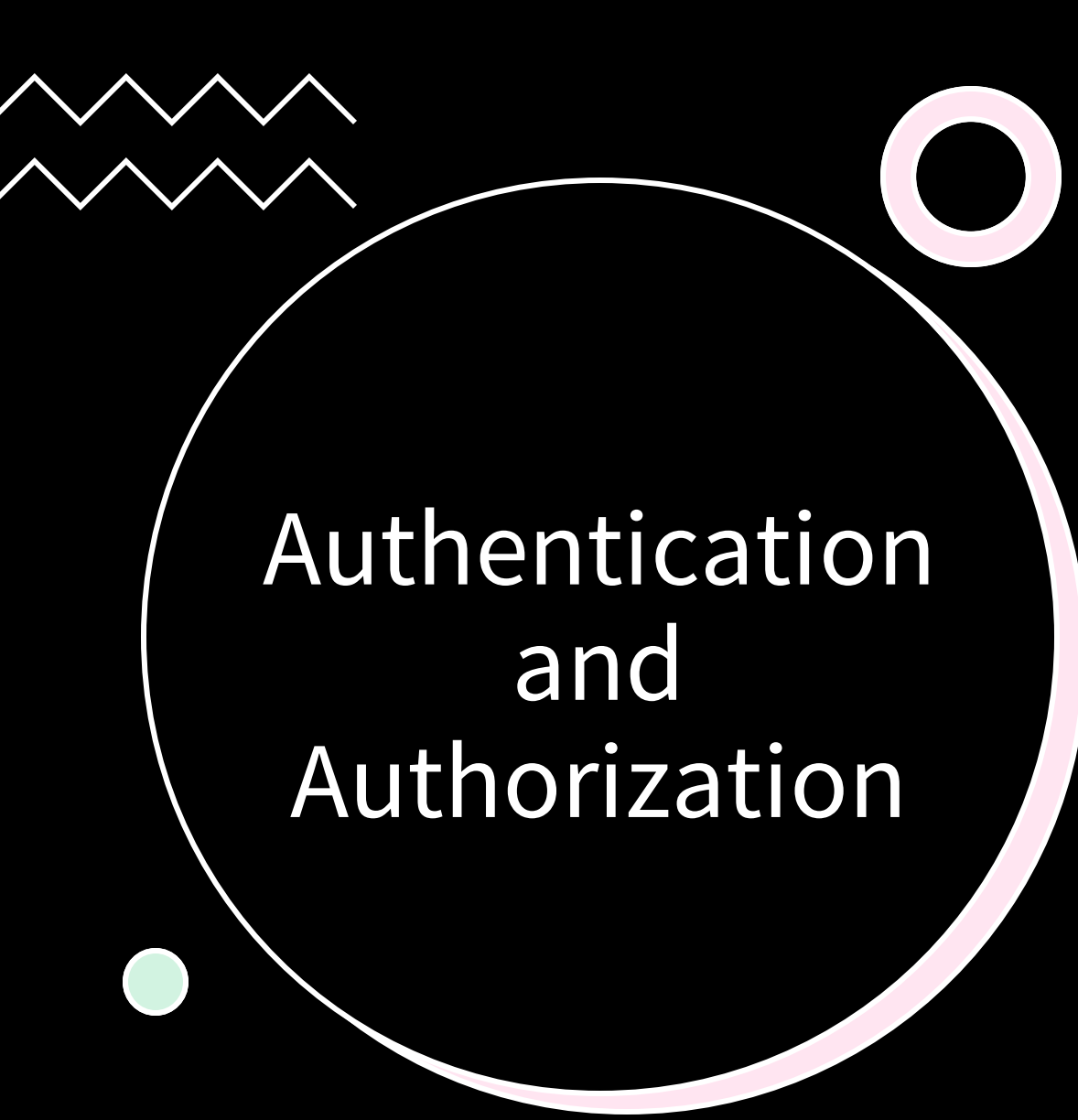



# AUTHENTICATION AND AUTHORIZATION





# Authentication and Authorization

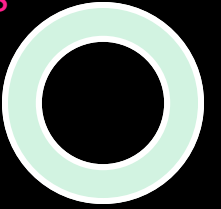
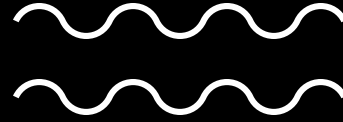
- Authentication is the process of determining if the user is who he/she claims to be. It involves validating their email/password.
  - Authorization is the process of determining if the user has permission to perform a given operation.
- 

```
const Joi = require('joi');
const mongoose = require('mongoose');
const User = mongoose.model('User', new mongoose.Schema({
  name: {
    type: String,
    required: true,
    minlength: 5,
    maxlength: 50
  },
  email: {
    type: String,
    required: true,
    minlength: 5,
    maxlength: 255,
    unique: true
  },
  password: {
    type: String,
    required: true,
    minlength: 5,
    maxlength: 1024
  },
}));
```

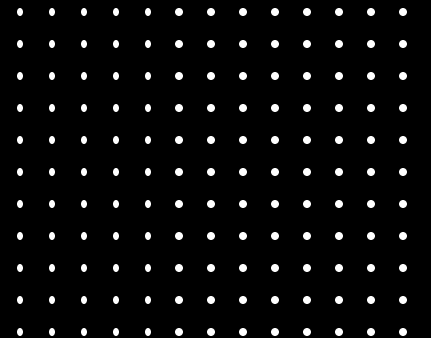
```
function validateUser(user) {
  const schema = Joi.object({
    name: Joi.string().min(5).max(50).required(),
    email: Joi.string().min(5).max(255).required().email(),
    password: Joi.string().min(5).max(255).required()
  });
  return schema.validate(user);
}
```

```
exports.User = User;
exports.validate = validateUser;
```

- Create file **user.js** in the folder **models**

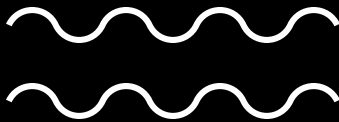
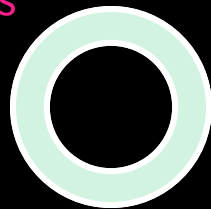


# Creating the User Model



users.js

- Create file **users.js** in the folder **routes**



```
const {User, validate} = require('../models/user');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();

router.post('/', async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);

  let user = await User.findOne({ email: req.body.email });
  if (user) return res.status(400).send('User already registred.');

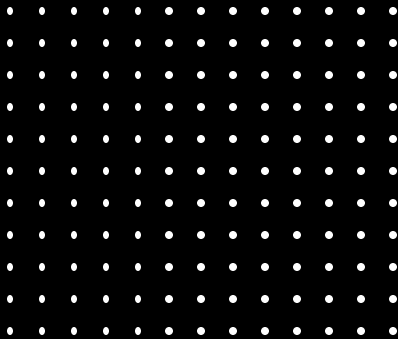
  user = new User({
    name: req.body.name,
    email: req. body.email,
    password: req.body.password
  });

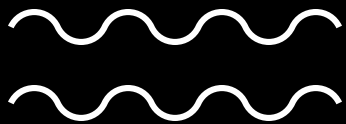
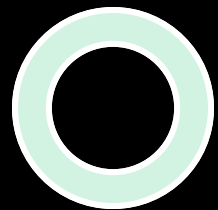
  await user.save();

  res.send(user);
});

module.exports = router;
```

# Registering Users





```
const mongoose = require('mongoose');
const genres = require('./routes/genres');
const customers = require('./routes/customers');
const movies = require('./routes/movies');
const rentals = require('./routes/rentals');
const users = require('./routes/users');
const express = require('express');
const app = express();

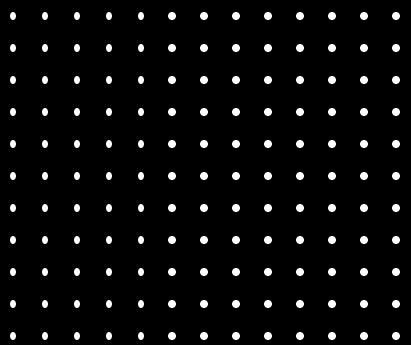
mongoose.connect('mongodb://localhost/videohost')
  .then(() => console.log('Connected to MongoDB...'))
  .catch(err => console.error('Could not connect to MongoDB...'));

app.use(express.json());
app.use('/api/genres', genres);
app.use('/api/customers', customers);
app.use('/api/movies', movies);
app.use('/api/rentals', rentals);
app.use('/api/users', users);

const port = process.env.PORT || 3000;
app.listen(port, () => console.log(`Listening on port ${port}...`));
```

Insert  
code

# Registering Users



# ● Registering Users



POSTMAN

The screenshot shows the Postman interface for a POST request to `http://localhost:3000/api/users`. The request body is a JSON object:

```
1 {  
2   "name": "mari",  
3   "email": "marizza",  
4   "password": "123321"  
5 }
```

The response is a 400 Bad Request error with the message: `1 "name" length must be at least 5 characters long`.



# ● Registering Users



POSTMAN

The screenshot shows the Postman interface for a POST request to `http://localhost:3000/api/users`. The request body is a JSON object: `{ "name": "marizza", "email": "marizza", "password": "123321" }`. The response is a `400 Bad Request` with the message `"email" must be a valid email`. Orange boxes and arrows highlight the method, URL, body tab, raw JSON format, the body content, the error status, and the error message.

POST `http://localhost:3000/api/users` **Send**

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** Beautify

```
1 {
2   "name": "marizza",
3   "email": "marizza",
4   "password": "123321"
5 }
```

Body Cookies Headers (7) Test Results **400 Bad Request** 13 ms 266 B **Save Response**

Pretty Raw Preview Visualize HTML `"email" must be a valid email`



# ● Registering Users



POSTMAN

The screenshot displays the Postman interface for a REST client. At the top, the request method is set to **POST** and the URL is `http://localhost:3000/api/users`. The **Body** tab is selected, showing the request payload in **raw** **JSON** format:

```
1 {
2   "name": "marizza",
3   "email": "marizza@gmail.com",
4   "password": "123321"
5 }
```

The **Send** button is highlighted with an orange arrow. Below the request, the response section shows a **200 OK** status, a response time of **33 ms**, and a size of **343 B**. The response body is displayed in **Pretty** **JSON** format:

```
1 {
2   "name": "marizza",
3   "email": "marizza@gmail.com",
4   "password": "123321",
5   "_id": "61445018310ad8772d5435fe",
6   "__v": 0
7 }
```

Orange boxes highlight the request method, URL, body tab, raw JSON format, request body, status code, and response body. A 'Beautify' button is also visible on the right side of the interface.





# ● Registering Users



POSTMAN

The screenshot shows the Postman interface for a POST request to `http://localhost:3000/api/users`. The request body is a JSON object: `{ "name": "marizza", "email": "marizza@gmail.com", "password": "123321" }`. The response is a `400 Bad Request` with the message `User already registred.`. Several elements are highlighted with orange boxes: the POST method and URL, the Body tab, the raw JSON body, the JSON format dropdown, the 400 Bad Request status, and the response message.

POST `http://localhost:3000/api/users` **Send**

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   "name": "marizza",
3   "email": "marizza@gmail.com",
4   "password": "123321"
5 }
```

**Body** Cookies Headers (7) Test Results **400 Bad Request** 10 ms 259 B **Save Response**

Pretty Raw Preview Visualize HTML

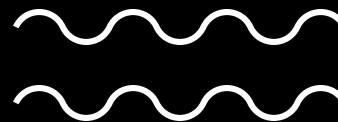
```
1 User already registred.
```



## Terminal

`npm i lodash`

- Modify file `users.js` in the folder `routes`



`users.js`

```
const _ = require('lodash');
const {User, validate} = require('../models/user');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();

router.post('/', async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);

  let user = await User.findOne({ email: req.body.email });
  if (user) return res.status(400).send('User already registred.');

  user = new User(_.pick(req.body, ['name', 'email', 'password']));
  await user.save();

  res.send(_.pick(user, ['_id', 'name', 'email']));
});

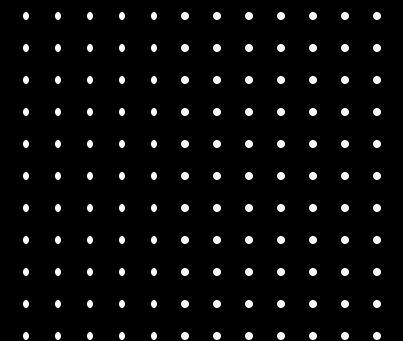
module.exports = router;
```

Insert  
code

Modify  
code

# Using Lodash

[more information  
about lodash](#)



# ● Using Lodash



POSTMAN

The screenshot displays the Postman interface for a POST request to `http://localhost:3000/api/users`. The request body is a JSON object with the following fields:

```
{
  "name": "marizza",
  "email": "marizza555@gmail.com",
  "password": "123321"
}
```

The response status is `200 OK` with a response time of 48 ms and a size of 317 B. The response body is a JSON object with the following fields:

```
{
  "_id": "614454b6650054d6e60cf762",
  "name": "marizza",
  "email": "marizza555@gmail.com"
}
```

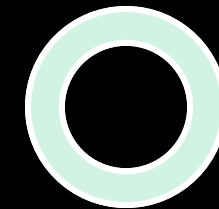
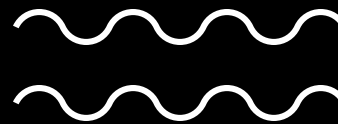
Red boxes highlight the POST method, the URL, the Body tab, the raw JSON input, the JSON output format, the 200 OK status, and the response body.



## Terminal

`npm i bcrypt`

- Create file `hash.js`



## hash.js

```
const bcrypt = require('bcrypt');

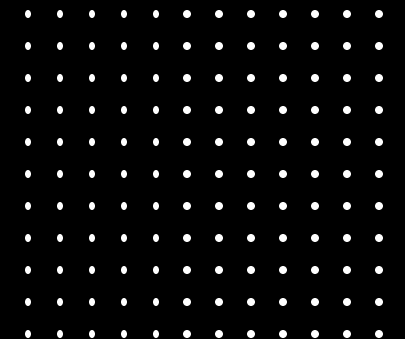
async function run() {
  const salt = await bcrypt.genSalt(10);
  console.log(salt);
}

run();
```

# Hashing Passwords

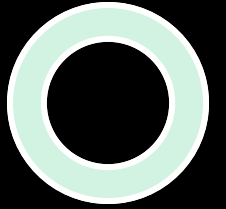
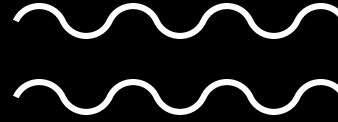
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\NodeJS_Course\Authentication_and_Authorization> node hash.js
$2b$10$dVmIUJ2ml8Ux20xD069FG0
PS C:\NodeJS_Course\Authentication_and_Authorization> █
```



hash.js

- Modify file `hash.js`



```
const bcrypt = require('bcrypt');

async function run() {
  const salt = await bcrypt.genSalt(10);
  const hashed = await bcrypt.hash('1234', salt);
  console.log(salt);
  console.log(hashed);
}

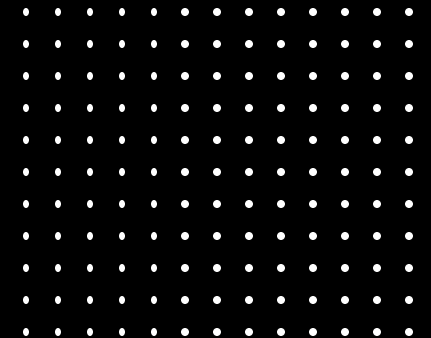
run();
```

Insert  
code

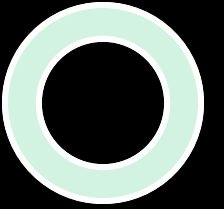
# Hashing Passwords

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\NodeJS_Course\Authentication_and_Authorization> node hash.js
$2b$10$CV8Ten4ZsDLI0XC0Tf8An.
$2b$10$CV8Ten4ZsDLI0XC0Tf8An.qvMN2a6KmGSfrawSxMi/NE5mWpN3Jum
PS C:\NodeJS_Course\Authentication_and_Authorization> █
```



- Modify file `users.js`



`users.js`

```
const bcrypt = require('bcrypt');
const _ = require('lodash');
const {User, validate} = require('../models/user');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();

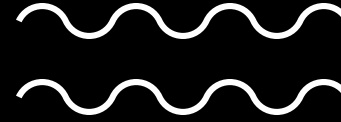
router.post('/', async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);

  let user = await User.findOne({ email: req.body.email });
  if (user) return res.status(400).send('User already registred.');

  user = new User(_.pick(req.body, ['name', 'email', 'password']));
  const salt = await bcrypt.genSalt(10);
  user.password = await bcrypt.hash(user.password, salt);
  await user.save();

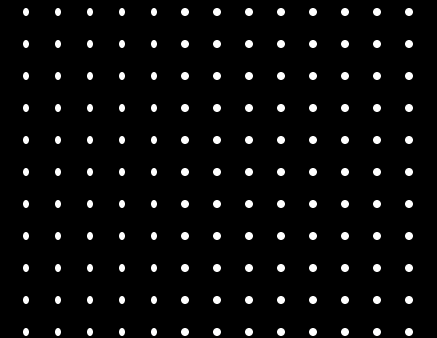
  res.send(_.pick(user, ['_id', 'name', 'email']));
});

module.exports = router;
```

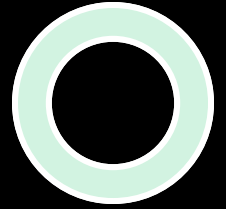
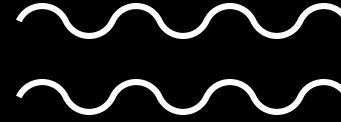


Insert  
code

# Hashing Passwords



Remove **users** in the MongoDB Compass

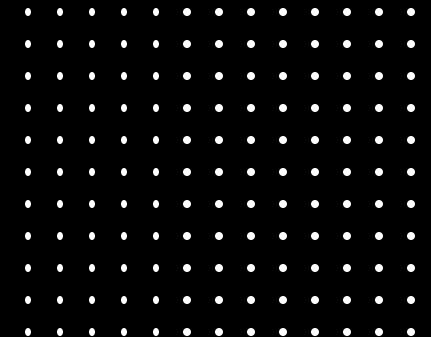


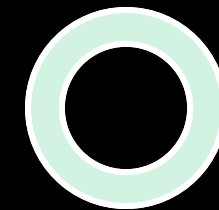
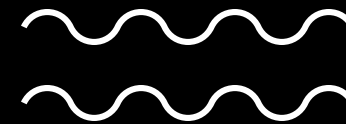
# Hashing Passwords

## Drop Collection

⚠ To drop **users** type the collection name **users**.

CANCELDROP COLLECTION





# Hashing Passwords

videohost.users

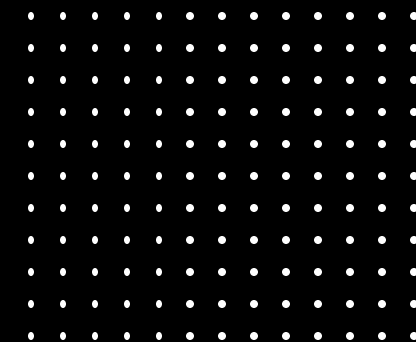
	DOCUMENTS	1	TOTAL SIZE	153B	AVG. SIZE	153B
	INDEXES	1	TOTAL SIZE	20.0KB	AVG. SIZE	20.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

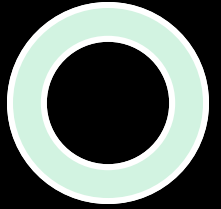
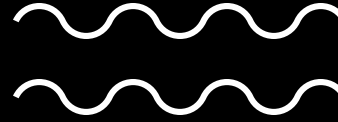
**FILTER** { field: 'value' } **OPTIONS** **FIND** **RESET** **↺** **⋮**

**ADD DATA** **VIEW** **⋮** **{ }** **⌂** Displaying documents 1 - 1 of 1 **<** **>** **REFRESH**

```
_id: ObjectId("61445f72d86aad655c39ad44")
name: "marizza"
email: "marizza@gmail.com"
password: "$2b$10$AEG8feaJ6Ru1EeTFSKy2t./Kv5LxeIT/.eDY1zPOiopKdsISJlk6"
__v: 0
```







POSTMAN

# Hashing Passwords

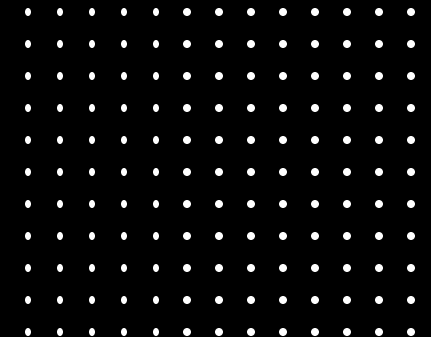
The screenshot shows the Postman interface with a POST request to `http://localhost:3000/api/users`. The request body is a JSON object with the following fields:

```
{  "name": "marizza",  "email": "marizza@gmail.com",  "password": "123321"}
```

The response is a 200 OK status with a response time of 94 ms and a body size of 314 B. The response body is a JSON object with the following fields:

```
{  "_id": "61445f72d86aad655c39ad44",  "name": "marizza",  "email": "marizza@gmail.com"}
```

Orange boxes highlight the request URL, the 'Body' tab, the 'JSON' format, the request body, the response status, and the response body.



auth.js

```
const Joi = require('joi');
const bcrypt = require('bcrypt');
const _ = require('lodash');
const {User} = require('../models/user');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();

router.post('/', async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);

  let user = await User.findOne({ email: req.body.email });
  if (!user) return res.status(400).send('Invalid email or password.');

  const validPassword = await bcrypt.compare(req.body.password, user.password)
  if (!validPassword) return res.status(400).send('Invalid email or password.');

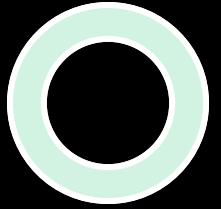
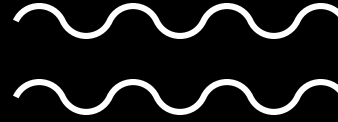
  res.send(true);
});

function validate(req) {
  const schema = Joi.object({
    email: Joi.string().min(5).max(255).required().email(),
    password: Joi.string().min(5).max(255).required()
  });

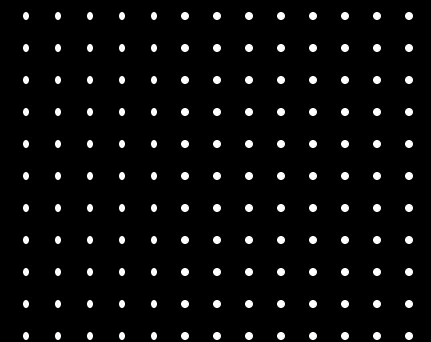
  return schema.validate(req);
}

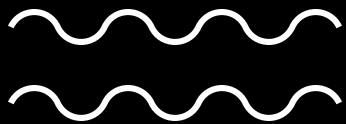
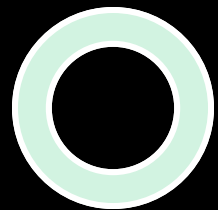
module.exports = router;
```

- Create file **auth.js** in the folder **routes**



# Authenticating Users





```
const mongoose = require('mongoose');
const genres = require('./routes/genres');
const customers = require('./routes/customers');
const movies = require('./routes/movies');
const rentals = require('./routes/rentals');
const users = require('./routes/users')
const auth = require('./routes/auth')
const express = require('express');
const app = express();

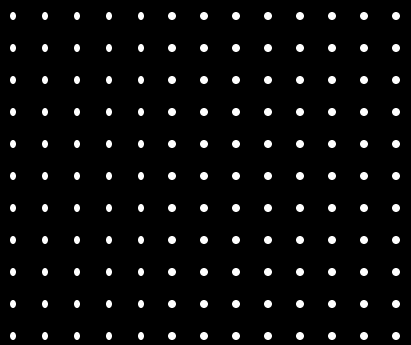
mongoose.connect('mongodb://localhost/videohost')
  .then(() => console.log('Connected to MongoDB...'))
  .catch(err => console.error('Could not connect to MongoDB...'));

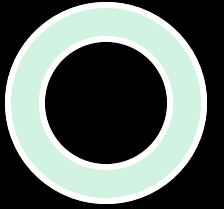
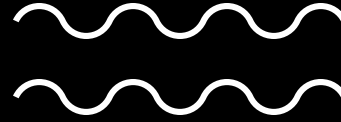
app.use(express.json());
app.use('/api/genres', genres);
app.use('/api/customers', customers);
app.use('/api/movies', movies);
app.use('/api/rentals', rentals);
app.use('/api/users', users);
app.use('/api/auth', auth);

const port = process.env.PORT || 3000;
app.listen(port, () => console.log(`Listening on port ${port}...`));
```

# Authenticating Users

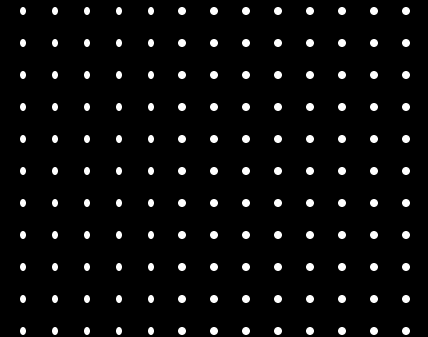
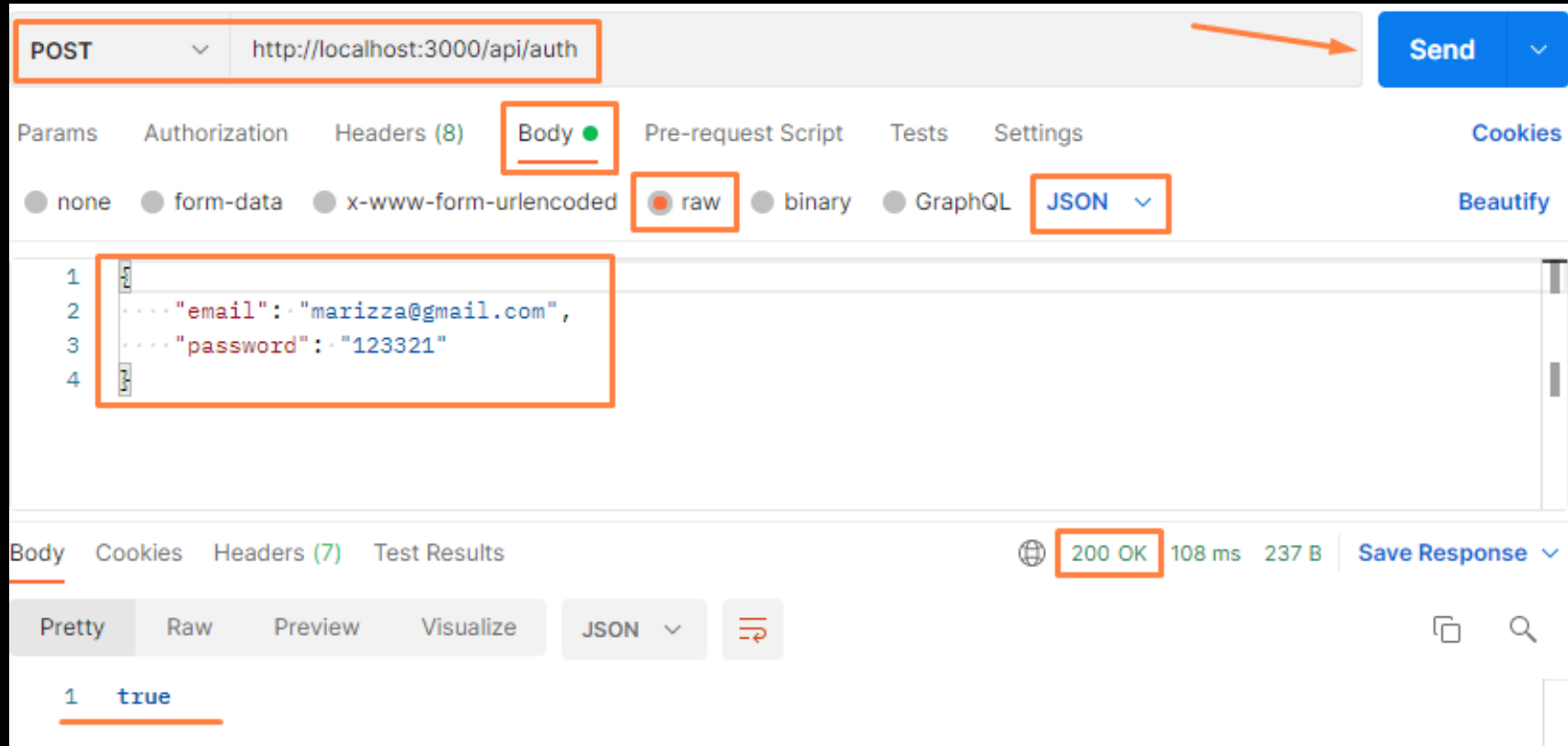
Insert code





POSTMAN

# Testing the Authentication





# JSON Web Tokens

A JSON Web Token (JWT) is a JSON object encoded as a long string. We use them to identify users. It's similar to a passport or driver's license. It includes a few public properties about a user in its payload. These properties cannot be tampered because doing so requires re-generating the digital signature.

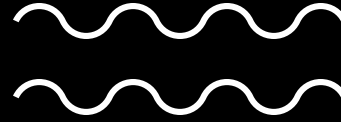
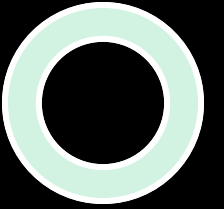


# JSON Web Tokens

When the user logs in, we generate a JWT on the server and return it to the client. We store this token on the client and send it to the server every time we need to call an API endpoint that is only accessible to authenticated users

auth.js

- Modify file **auth.js** in the folder **routes**



```
const jwt = require('jsonwebtoken');
const Joi = require('joi');
const bcrypt = require('bcrypt');
const _ = require('lodash');
const {User} = require('../models/user');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();
```

Insert  
code

```
router.post('/', async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);

  let user = await User.findOne({ email: req.body.email });
  if (!user) return res.status(400).send('Invalid email or password.');
```

```
  const validPassword = await bcrypt.compare(req.body.password, user.password)
  if (!validPassword) return res.status(400).send('Invalid email or password.');
```

```
  const token = jwt.sign({ _id: user._id }, 'jwtPrivateKey');
  res.send(token);
});
```

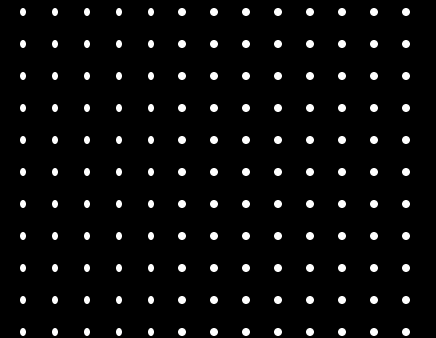
```
function validate(req) {
  ...
}
```

```
module.exports = router;
```

Modify  
code

# Generating Authentication Tokens

To generate JSON Web Tokens in an Express app use **jsonwebtoken** package





# Generating Authentication Tokens



POSTMAN

The screenshot shows the Postman interface for a POST request. The URL bar is set to `http://localhost:3000/api/auth...`. The request body is in JSON format, containing the following data:

```
{
  "email": "marizza@gmail.com",
  "password": "123321"
}
```

The response is a 200 OK status with a response time of 102 ms and a size of 378 B. The response body is displayed in the 'Body' tab, showing a long token string:

```
1 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2MTQ0NWY3MmQ4NmFhZDY1NWZ0WFkNDQpYXQiOiE2MzE4NDY1ODN9.GAqql4FNSNlVaSpSc1JQyuf9MXkXJ1ptoDelkuEEipQ
```







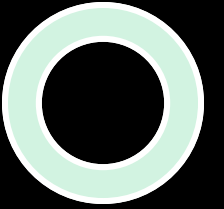
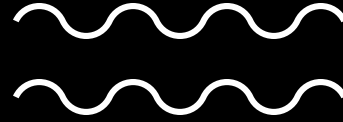
## **Storing Secrets in Environment Variables**

Never store private keys and other secrets in your codebase. Store them in environment variables.

Use the config package to read application settings stored in environment variables.

Terminal

`npm i config`



Create `config` folder

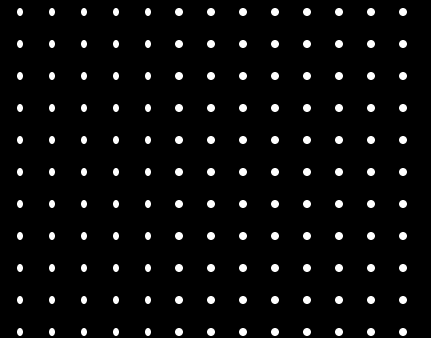
Into folder config create `default.json` file

```
{  
  "jwtPrivateKey": ""  
}
```

Into folder config create `custom-environment-variables.json` file

```
{  
  "jwtPrivateKey": "videohost_jwtPrivateKey"  
}
```

# Storing Secrets in Environment Variables



```
const config = require('config');
const jwt = require('jsonwebtoken');
const Joi = require('joi');
const bcrypt = require('bcrypt');
const _ = require('lodash');
const {User} = require('../models/user');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();
```

```
router.post('/', async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);

  let user = await User.findOne({ email: req.body.email });
  if (!user) return res.status(400).send('Invalid email or password.');
```

```
  const validPassword = await bcrypt.compare(req.body.password, user.password)
  if (!validPassword) return res.status(400).send('Invalid email or password.');
```

```
  const token = jwt.sign({ _id: user._id }, config.get('jwtPrivateKey'));
  res.send(token);
});
```

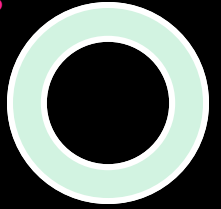
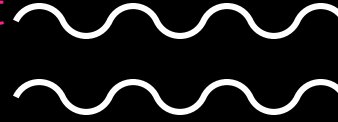
```
function validate(req) {
  const schema = Joi.object({
    email: Joi.string().min(5).max(255).required().email(),
    password: Joi.string().min(5).max(255).required()
  });

  return schema.validate(req);
}
```

```
module.exports = router;
```

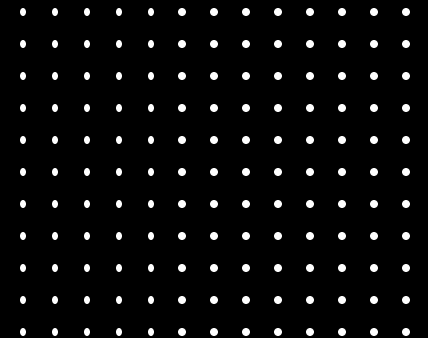
- Modify file `auth.js` in the folder `routes`

Insert  
code

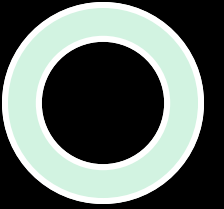


# Storing Secrets in Environment Variables

Modify  
code

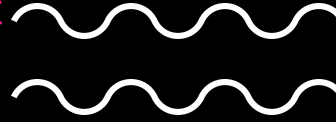


- Modify file `index.js`



```
const config = require('config');
const mongoose = require('mongoose');
const genres = require('./routes/genres');
const customers = require('./routes/customers');
const movies = require('./routes/movies');
const rentals = require('./routes/rentals');
const users = require('./routes/users');
const auth = require('./routes/auth');
const express = require('express');
const app = express();
```

Insert  
code



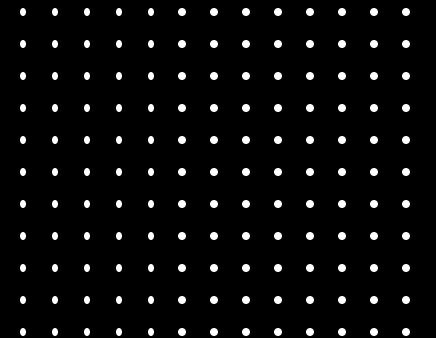
```
if(!config.get('jwtPrivateKey')){
  console.error('FATAL ERROR: jwtPrivateKey is not defined!');
  process.exit(1);
}
```

```
mongoose.connect('mongodb://localhost/videohost')
  .then(() => console.log('Connected to MongoDB...'))
  .catch(err => console.error('Could not connect to MongoDB...'));
```

```
app.use(express.json());
app.use('/api/genres', genres);
app.use('/api/customers', customers);
app.use('/api/movies', movies);
app.use('/api/rentals', rentals);
app.use('/api/users', users);
app.use('/api/auth', auth);
```

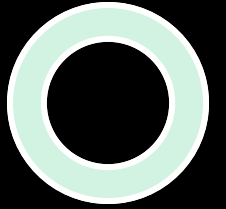
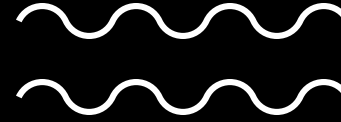
```
const port = process.env.PORT || 3000;
app.listen(port, () => console.log(`Listening on port ${port}...`));
```

# Storing Secrets in Environment Variables



Terminal

`nodemon` index.js



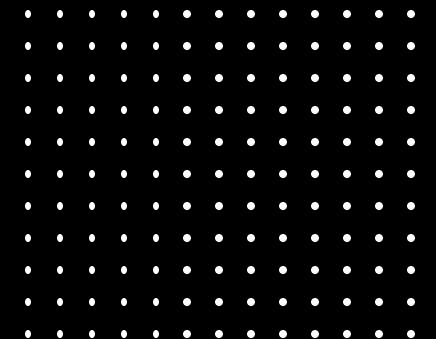
```
PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE
PS C:\NodeJS_Course\Authentication_and_Authorization> nodemon index.js
[nodemon] 2.0.12
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
FATAL ERROR: jwtPrivateKey is not defined!
[nodemon] app crashed - waiting for file changes before starting...
█
```

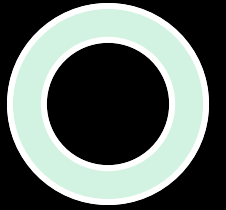
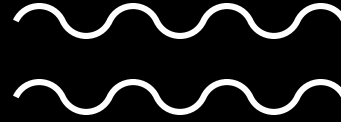
# Storing Secrets in Environment Variables

Terminal

`node` index.js

```
PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE
PS C:\NodeJS_Course\Authentication_and_Authorization> node index.js
FATAL ERROR: jwtPrivateKey is not defined!
PS C:\NodeJS_Course\Authentication_and_Authorization> █
```



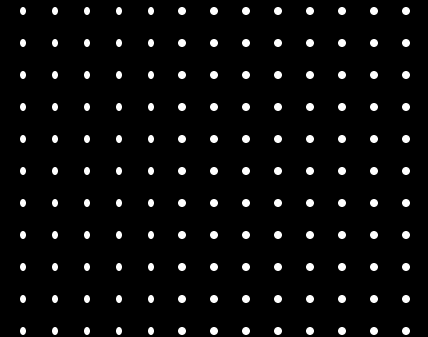


PowerShell

```
$env:videohost_jwtPrivateKey="mySecureKey"  
node index.js
```

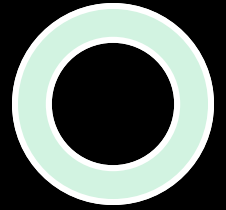
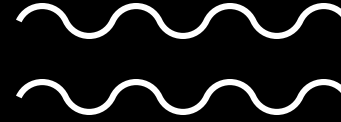
```
PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE powershell + - [ ] [X] ^ X  
  
PS C:\NodeJS_Course\Authentication_and_Authorization> $env:videohost_jwtPrivateKey="mySecureKey"  
PS C:\NodeJS_Course\Authentication_and_Authorization> node index.js  
Listening on port 3000...  
Connected to MongoDB...  
█
```

# Storing Secrets in Environment Variables

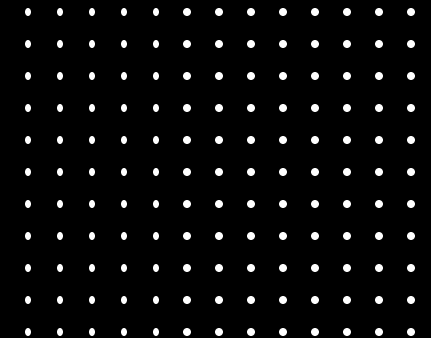
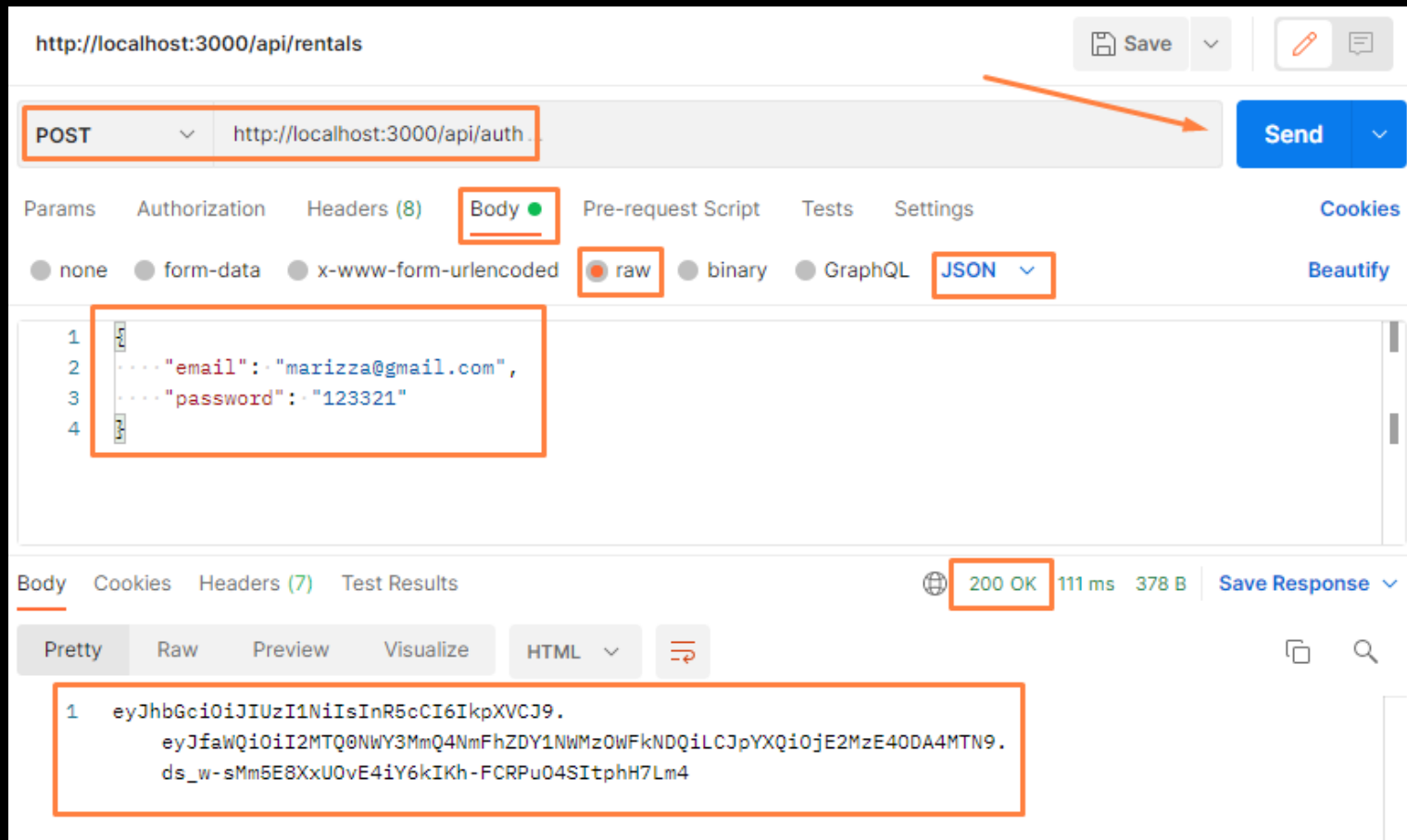




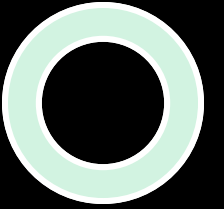
POSTMAN



# Storing Secrets in Environment Variables

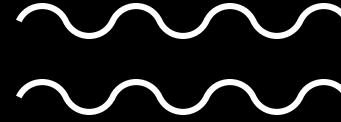


- Modify file `users.js`



```
const config = require('config');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const _ = require('lodash');
const {User, validate} = require('../models/user');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();
```

Insert  
code



```
router.post('/', async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);

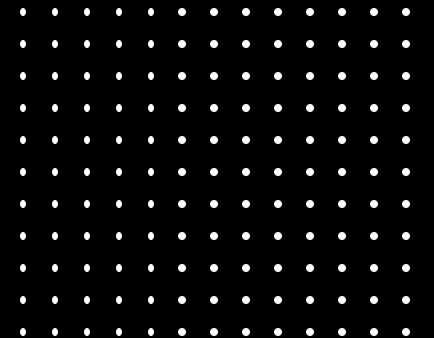
  let user = await User.findOne({ email: req.body.email });
  if (user) return res.status(400).send('User already registred.');
```

```
  user = new User(_.pick(req.body, ['name', 'email', 'password']));
  const salt = await bcrypt.genSalt(10);
  user.password = await bcrypt.hash(user.password, salt);
  await user.save();
```

```
  const token = jwt.sign({ _id: user._id }, config.get('jwtPrivateKey'));
  res.header('x-auth-token', token).send(_.pick(user, ['_id', 'name', 'email']));
});
```

```
module.exports = router;
```

Insert  
code

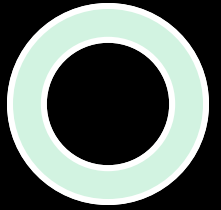


# Setting Response Headers





POSTMAN



# Setting Response Headers

POST ▼ http://localhost:3000/api/users Send ▼

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings Cookies Beautify

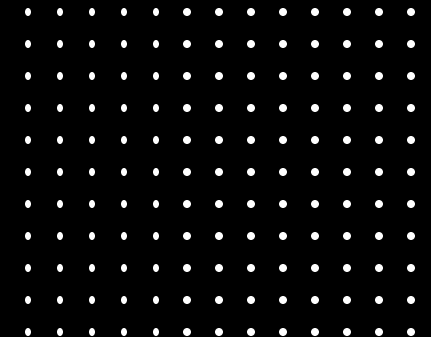
☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

```
1 {
2   "name": "marizza",
3   "email": "marizza333@gmail.com",
4   "password": "123321"
5 }
```

Body Cookies Headers (8) Test Results 200 OK 111 ms 482 B Save Response ▼

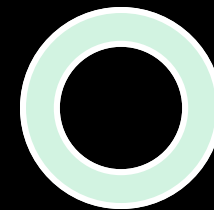
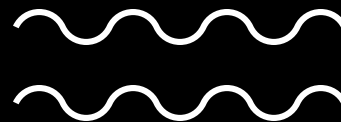
Pretty Raw Preview Visualize **JSON** ▼ ≡

```
1 {
2   "_id": "61448b318def1fb4c1075e6f",
3   "name": "marizza",
4   "email": "marizza333@gmail.com"
5 }
```





POSTMAN



POST http://localhost:3000/api/users Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

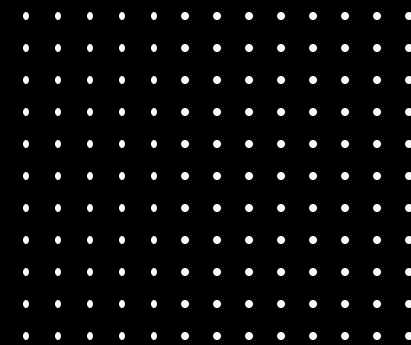
☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** Beautify

```
1 {
2   ... "name": "marizza",
3   ... "email": "marizza333@gmail.com",
4   ... "password": "123321"
5 }
```

Body Cookies **Headers (8)** Test Results 200 OK 111 ms 482 B Save Response

KEY	VALUE
X-Powered-By ⓘ	Express
x-auth-token ⓘ	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2MTQ0OGIz...
Content-Type ⓘ	application/json; charset=utf-8
Content-Length ⓘ	82
ETag ⓘ	W/"52-RdIOrepRYjk7YGMbqKKE3K7llml"
Date ⓘ	Fri, 17 Sep 2021 12:33:54 GMT
Connection ⓘ	keep-alive
Keep-Alive ⓘ	timeout=5

# Setting Response Headers



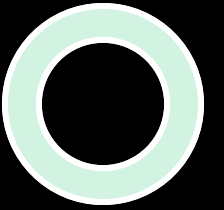
```
const config = require('config');
const jwt = require('jsonwebtoken');
const Joi = require('joi');
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    minlength: 5,
    maxlength: 50
  },
  email: {
    type: String,
    required: true,
    minlength: 5,
    maxlength: 255,
    unique: true
  },
  password: {
    type: String,
    required: true,
    minlength: 5,
    maxlength: 1024
  },
})

userSchema.methods.generateAuthToken = function() {
  const token = jwt.sign({ _id: this._id }, config.get('jwtPrivateKey'));
  return token;
}

const User = mongoose.model('User', userSchema);
...

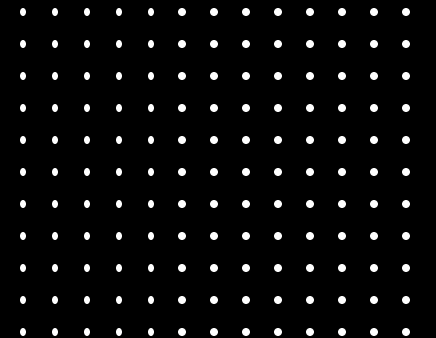
```

- Modify file `user.js`

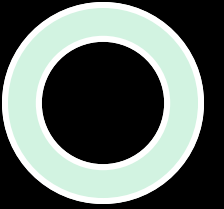


# Encapsulating Logic in Mongoose Models

Modify  
code



- Modify file `users.js`



```
const bcrypt = require('bcrypt');
const _ = require('lodash');
const {User, validate} = require('../models/user');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();
```

```
router.post('/', async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);
```

```
  let user = await User.findOne({ email: req.body.email });
  if (user) return res.status(400).send('User already registred.');
```

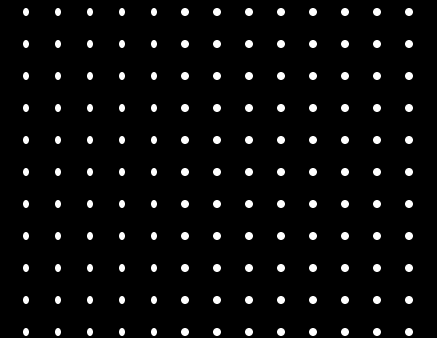
```
  user = new User(_.pick(req.body, ['name', 'email', 'password']));
  const salt = await bcrypt.genSalt(10);
  user.password = await bcrypt.hash(user.password, salt);
  await user.save();
```

```
  const token = user.generateAuthToken();
  res.header('x-auth-token', token).send(_.pick(user, ['_id', 'name', 'email']));
});
```

```
module.exports = router;
```

Modify  
code

# Encapsulating Logic in Mongoose Models



```
const config = require('config');
const jwt = require('jsonwebtoken');
const Joi = require('joi');
const bcrypt = require('bcrypt');
const _ = require('lodash');
const {User} = require('../models/user');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();
```

```
router.post('/', async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);

  let user = await User.findOne({ email: req.body.email });
  if (!user) return res.status(400).send('Invalid email or password.');

  const validPassword = await bcrypt.compare(req.body.password, user.password)
  if (!validPassword) return res.status(400).send('Invalid email or password.');
```

```
  const token = user.generateAuthToken();
  res.send(token);
```

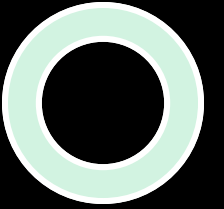
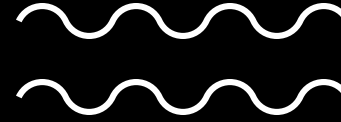
```
});
```

```
function validate(req) {
  const schema = Joi.object({
    email: Joi.string().min(5).max(255).required().email(),
    password: Joi.string().min(5).max(255).required()
  });

  return schema.validate(req);
}
```

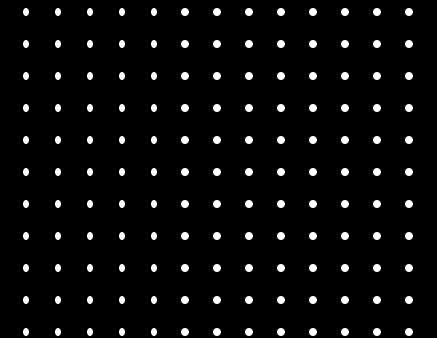
```
module.exports = router;
```

- Modify file `auth.js`

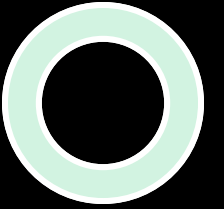
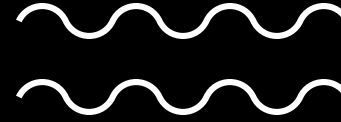


# Encapsulating Logic in Mongoose Models

Modify  
code



- Create folder **middleware**
- Create file **auth.js** into the folder **middleware**



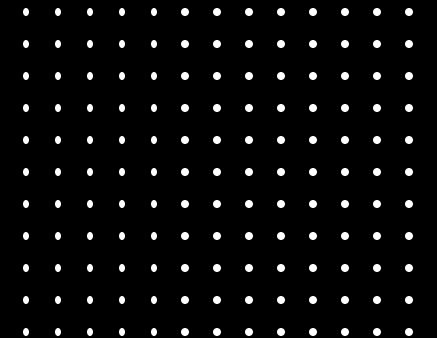
```
const config = require('config');
const jwt = require('jsonwebtoken');

module.exports = function (req, res, next) {
  const token = req.header('x-auth-token');
  if (!token) return res.status(401).send('Access denied. No token provide.');
```

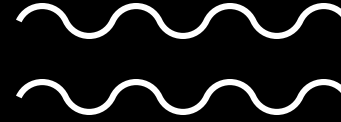
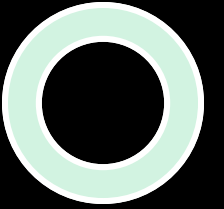
  

```
  try{
    const decoded = jwt.verify(token, config.get('jwtPrivateKey'));
    req.user = decoded;
    next();
  }
  catch(ex){
    res.status(400).send('Invalid token!');
  }
}
```

# Authorization Middleware



- Modify file `genres.js`



```
const auth = require('../middleware/auth');
const Joi = require('joi');
const {Genre, validate} = require('../models/genre');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();

router.get('/', async (req, res) => {
  const genres = await Genre.find().sort('name');
  res.send(genres);
});

router.post('/', auth, async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);

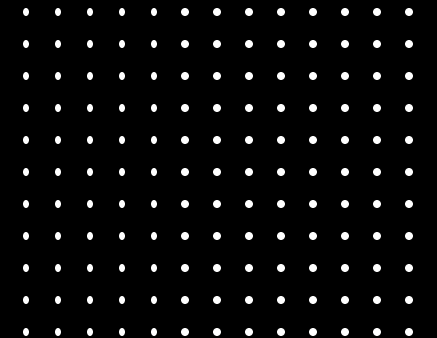
  let genre = new Genre({ name: req.body.name });
  genre = await genre.save();

  res.send(genre);
});

...
```

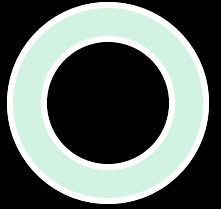
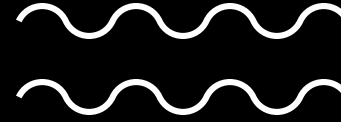
Insert  
code

# Protecting Routes





POSTMAN



POST  [Send](#)

Params Authorization **Headers (9)** Body ☒ Pre-request Script Tests Settings [Cookies](#)

Headers ☐ 8 hidden

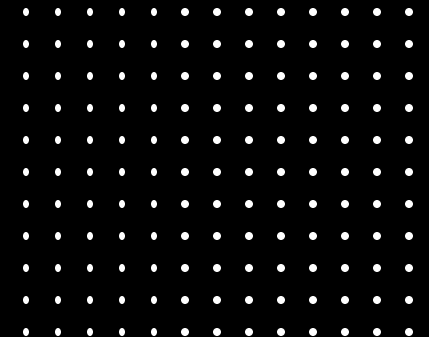
	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e...				
	Key	Value	Description			

Body [Cookies](#) [Headers \(7\)](#) [Test Results](#) [200 OK](#) [25 ms](#) [296 B](#) [Save Response](#)

Pretty Raw Preview Visualize JSON

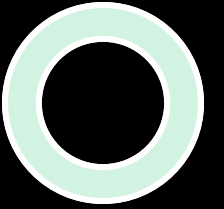
```
1 {
2   "name": "New Genre",
3   "_id": "61449d12f7140f7115a72509",
4   "__v": 0
5 }
```

# Protecting Routes





- Modify file `customers.js`



```
const auth = require('../middleware/auth');
const Joi = require('joi');
const {Customer, validate} = require('../models/customer');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();

router.get('/', async (req, res) => {
  const customers = await Customer.find().sort('name');
  res.send(customers);
});

router.post('/', auth, async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);

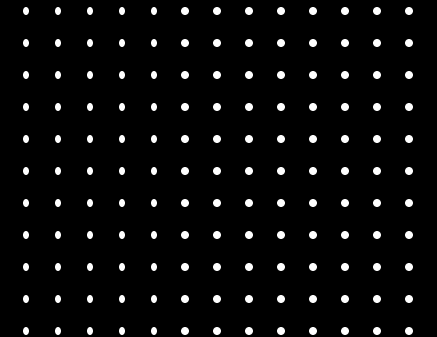
  let customer = new Customer({
    name: req.body.name,
    isGold: req.body.isGold,
    phone: req.body.phone
  });
  customer = await customer.save();

  res.send(customer);
});

...
```

Insert  
code

# Protecting Routes



```
const auth = require('../middleware/auth');
const Joi = require('joi');
const {Movie, validate} = require('../models/movie');
const {Genre} = require('../models/genre');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();
```

```
router.get('/', async (req, res) => {
  const movies = await Movie.find().sort('name');
  res.send(movies);
});
```

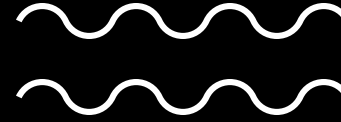
```
router.post('/', auth, async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);
```

```
  const genre = await Genre.findById(req.body.genreId);
  if (!genre) return res.status(400).send('Invalid genre.');
```

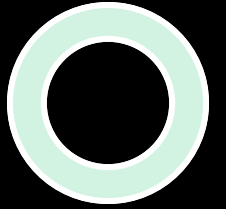
```
  let movie = new Movie({
    title: req.body.title,
    genre: {
      _id: genre._id,
      name: genre.name
    },
    numberInStock: req.body.numberInStock,
    dailyRentalRate: req.body.dailyRentalRate
  });
  movie = await movie.save();

  res.send(movie);
});
```

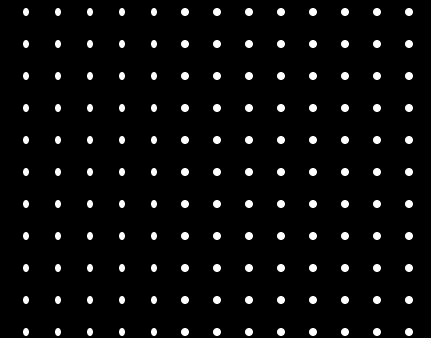
- Modify file `movies.js`



Insert  
code



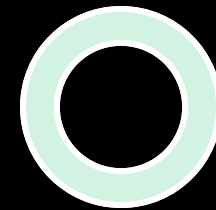
# Protecting Routes



```
const auth = require('../middleware/auth');
const bcrypt = require('bcrypt');
const _ = require('lodash');
const {User, validate} = require('../models/user');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();
```

Insert  
code

- Modify file `users.js`



```
router.get('/me', auth, async (req, res) => {
  const user = await User.findById(req.user._id).select('-password');
  res.send(user);
});
```

```
router.post('/', async (req, res) => {
  const { error } = validate(req.body);
  if (error) return res.status(400).send(error.details[0].message);

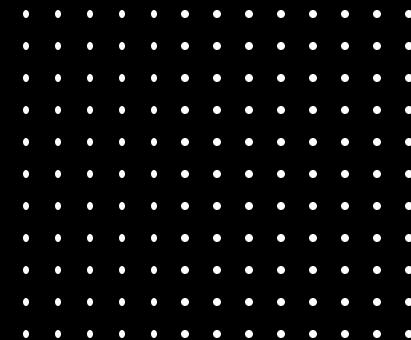
  let user = await User.findOne({ email: req.body.email });
  if (user) return res.status(400).send('User already registred.');

  user = new User(_.pick(req.body, ['name', 'email', 'password']));
  const salt = await bcrypt.genSalt(10);
  user.password = await bcrypt.hash(user.password, salt);
  await user.save();

  const token = user.generateAuthToken();
  res.header('x-auth-token', token).send(_.pick(user, ['_id', 'name', 'email']));
});

module.exports = router;
```

# Getting the Current User

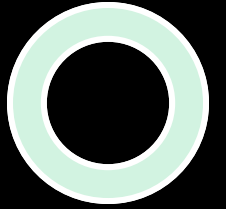
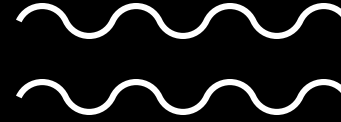


```
const config = require('config');
const jwt = require('jsonwebtoken');
const Joi = require('joi');
const mongoose = require('mongoose');
```

```
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    minlength: 5,
    maxlength: 50
  },
  email: {
    type: String,
    required: true,
    minlength: 5,
    maxlength: 255,
    unique: true
  },
  password: {
    type: String,
    required: true,
    minlength: 5,
    maxlength: 1024
  },
  isAdmin: Boolean,
})
```

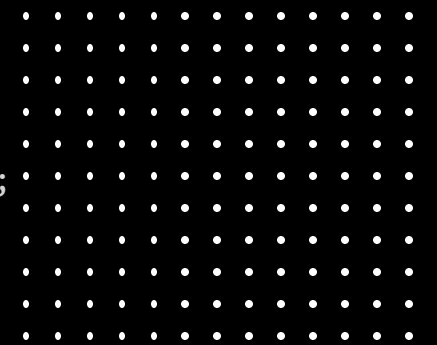
```
userSchema.methods.generateAuthToken = function() {
  const token = jwt.sign({ _id: this._id, isAdmin: this.isAdmin }, config.get('jwtPrivateKey'));
  return token;
}
...
```

- Modify file `user.js`

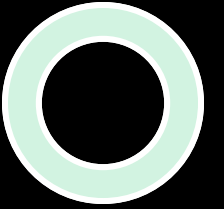
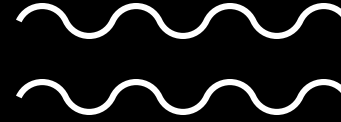


# Role-based Authorization

Insert  
code

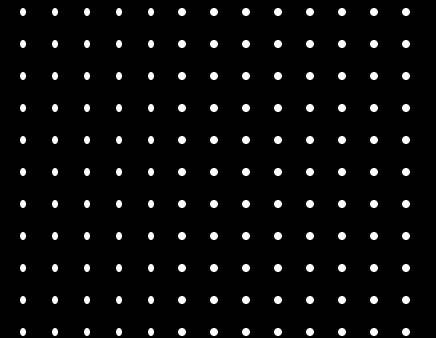


- Create folder **middleware**
- Create file **admin.js** into the folder **middleware**



```
module.exports = function (req, res, next){  
  // 401 Unauthorized  
  // 403 Forbidden  
  
  if (!req.user.isAdmin) return res.status(403).send('Access denied.')  
  next();  
}
```

# Role-based Authorization



```
const auth = require('../middleware/auth');
const admin = require('../middleware/admin');
const Joi = require('joi');
const {Genre, validate} = require('../models/genre');
const mongoose = require('mongoose');
const express = require('express');
const router = express.Router();
```

```
router.get('/', async (req, res) => {
  ...
});
```

```
router.post('/', auth, async (req, res) => {
  ...
});
```

```
router.put('/:id', async (req, res) => {
  ...
});
```

```
router.delete('/:id', [auth, admin], async (req, res) => {
  const genre = await Genre.findByIdAndRemove(req.params.id);

  if (!genre) return res.status(404).send('The genre with the given ID was not found.');
```

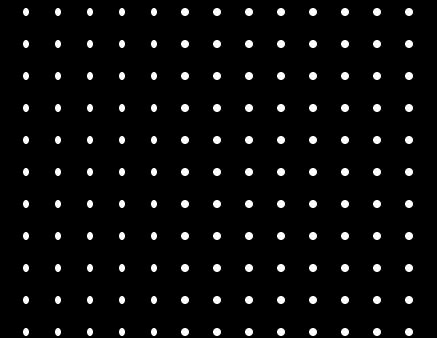
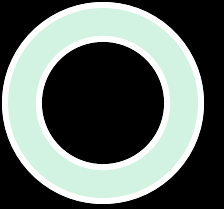
```
  res.send(genre);
});
```

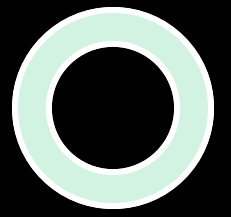
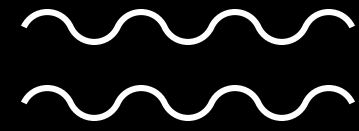
```
router.get('/:id', async (req, res) => {
  ...
});
module.exports = router;
```

- Modify file `genres.js`

Insert  
code

# Role-based Authorization





Edit user document in the MongoDB Compass:  
Add property **isAdmin: true** (Boolean)

# Role-based Authorization

videohost.users

DOCUMENTS 3 TOTAL SIZE 465B AVG. SIZE 155B INDEXES 2 TOTAL SIZE 72.0KB AVG. SIZE 36.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER { field: 'value' } OPTIONS FIND RESET

ADD DATA VIEW

Displaying documents 1 - 3 of 3 REFRESH

1 \_id: ObjectId("61445f72d86aad655c39ad44")

2 name: "marizza"

3 email: "marizza@gmail.com"

4 password: "\$2b\$10\$AEG8feaJ6Ru1EeTFSKy2t./Kv5LxeIT/.eDY1zPOiopoKdsISJlk6"

5 isAdmin: true

6 \_\_v: 0

ObjectId

String

String

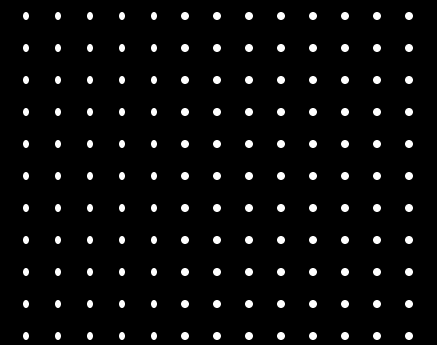
String

Boolean

Int32

Document Modified.

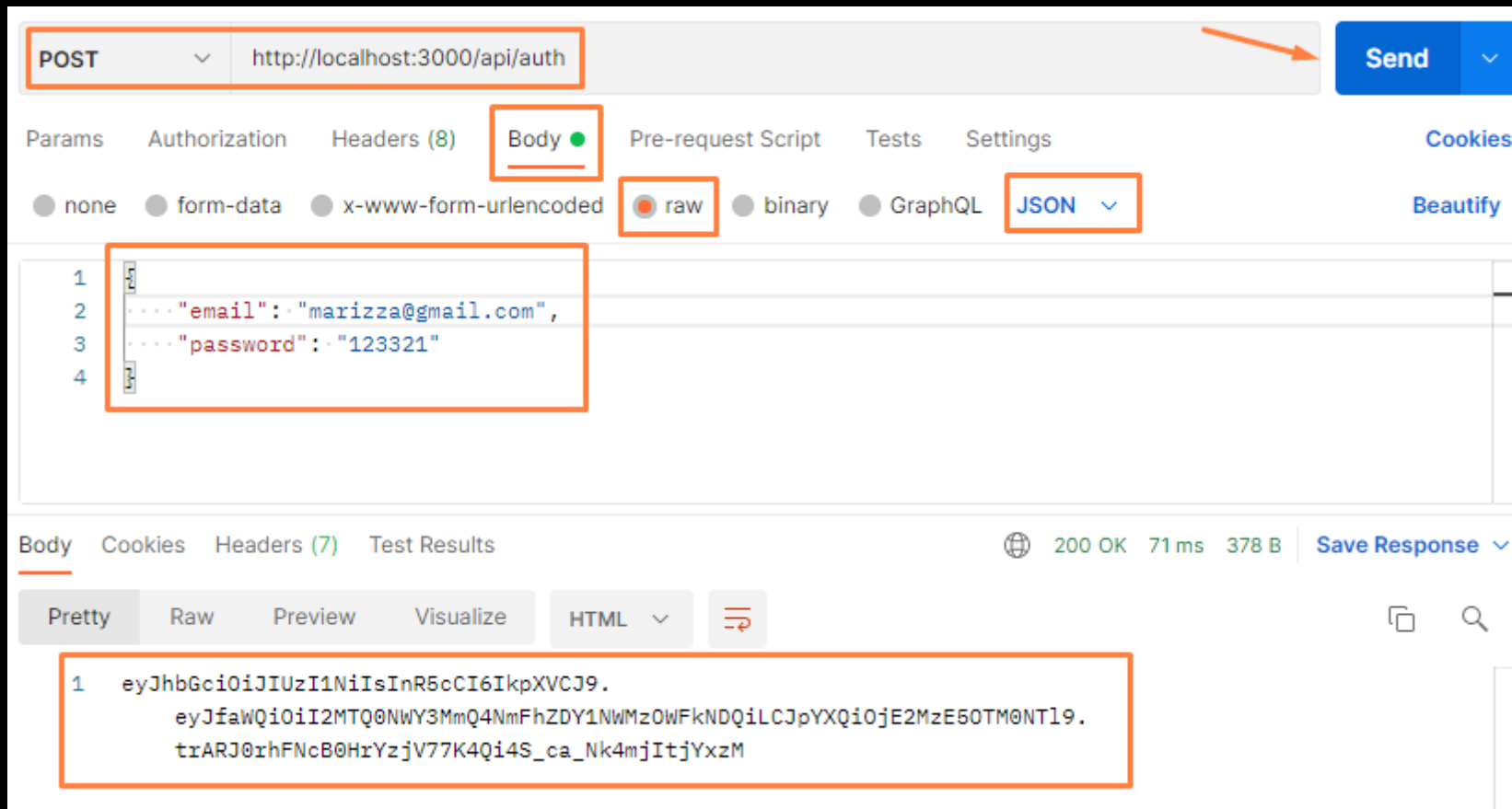
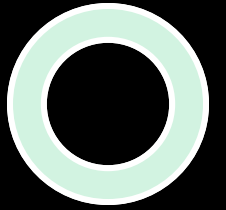
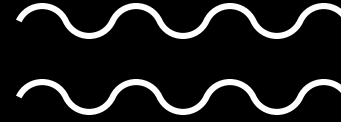
CANCEL UPDATE



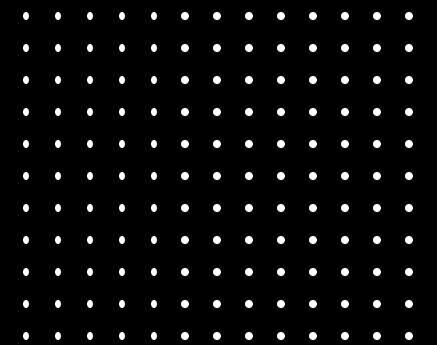


POSTMAN

Authorization with edited user



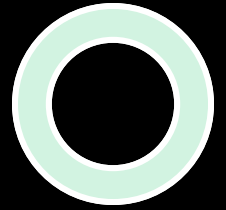
# Role-based Authorization







POSTMAN



DELETE ☐ http://localhost:3000/api/genres/61449d12f7140f7115a72509 Send

Params Authorization **Headers (9)** Body ☒ Pre-request Script Tests Settings Cookies

Headers ☐ 8 hidden

	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e...				
	Key	Value	Description			

Paste Admin Token

Body Cookies Headers (7) Test Results 200 OK 17 ms 296 B Save Response

Pretty Raw Preview Visualize JSON ☐

```
1 {
2   "_id": "61449d12f7140f7115a72509",
3   "name": "New Genre",
4   "__v": 0
5 }
```

# Role-based Authorization

