Authentication and Authorization

Whats Auth

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.

Helping Code

https://1drv.ms/f/s!AtGKdbMmNBGd0w6uAf8X7FHX584T

This is a sample code with before and after.

Before section is a simple API built on top of mongoose, express without auth and in after a complete API with auth is implemented.

Users Model

Name

Email

Password: // must be hashed

isAdmin

Hashing Function

npm i bcryptjs

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

```
const salt = await bcrypt.genSalt(10);
```

```
const hashed = await bcrypt.hash('1234', salt);
```

```
//salt must be generated once in lifetime
```

Validating Password

const isValid = await bcrypt.compare(‘1234’, hashed);

hashed is already stored password inside our db

Types of Authentication

Session Based

- Typically used in server side rendering apps

JSON Web Token Based

- Typically used where APIs are exposed

Session Based Auth

```
 router.get("/login", function (req, res, next) {
    return res.render("site/login");
 });
```

//render a form to get username and password

Handle Form and set session

```
router.post("/login", async function (req, res, next) {
  let user = await User.findOne({ email: req.body.email });
  if (!user) return res.redirect("/login");
  const validPassword = await bcrypt.compare(req.body.password, user.password);
  if (validPassword) {
    req.session.user = user;
    return res.redirect("/");
  } else return res.redirect("/login");
});
```

Login.ejs

```
<form action="" method="POST">
        <input
          type="text"
          name="email"
          value="admin@admin.com"
        />
        <input
          type="password"
          name="password"
          value="admin"
        />
        <button class="btn btn-info" type="submit">Login</button>
  </form>
```

Logging Out

```
router.get("/logout", async (req, res) => {
  req.session.user = null;
  console.log("session clear");
  return res.redirect("/login");
});
```

Register a User

```
router.post("/register", async function (req, res, next) {
  let user = await User.findOne({ email: req.body.email });
  if (user) {
    return res.redirect("/register");
  }
  user = new User(req.body);
  const salt = await bcrypt.genSalt(10);
  user.password = await bcrypt.hash(req.body.password, salt);

  await user.save();
  return res.redirect("/login");
});
```

Protecting a route (make a middleware) /middlewares/checkSessionAuth

```
async function checkSessionAuth(req, res, next)
{
   if (!req.session.user) {
     return res.redirect("/login");
   }
   next();
}
module.exports = checkSessionAuth;
```

How to Apply middleware

Apply to a router

- `app.use("/", sessionAuth, indexRouter);`

Apply to a single route

```
router.get("/",sessionAuth, function (req, res, next) {
  res.render("site/myaccount");
});
```

JSON Web Token (JWT)

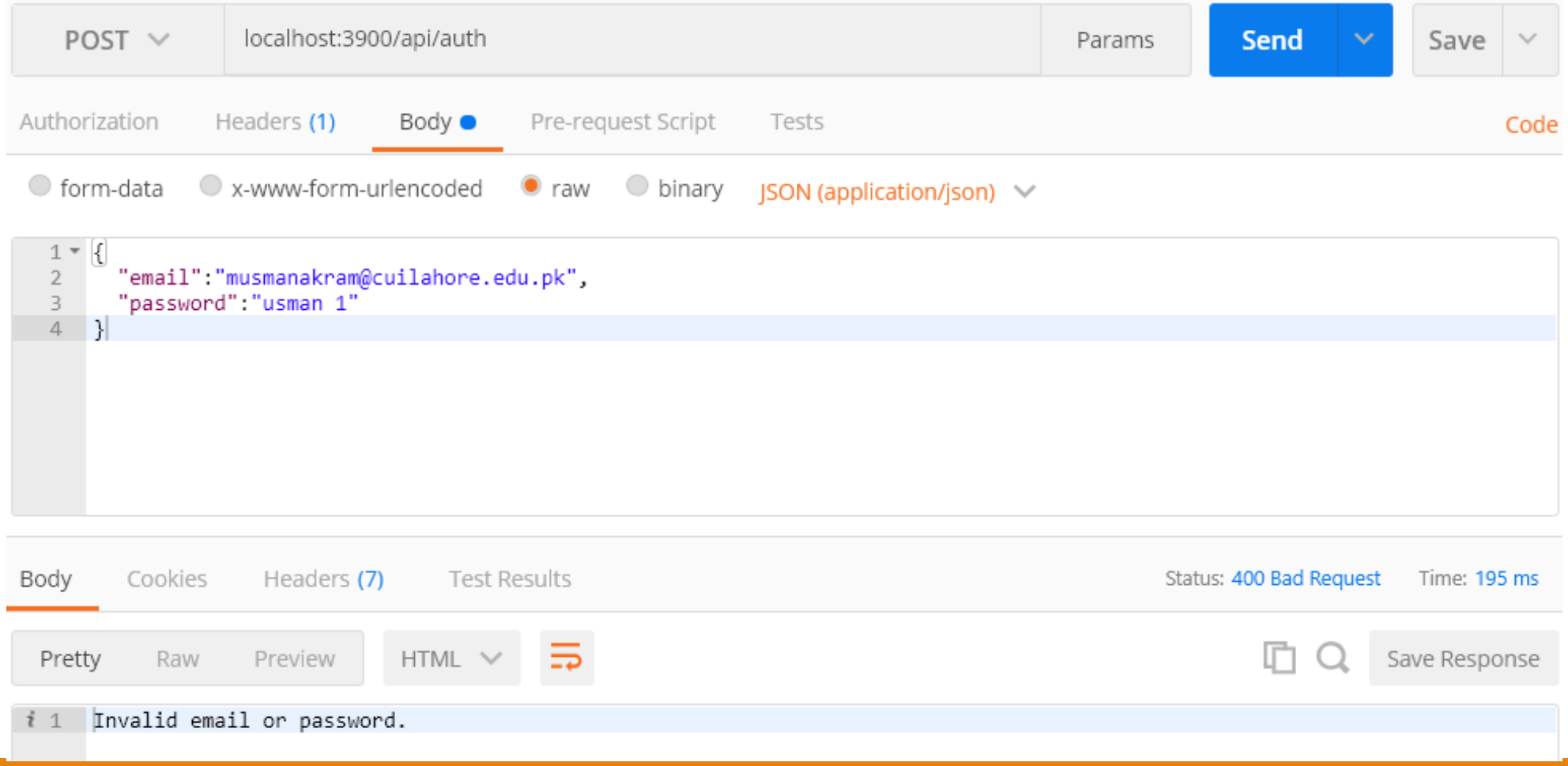
JSON object encoded as a long string

Similar to a passport or driver’s license

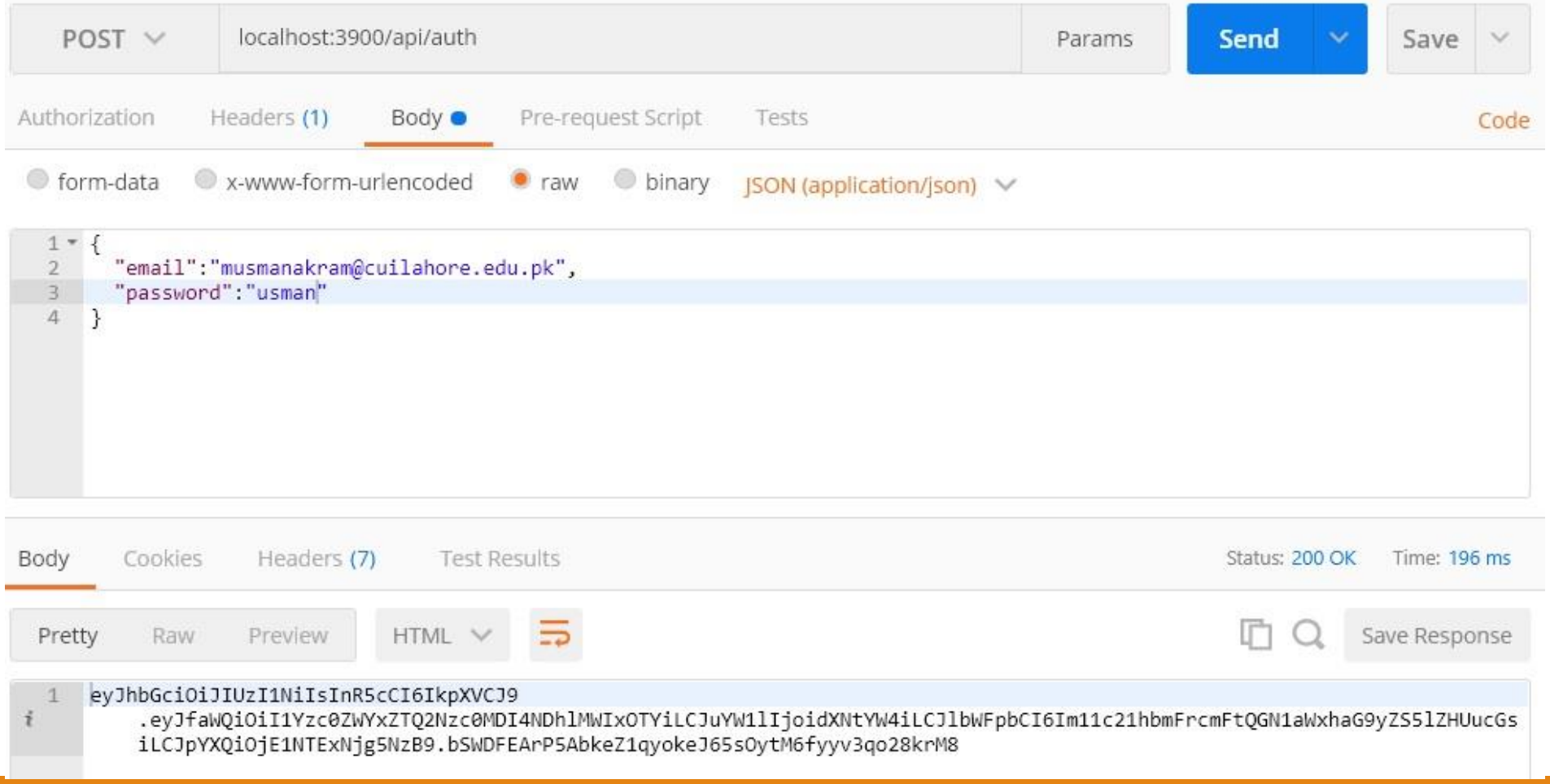
includes a few public properties about a user

These properties cannot be tampered because doing so requires re-generating the digital signature.

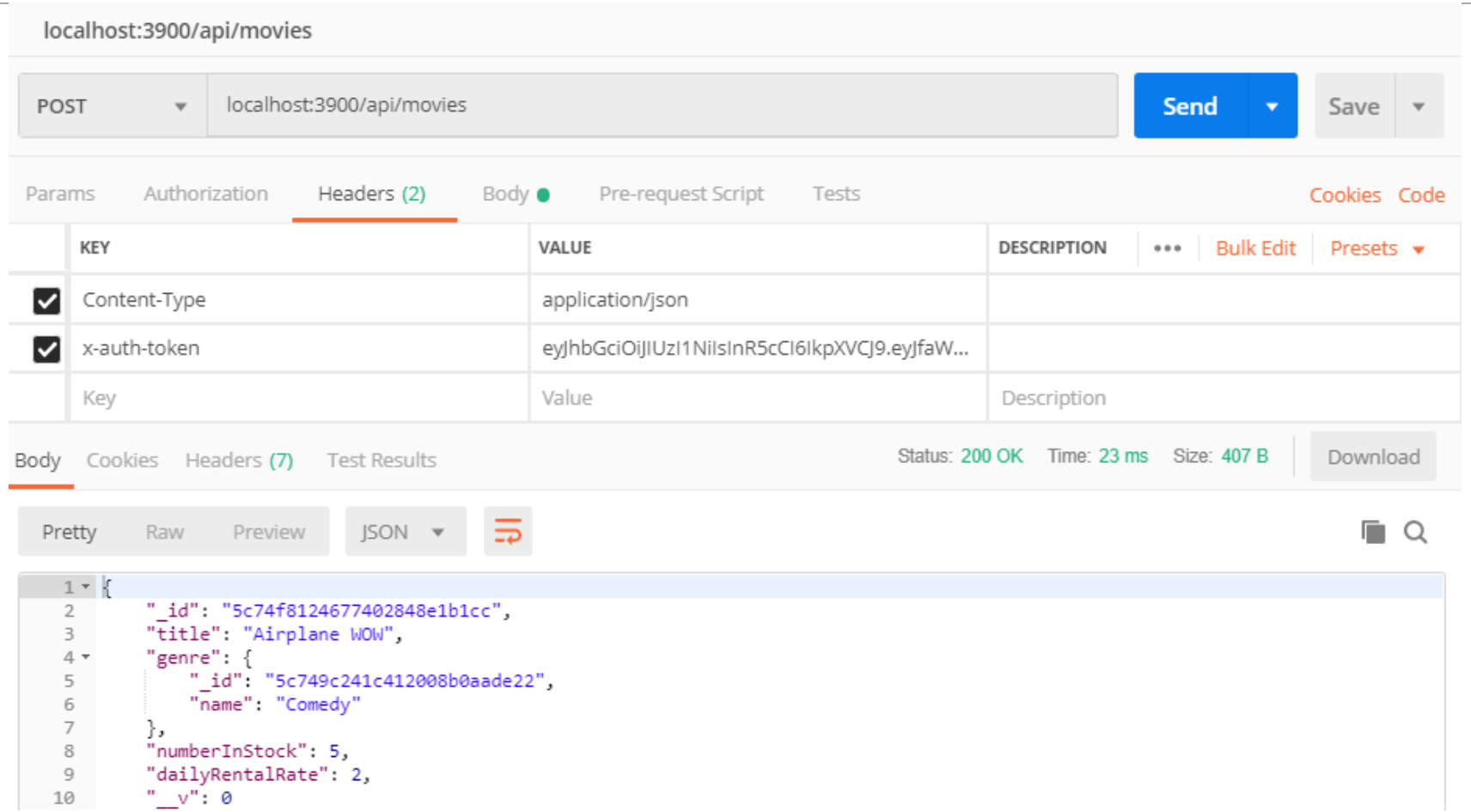
Logging In Fail



Logging In Success



Sending Request with token



JSON Web Token (JWT) Process

If a user provide right credentials we generate a token and send back to client.

Client send that token back with each request

JSON Web Token (JWT) Process

```
const jwt = require('jsonwebtoken');
const token = jwt.sign({ _id: user._id}, 'privateKey');


//install jsonwebtoken if not done yet
```

Private Keys

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.

Fatty Models

```
// Adding a method to a Mongoose model
userSchema.methods.
generateAuthToken = function() {
}
const token = user.generateAuthToken();
```

Things to Do

Implement authorization using a middleware function.

Return a 401 error (unauthorized) if the client doesn’t send a valid token.

Return 403 (forbidden) if

the user provided a valid token but is not allowed to perform the given operation.

You don’t need to implement logging out on the server. Implement it on the client by simply removing the JWT from the client.

A Quick Knowledge Check

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

Lets Comb The Code

https://1drv.ms/f/s!AtGKdbMmNBGd0w6uAf8X7FHX584T

This is a sample code with before and after.

Before section is a simple API built on top of mongoose, express without auth and in after a complete API with auth is implemented.