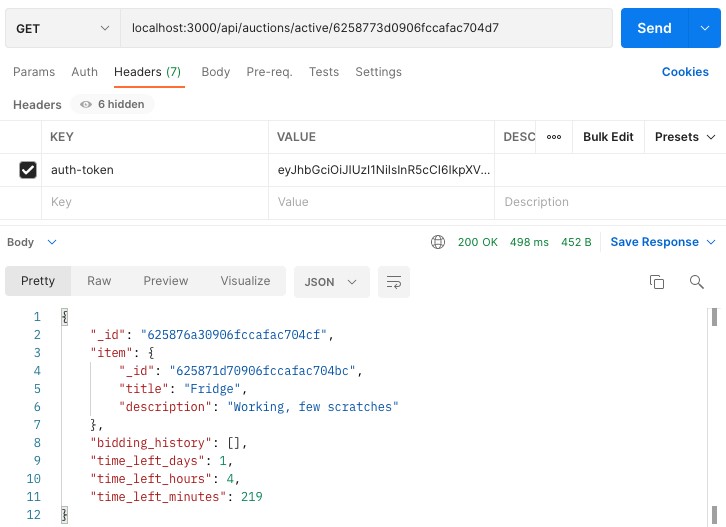
Users add item to auction using item id



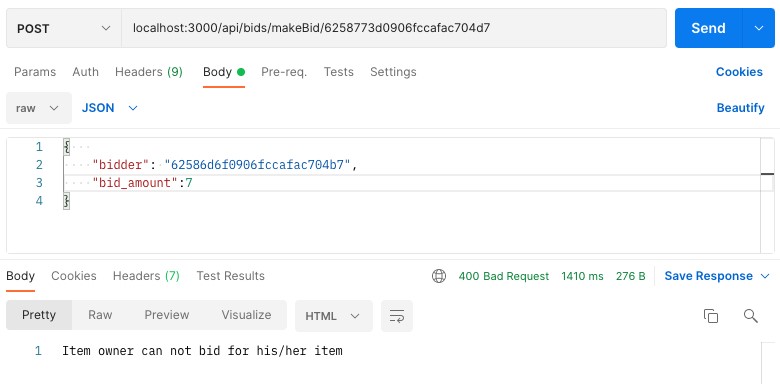
After users adds items to the auction using expiration date, there are 3 auctions with three unique items



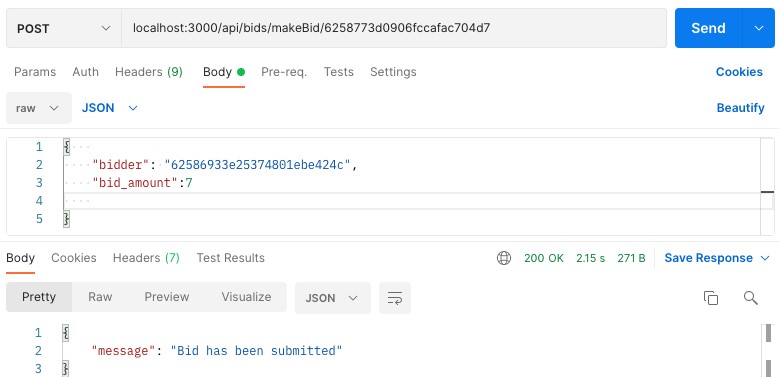
Nick and Olga get the details of Marys item



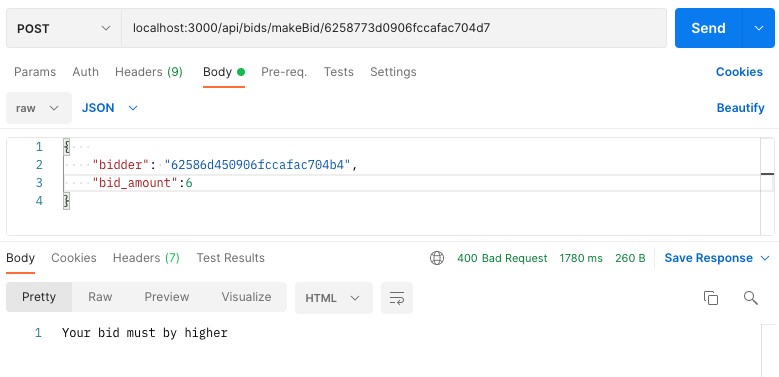
12. Mary bids for her item in a auction using her ID and bid\_amount:



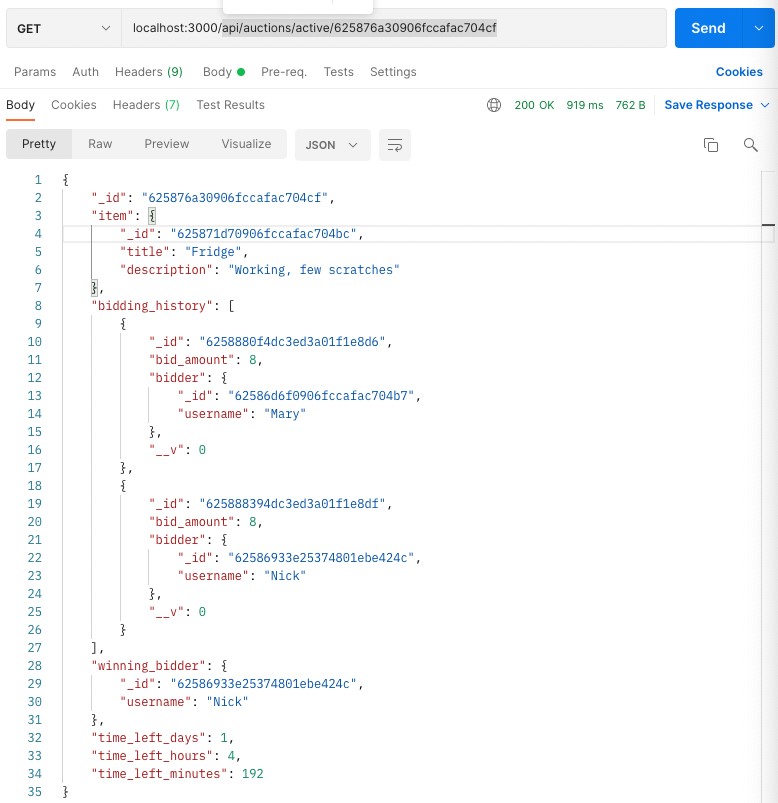
Other users bid for an item



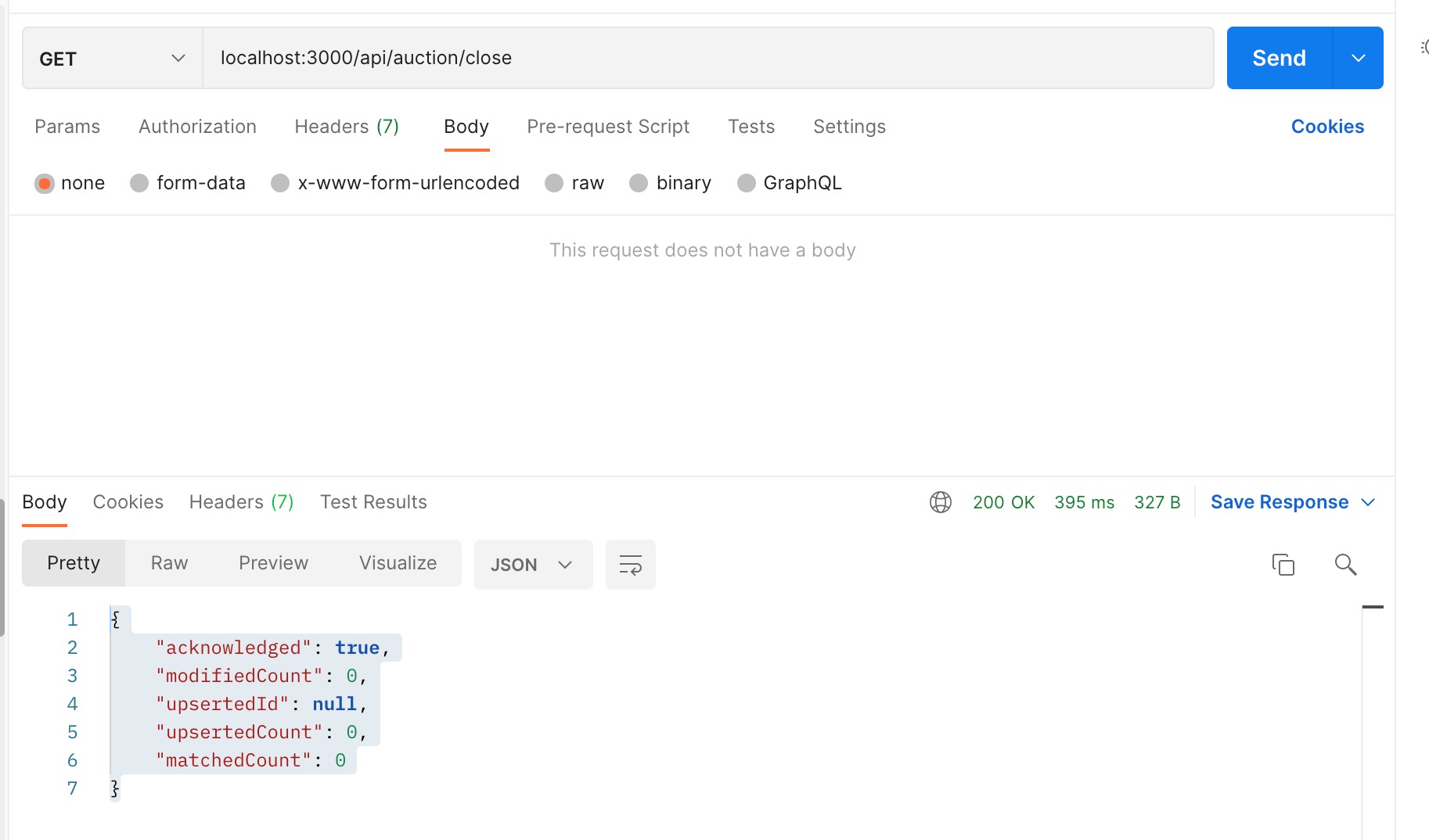
14 .User tries to post a bid that is less than the last bid



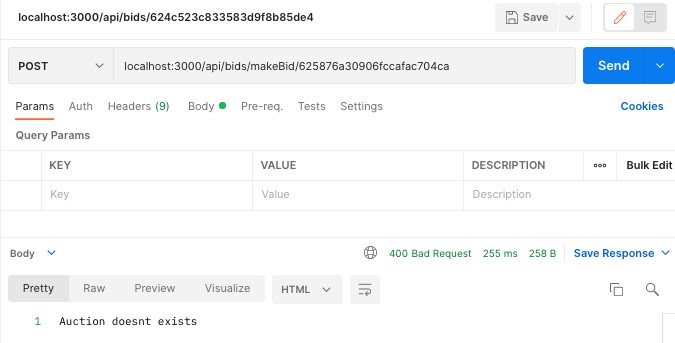
Bidders submit their bids to the active auction:



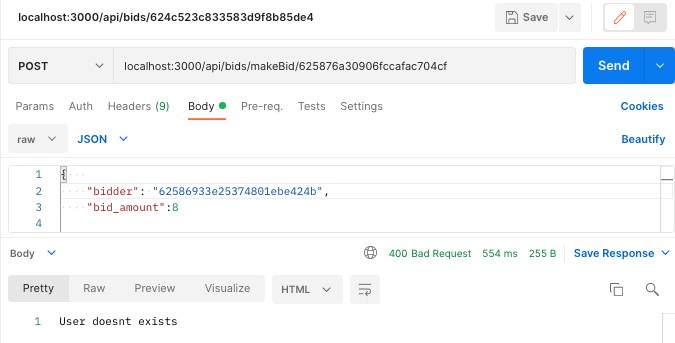
Close Items that are expired. If there are no expired items keys matchedCount and modifiedCount will have zero values, else values will indicate how many auctions has been closed.



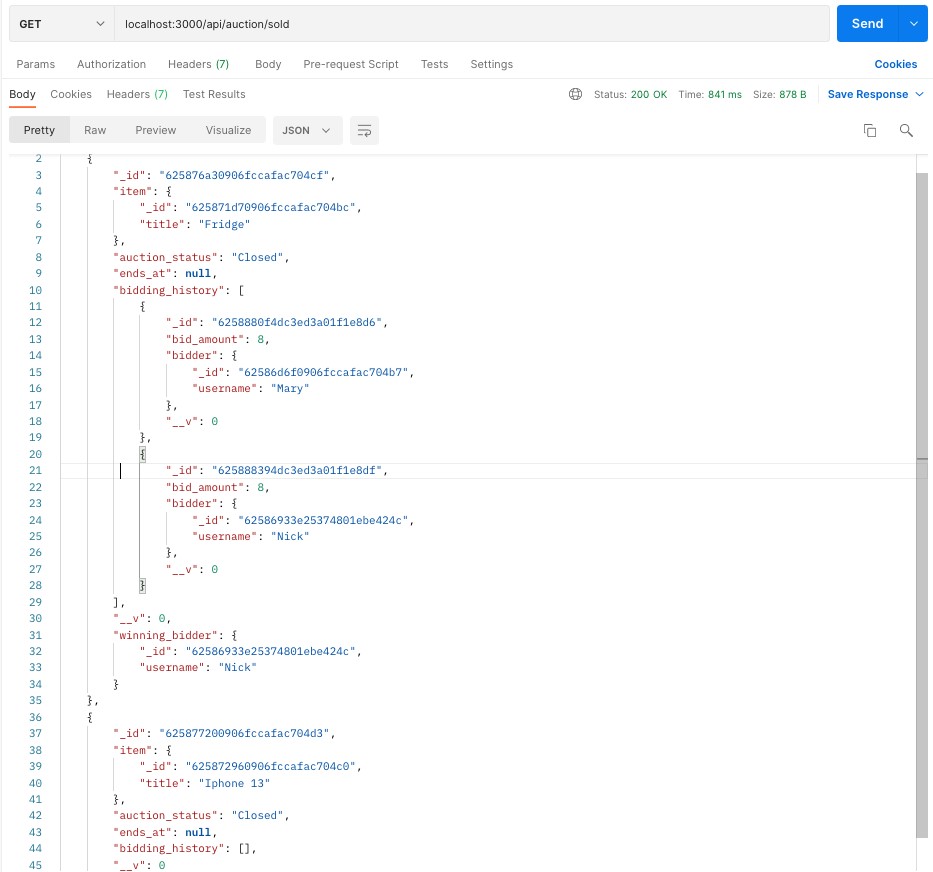
1. User bids for auction that doesn’t exists



1. Bidder (user) doesn’t exist

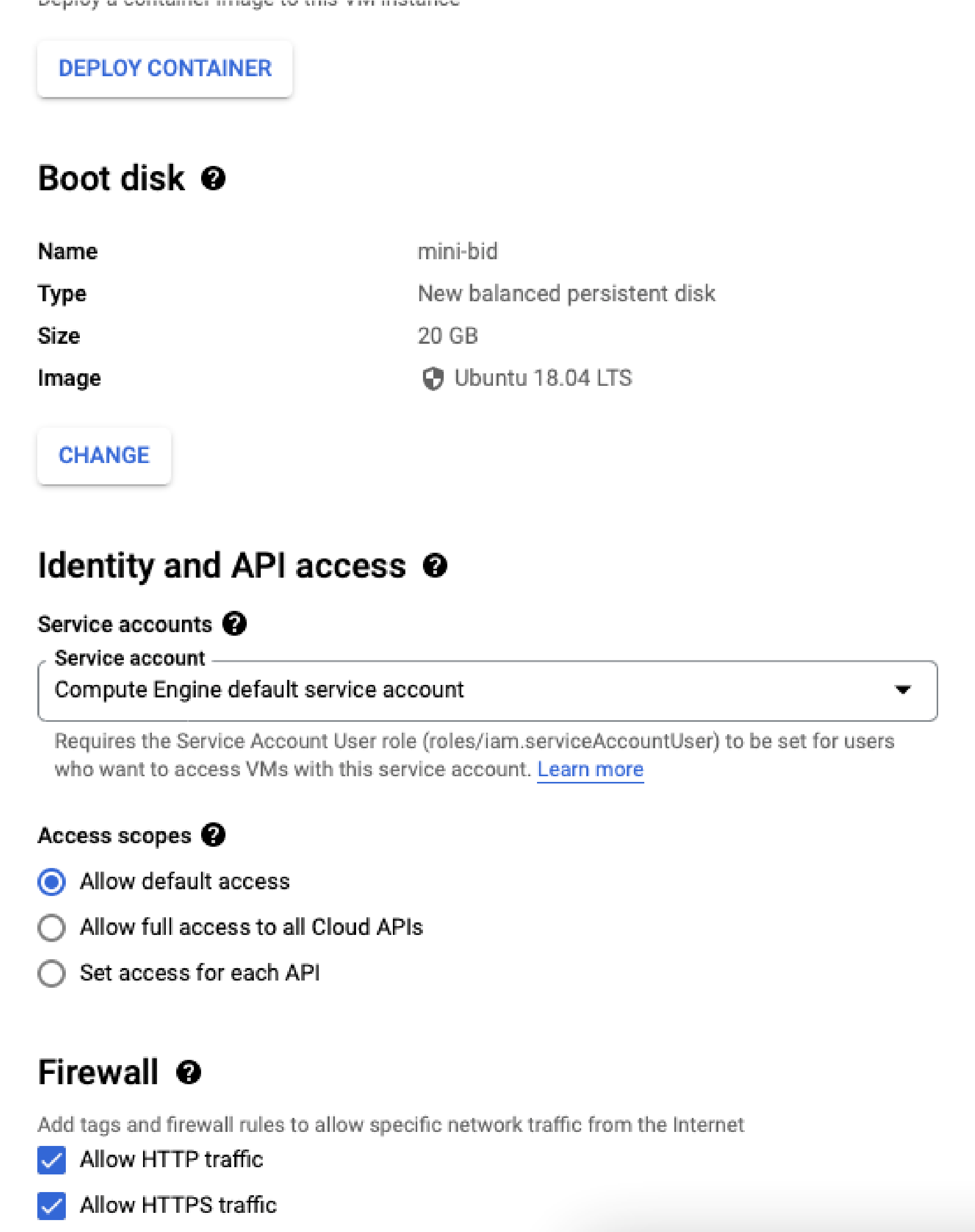


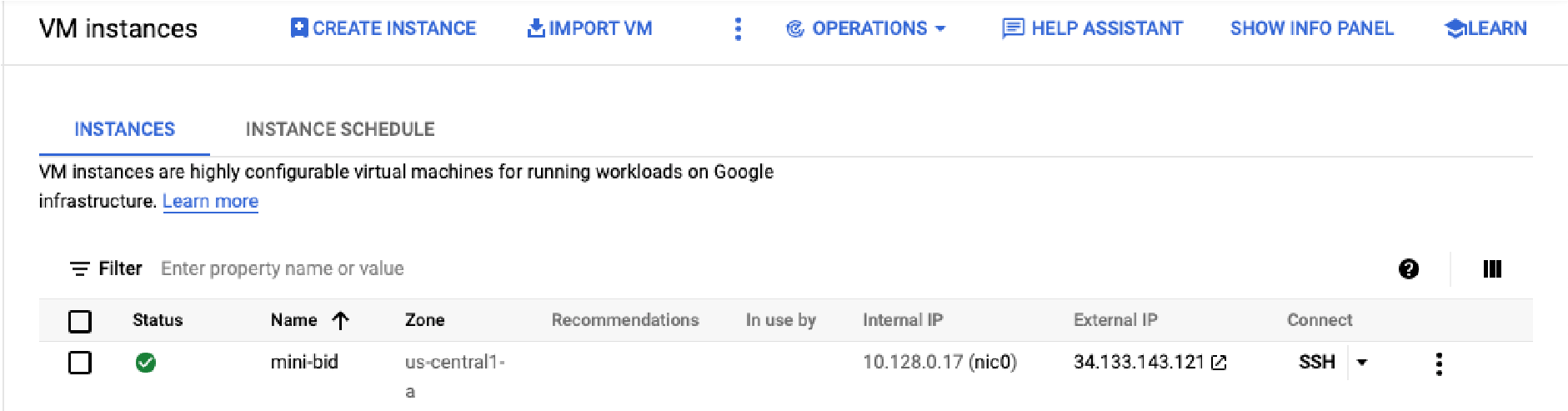
1. After expired auctions has been closed, auctions status is changed from open, to closed.



### Deploy your MiniBid project into a GCP VM using Docker

1 Create VM in GCP





1. Open SSH connection

1. Run command to update system

sudo apt-get update

1. Install docker

sudo apt-get install docker.io

1. Check Docker version

Docker version 20.10.7, build 20.10.7-0ubuntu5~18.04.3

1. Create docker user

sudo adduser docker-user

1. Add user to sudo group

sudo usermod -aG sudo docker-user

1. Give sudo permisions to docker

sudo usermod -aG docker docker-user

1. Switch user

su - docker-user

1. Clone minibid repository from github

git clone --branch master

https://pranaitism:ghp\_LvxGA3zJ6003RoqfJgp0oMWRfU9myM0lufcl@github.com/pranaitis m/minibid.git

1. cd into the cloned repo:

cd minibid

12. Create Docker file using pico

*pico Dockerfile*

and ddd following code:

*FROM alpine*

*RUN apk add --update nodejs npm*

*COPY . /src*

*WORKDIR /src*

*COPY package\*.json ./*

*RUN npm install*

*EXPOSE 3000*

*ENTRYPOINT ["node", "./app.js"]*

*Conrol+s* to save, *control+x* to exit.

1. Build an image called minibid-image:1

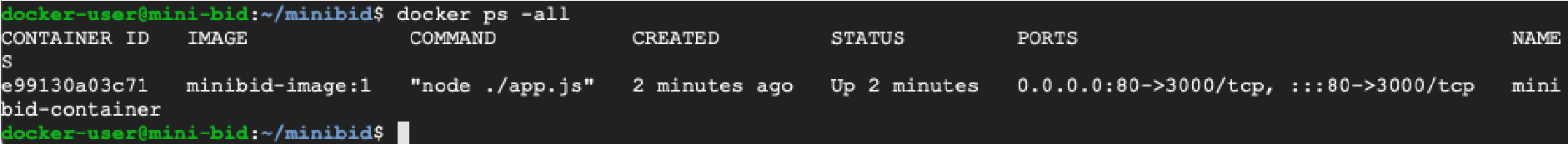
*docker image build -t minibid-image:1 .*

1. Run the container minibid-container using the minibid-image image.

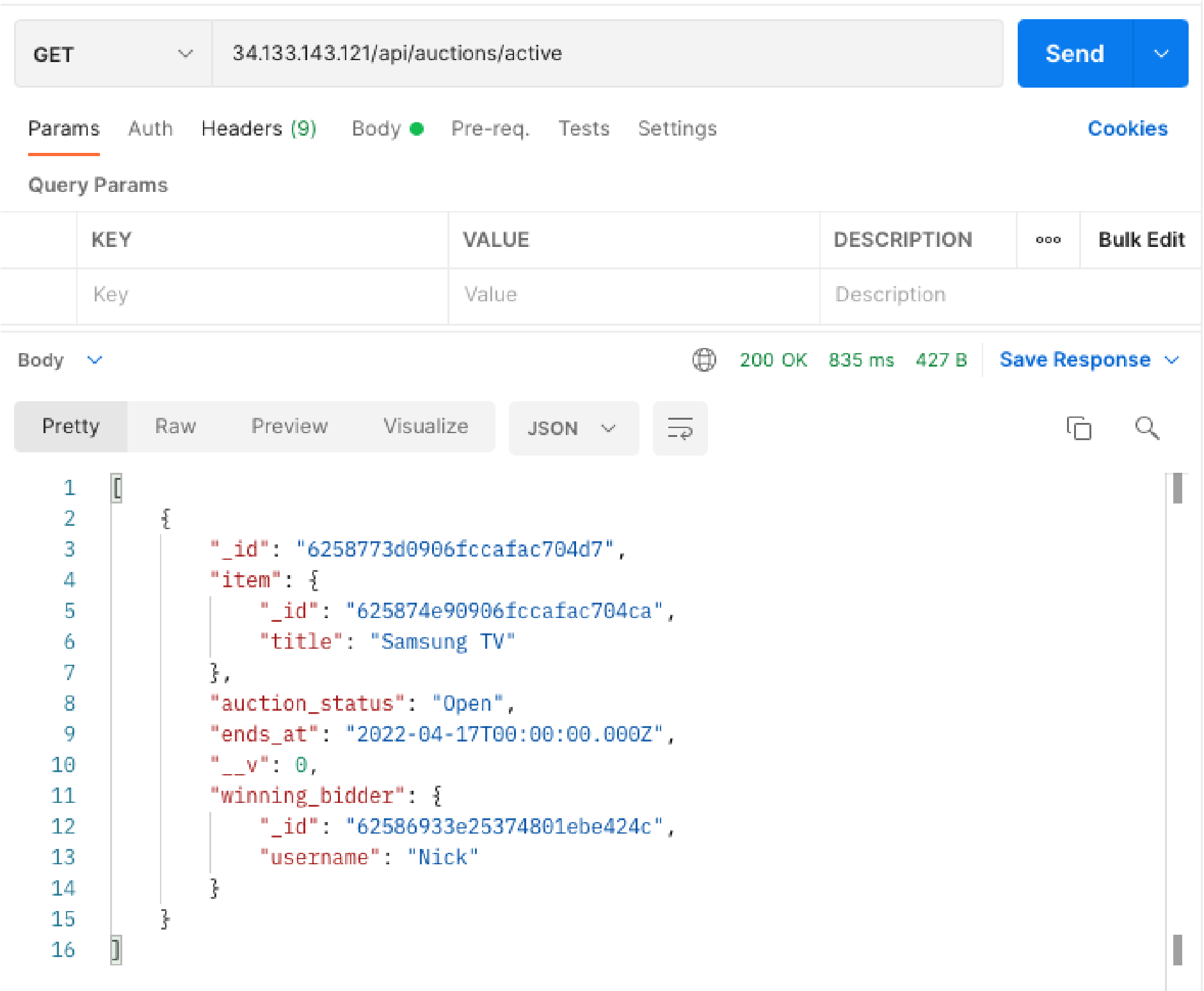
*docker container run -d --name minibid-container --publish 80:3000 minibid-image:1*

1. Display running containers

*docker ps -all*



Postman test



### Future improvements

1. Change endpoint naming’s
2. Allow users to see their items
3. Categorise items
4. Real time auction
5. Add Bid max price limit message
6. Add timestamp for Bid
7. Implement script to close expired auctions functionality
8. Develop front end
9. Allow users to update, delete auctions and items.
10. Email notification for the users

### Resources

MongoDb - https://www.mongodb.com

Mongoose - https://mongoosejs.com

ExpressJS - https://expressjs.com NodeJS - https://nodejs.org/en/ Cloud computing Concepts lab slides.

Reference:

1. https://www.oreilly.com/library/view/mongodb-applied-design/9781449340056/ch01.html
2. https://stackoverflow.com/questions/8990595/dates-in-mongoose
3. Auction.js, line 34 - https://stackoverflow.com/questions/8990595/dates-inmongoose
4. routes/bid.js line93-95 https://stackoverflow.com/questions/59617836/nodejsmongodb-return-a-json-response-from-aggregate-inside-a-get-request

Acknowledgments:

I Acknowledge that MiniBid is work of my own. Code for user management is taken from lab tutorials. Work that has been used is referenced.

**The Academic Declaration should read as follows: “I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta-searching software.”**