200312082

Norestforthewiccad Unit Test

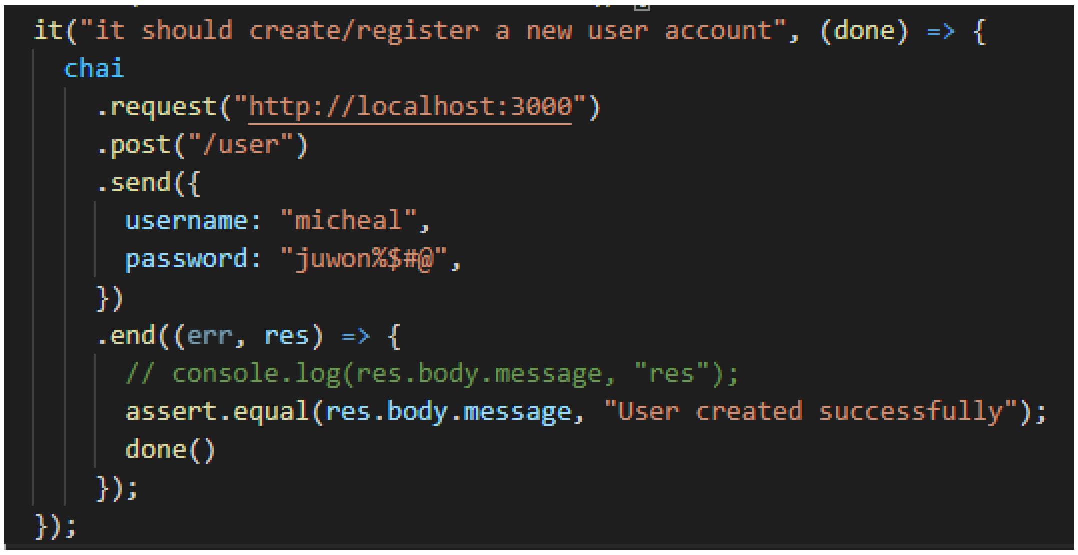
The repetitive approach used in all test and test cases is that; A test that fails is firstly written and a code that passes the test is implemented. Strategies are also explained in the test cases.

Test set 1 - /user route

This set of tests focus on the /user endpoint by ensuring a user can be created, logged in and out of the program. There is an endpoint '/user/all' that returns all the user in the system as a list(Array) and each user is an object with various fields(properties) like username, password and logged-in which determines whether a user is logged in to the system or not, since it's possible to have different user logged in on a server at the same time.

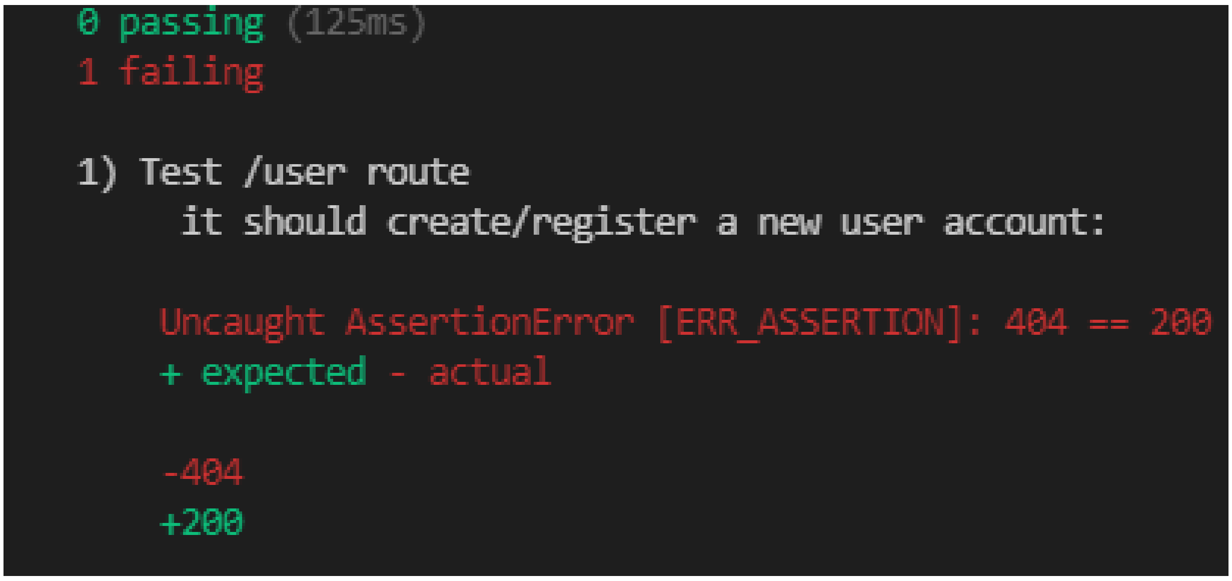
- ### Test 1 - create a new account (POST)

  The endpoint was firstly tested to ensure that the right message is returned and actual account creation is tested.
  BEFORE:



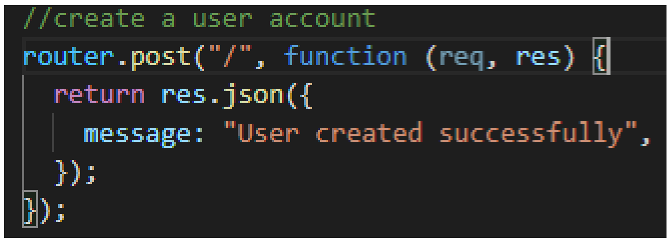
```
it("it should create/register a new user account", (done) => {
  chai
    .request("http://localhost:3000")
    .post("/user")
    .send({
      username: "micheal",
      password: "juwon%$#@",
    })
    .end((err, res) => {
      // console.log(res.body.message, "res");
      assert.equal(res.body.message, "User created successfully");
      done()
    });
});
```

-

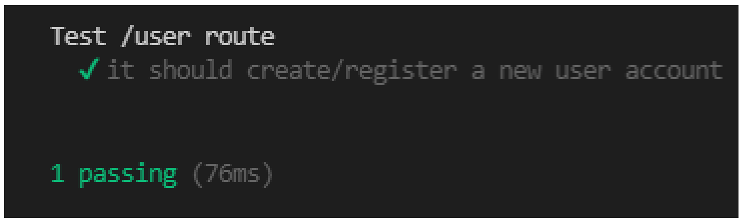


```
  0 passing (125ms)
  1 failing

  1) Test /user route
       it should create/register a new user account:

     Uncaught AssertionError [ERR_ASSERTION]: 404 == 200
     + expected - actual

     -404
     +200
```

AFTER:

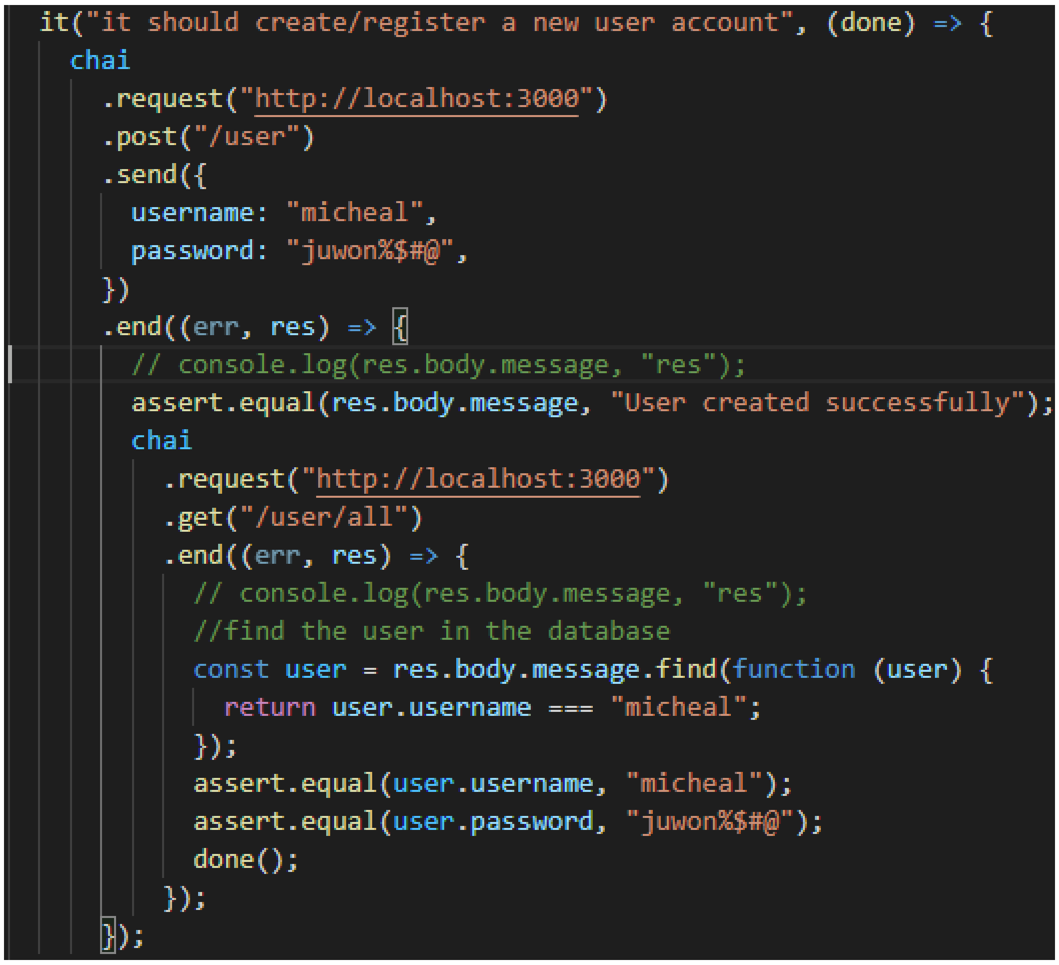


```
//create a user account
router.post("/", function (req, res) {
  return res.json({
    message: "User created successfully",
  });
});
```

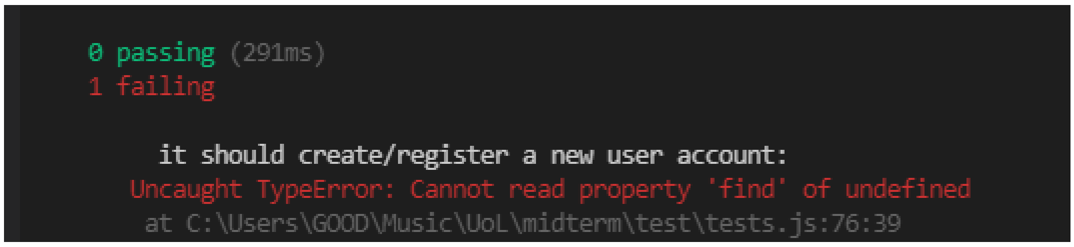


```
  Test /user route
    ✓ it should create/register a new user account


  1 passing (76ms)
```

BEFORE:

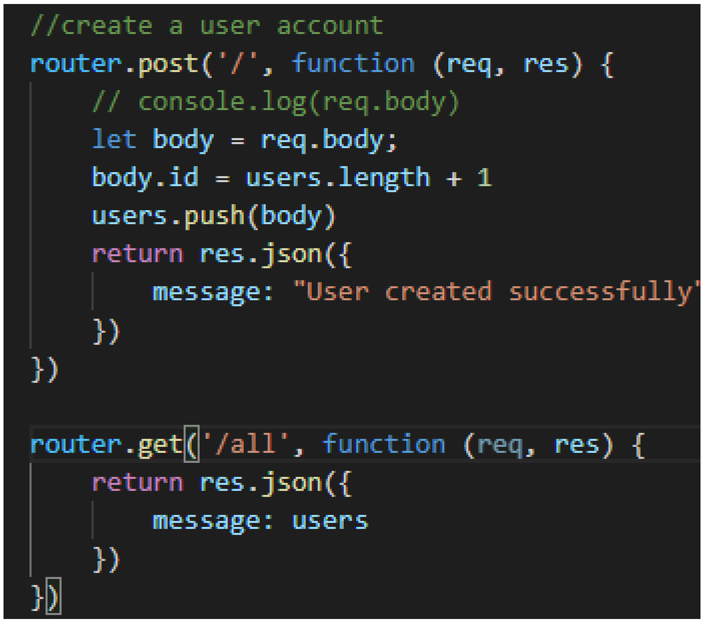


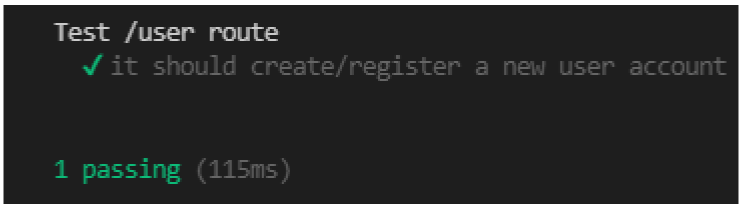
```
  it("it should create/register a new user account", (done) => {
    chai
      .request("http://localhost:3000")
      .post("/user")
      .send({
        username: "micheal",
        password: "juwon%$#@",
      })
      .end((err, res) => {
        // console.log(res.body.message, "res");
        assert.equal(res.body.message, "User created successfully");
        chai
          .request("http://localhost:3000")
          .get("/user/all")
          .end((err, res) => {
            // console.log(res.body.message, "res");
            //find the user in the database
            const user = res.body.message.find(function (user) {
              return user.username === "micheal";
            });
            assert.equal(user.username, "micheal");
            assert.equal(user.password, "juwon%$#@");
            done();
          });
      });
```



```
    0 passing (291ms)
    1 failing

         it should create/register a new user account:
       Uncaught TypeError: Cannot read property 'find' of undefined
        at C:\Users\GOOD\Music\UoL\midterm\test\tests.js:76:39
```

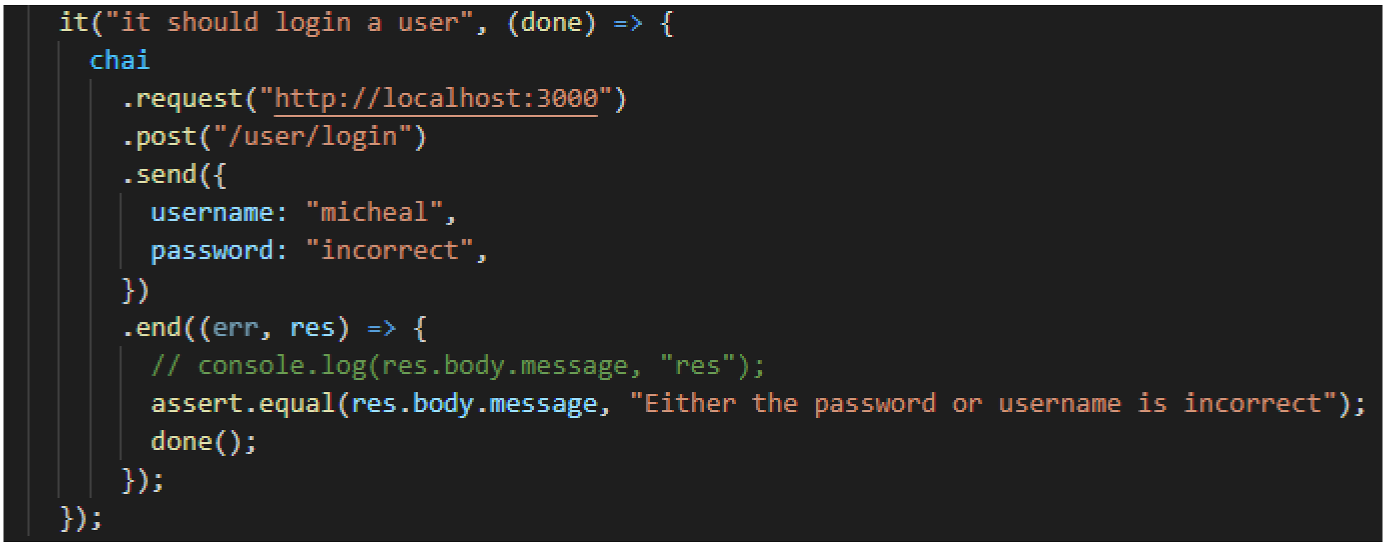
AFTER:

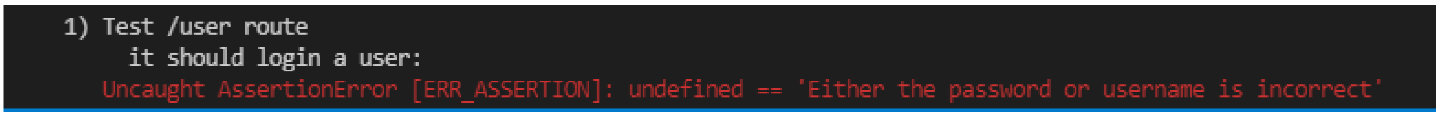

```
//create a user account
router.post('/', function (req, res) {
    // console.log(req.body)
    let body = req.body;
    body.id = users.length + 1
    users.push(body)
    return res.json({
        message: "User created successfully'
    })
})

router.get('/all', function (req, res) {
    return res.json({
        message: users
    })
})
```


```
  Test /user route
    ✓ it should create/register a new user account


  1 passing (115ms)
```

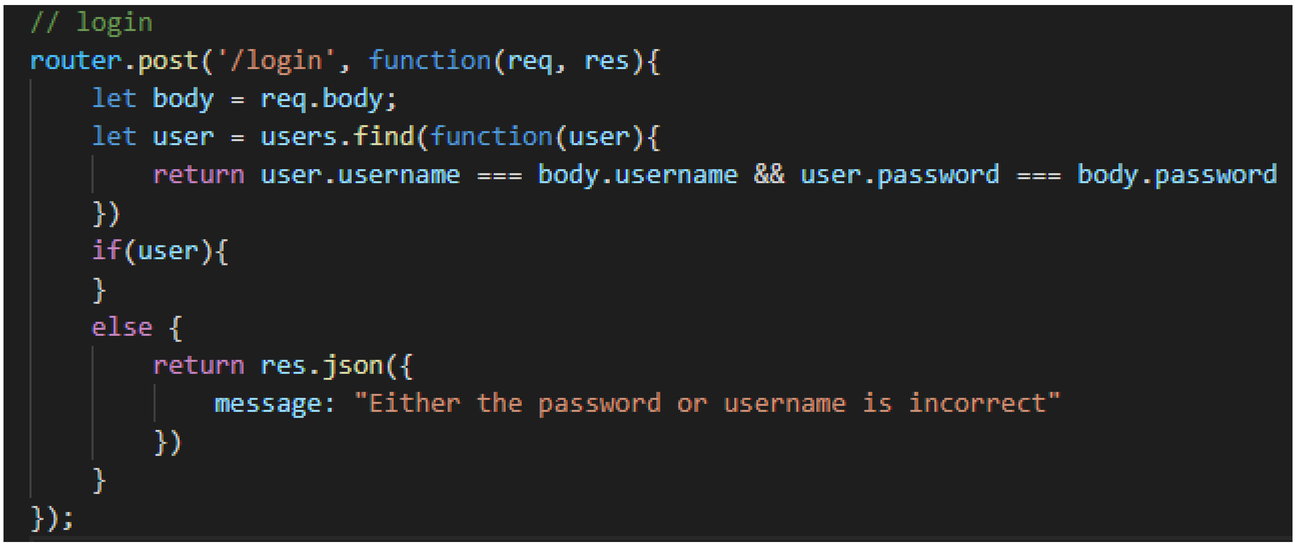
- ## Test 2 - Login user (POST)

  The test checks for invalid login details and valid details.
  BEFORE:

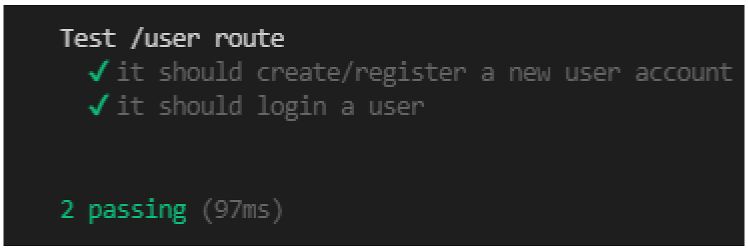

```
  it("it should login a user", (done) => {
    chai
      .request("http://localhost:3000")
      .post("/user/login")
      .send({
        username: "micheal",
        password: "incorrect",
      })
      .end((err, res) => {
        // console.log(res.body.message, "res");
        assert.equal(res.body.message, "Either the password or username is incorrect");
        done();
      });
  });
```


```
  1) Test /user route
       it should login a user:
     Uncaught AssertionError [ERR_ASSERTION]: undefined == 'Either the password or username is incorrect'
```

AFTER:

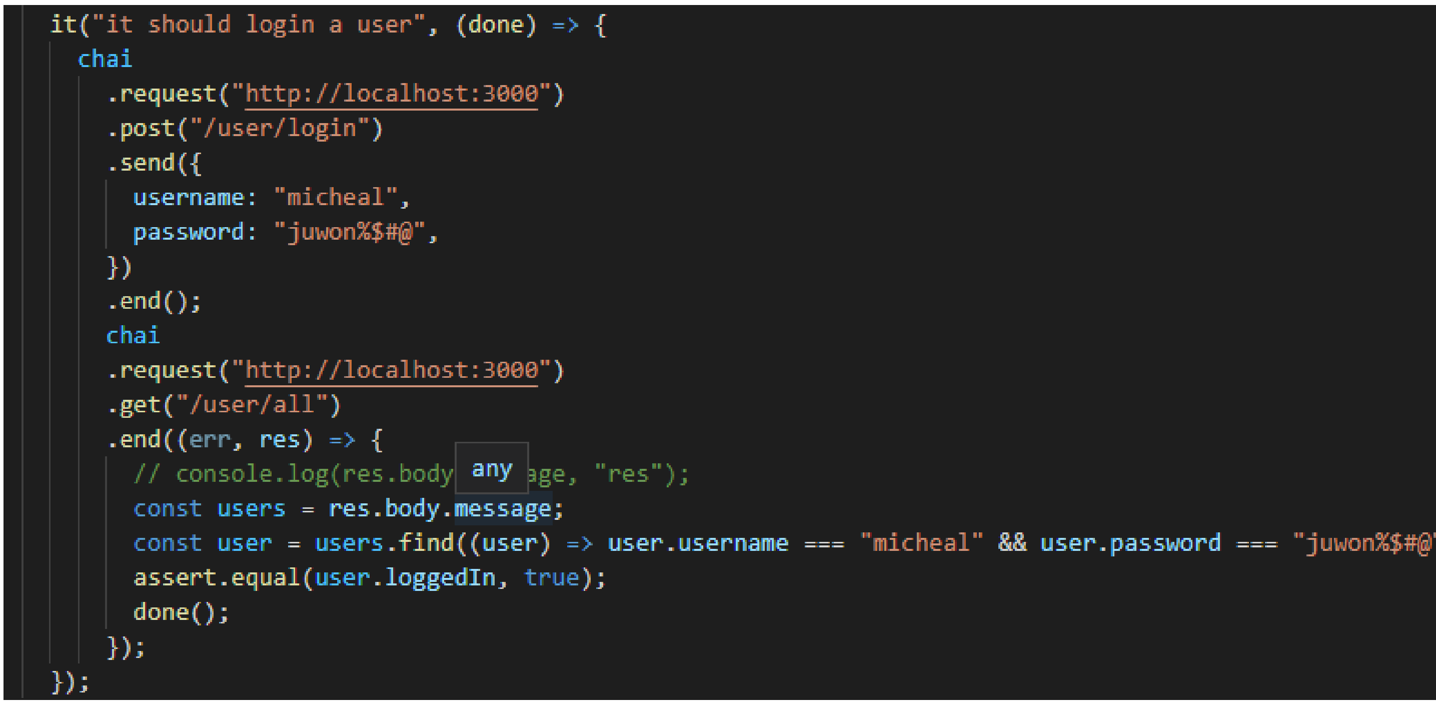


```
// login
router.post('/login', function(req, res){
    let body = req.body;
    let user = users.find(function(user){
        return user.username === body.username && user.password === body.password
    })
    if(user){
    }
    else {
        return res.json({
            message: "Either the password or username is incorrect"
        })
    }
});
```

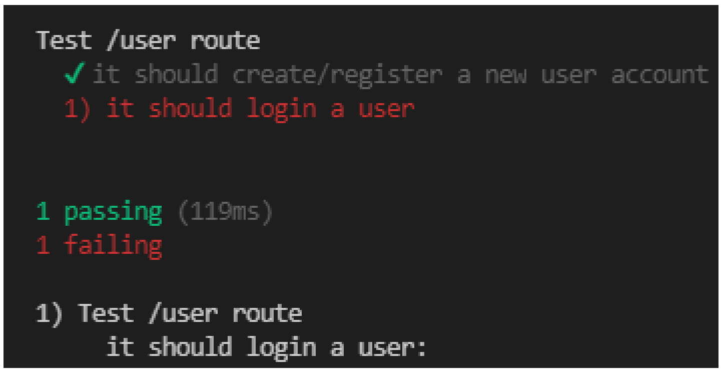


```
  Test /user route
    ✓ it should create/register a new user account
    ✓ it should login a user


  2 passing (97ms)
```

BEFORE:

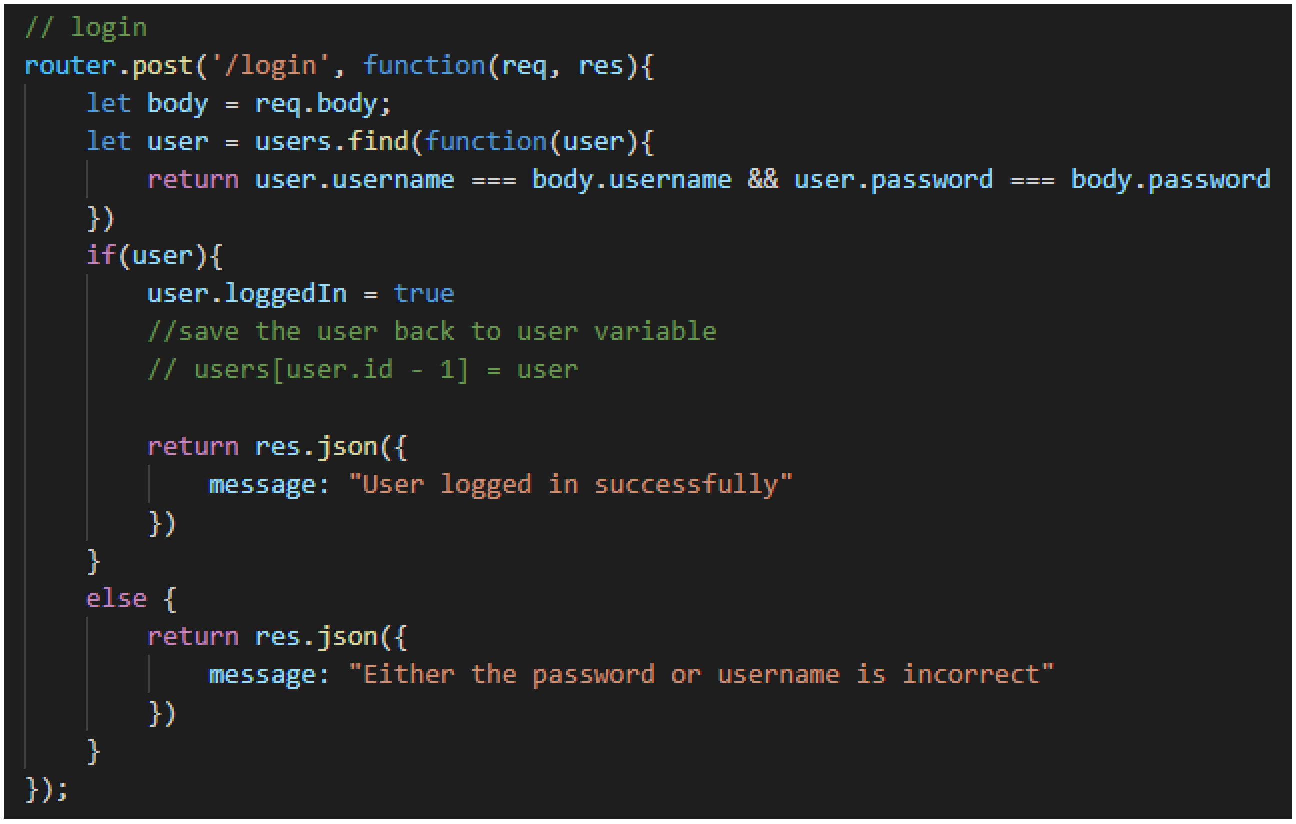


```
  it("it should login a user", (done) => {
    chai
      .request("http://localhost:3000")
      .post("/user/login")
      .send({
        username: "micheal",
        password: "juwon%$#@",
      })
      .end();
      chai
      .request("http://localhost:3000")
      .get("/user/all")
      .end((err, res) => {
        // console.log(res.body any age, "res");
        const users = res.body.message;
        const user = users.find((user) => user.username === "micheal" && user.password === "juwon%$#@
        assert.equal(user.loggedIn, true);
        done();
      });
  });
```

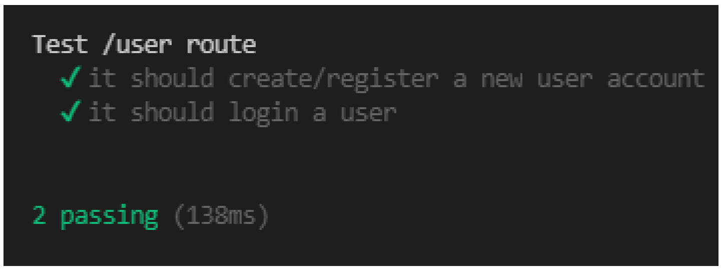


```
  Test /user route
    ✓ it should create/register a new user account
    1) it should login a user


  1 passing (119ms)
  1 failing

  1) Test /user route
       it should login a user:
```

AFTER:



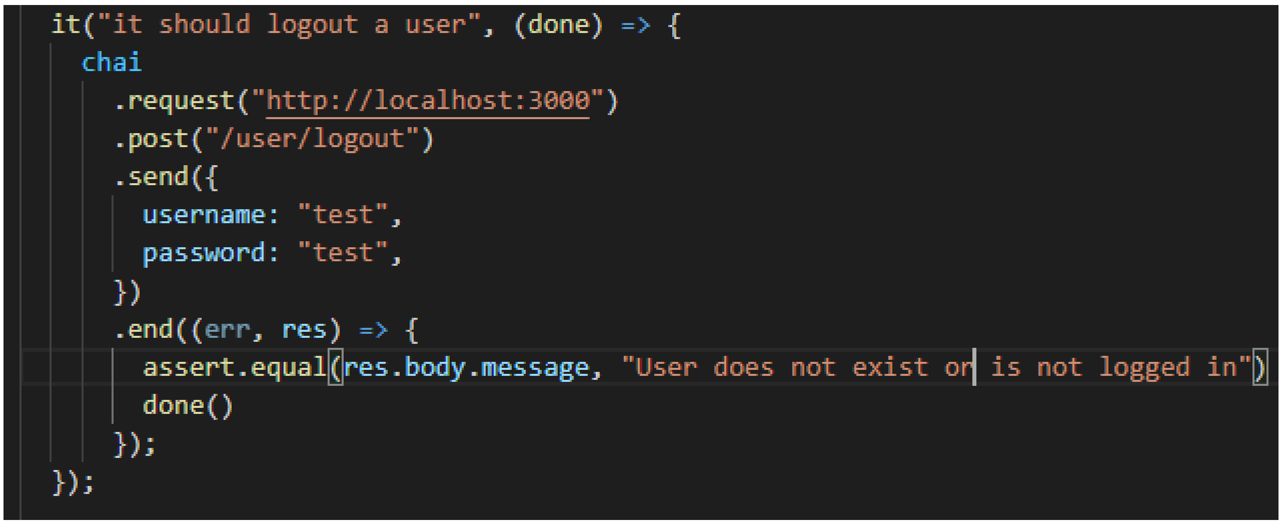
```
// login
router.post('/login', function(req, res){
    let body = req.body;
    let user = users.find(function(user){
        return user.username === body.username && user.password === body.password
    })
    if(user){
        user.loggedIn = true
        //save the user back to user variable
        // users[user.id - 1] = user

        return res.json({
            message: "User logged in successfully"
        })
    }
    else {
        return res.json({
            message: "Either the password or username is incorrect"
        })
    }
});
```



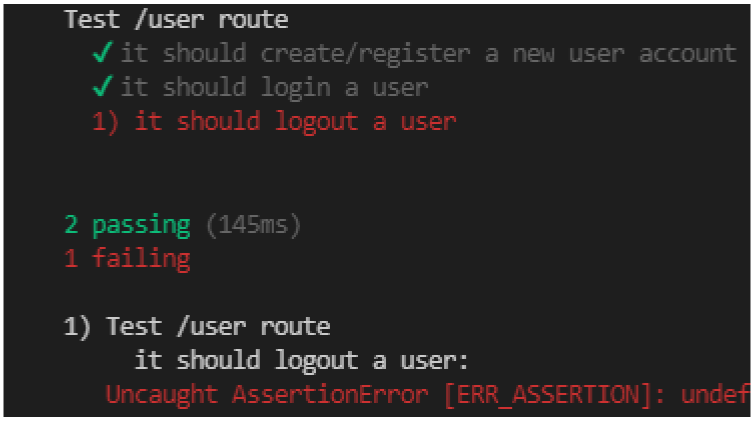
```
Test /user route
  ✓ it should create/register a new user account
  ✓ it should login a user


2 passing (138ms)
```

- ## Test 3 - Logout user

  This test ensures that the user not logged-in is not logged-out and the logged-out user is logged-in.
  BEFORE:

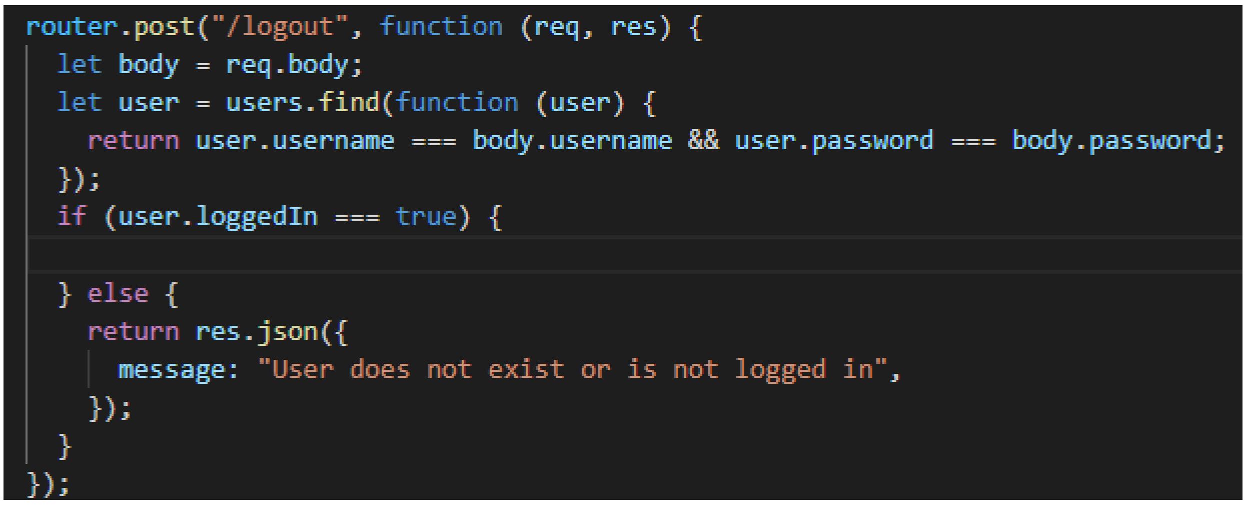


```
it("it should logout a user", (done) => {
  chai
    .request("http://localhost:3000")
    .post("/user/logout")
    .send({
      username: "test",
      password: "test",
    })
    .end((err, res) => {
      assert.equal(res.body.message, "User does not exist or is not logged in")
      done()
    });
});
```

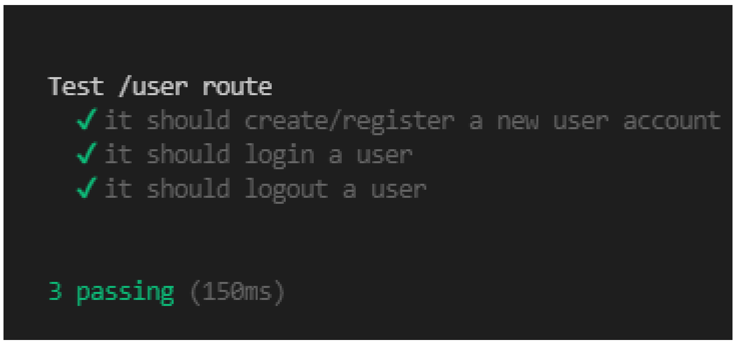


```
Test /user route
  ✓ it should create/register a new user account
  ✓ it should login a user
  1) it should logout a user


2 passing (145ms)
1 failing

1) Test /user route
     it should logout a user:
   Uncaught AssertionError [ERR_ASSERTION]: undef
```

AFTER:

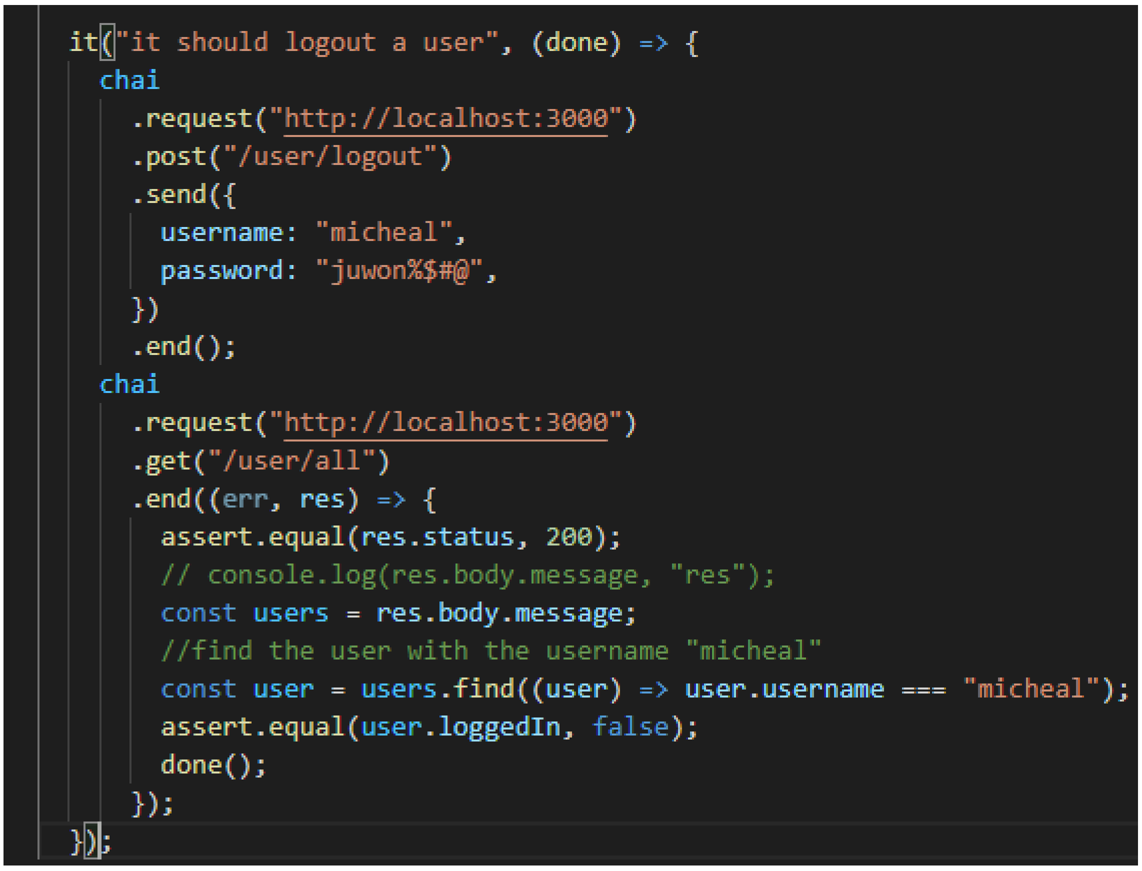


```
router.post("/logout", function (req, res) {
  let body = req.body;
  let user = users.find(function (user) {
    return user.username === body.username && user.password === body.password;
  });
  if (user.loggedIn === true) {

  } else {
    return res.json({
      message: "User does not exist or is not logged in",
    });
  }
});
```

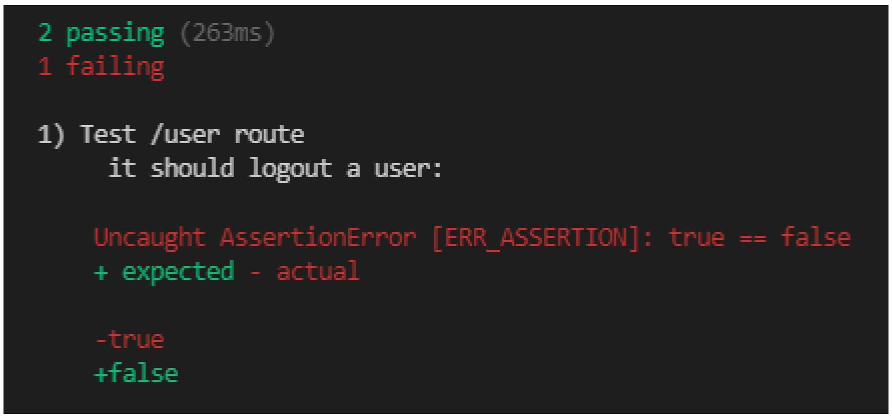


```
Test /user route
  ✓ it should create/register a new user account
  ✓ it should login a user
  ✓ it should logout a user


3 passing (150ms)
```

BEFORE:

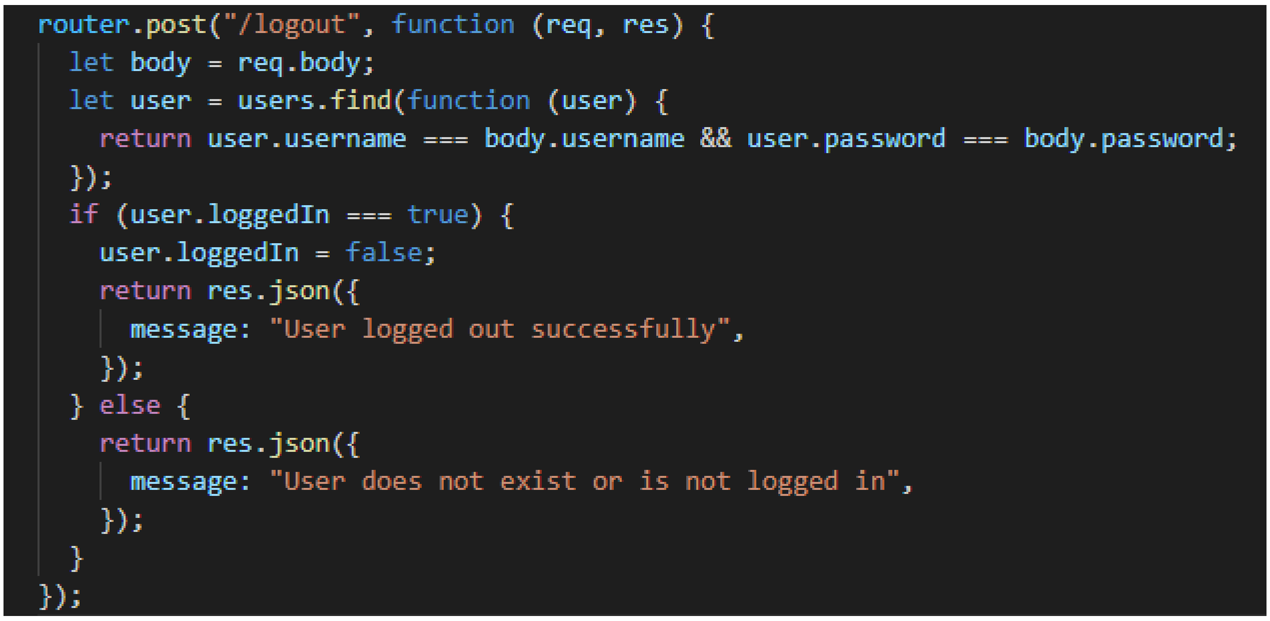


```
  it("it should logout a user", (done) => {
    chai
      .request("http://localhost:3000")
      .post("/user/logout")
      .send({
        username: "micheal",
        password: "juwon%$#@",
      })
      .end();
    chai
      .request("http://localhost:3000")
      .get("/user/all")
      .end((err, res) => {
        assert.equal(res.status, 200);
        // console.log(res.body.message, "res");
        const users = res.body.message;
        //find the user with the username "micheal"
        const user = users.find((user) => user.username === "micheal");
        assert.equal(user.loggedIn, false);
        done();
      });
  });
```

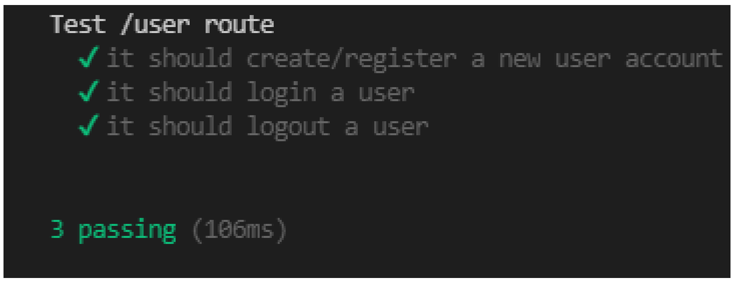


```
2 passing (263ms)
1 failing

1) Test /user route
     it should logout a user:

    Uncaught AssertionError [ERR_ASSERTION]: true == false
    + expected - actual

    -true
    +false
```

AFTER:



```
router.post("/logout", function (req, res) {
  let body = req.body;
  let user = users.find(function (user) {
    return user.username === body.username && user.password === body.password;
  });
  if (user.loggedIn === true) {
    user.loggedIn = false;
    return res.json({
      message: "User logged out successfully",
    });
  } else {
    return res.json({
      message: "User does not exist or is not logged in",
    });
  }
});
```



```
  Test /user route
    ✓ it should create/register a new user account
    ✓ it should login a user
    ✓ it should logout a user


  3 passing (106ms)
```

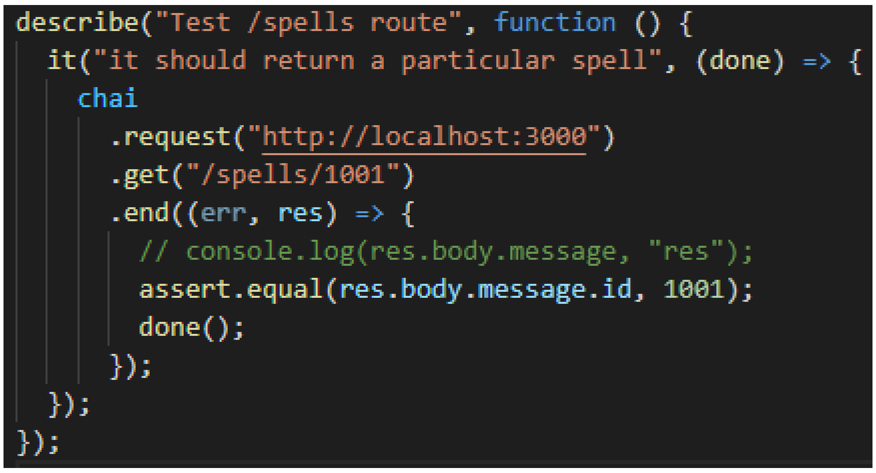
Test set 2 - /spells route

These sets of tests approach testing /spells route to ensure that spells can be added to the server without duplicates, /spells/:id return a spell with the ID so that after adding a spell to the server it can be confirmed that it exists.

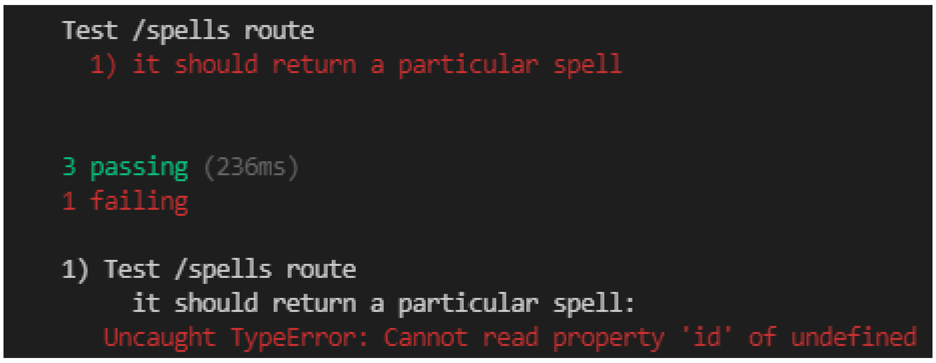
- ## Test 1 - Fetch a specific spell (Get)

  The test was firstly written, and it failed because the function hasn't been implemented.
  A spell was created in the server by default, so that there'll be some spells that this endpoint can fetch.

BEFORE:

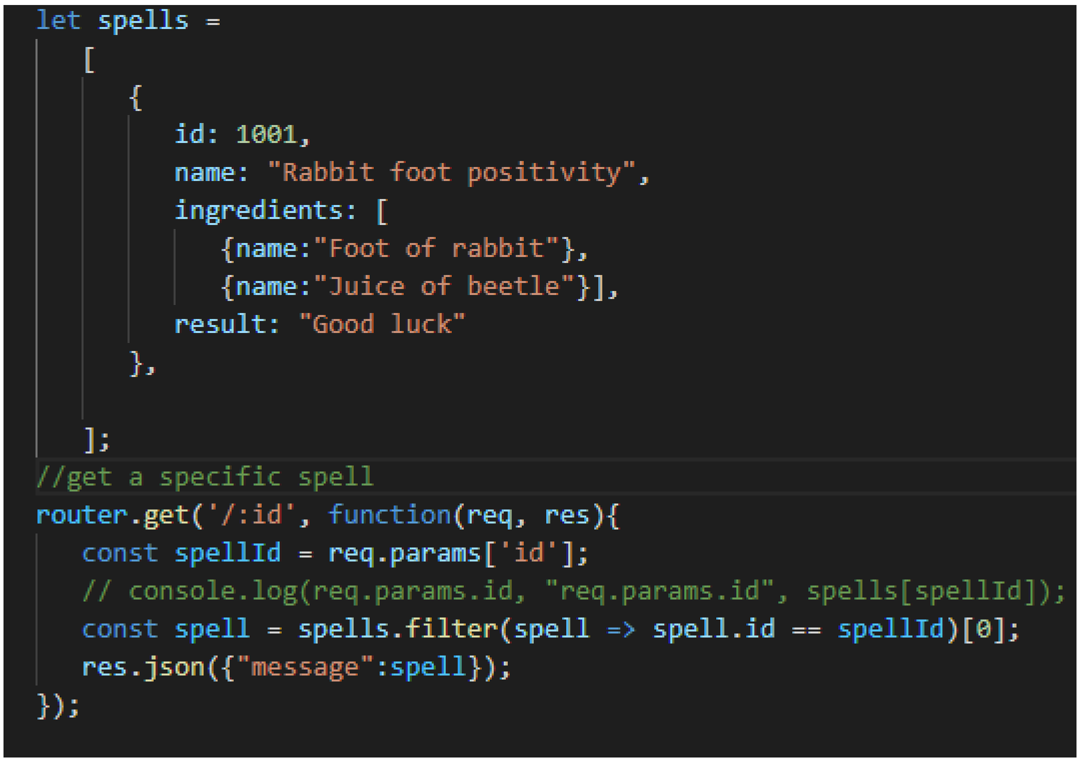


```
describe("Test /spells route", function () {
  it("it should return a particular spell", (done) => {
    chai
      .request("http://localhost:3000")
      .get("/spells/1001")
      .end((err, res) => {
        // console.log(res.body.message, "res");
        assert.equal(res.body.message.id, 1001);
        done();
      });
  });
});
```

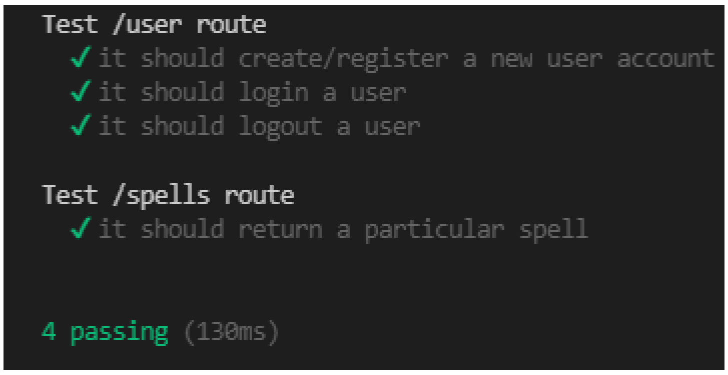


```
  Test /spells route
    1) it should return a particular spell


  3 passing (236ms)
  1 failing

  1) Test /spells route
       it should return a particular spell:
     Uncaught TypeError: Cannot read property 'id' of undefined
```

AFTER:



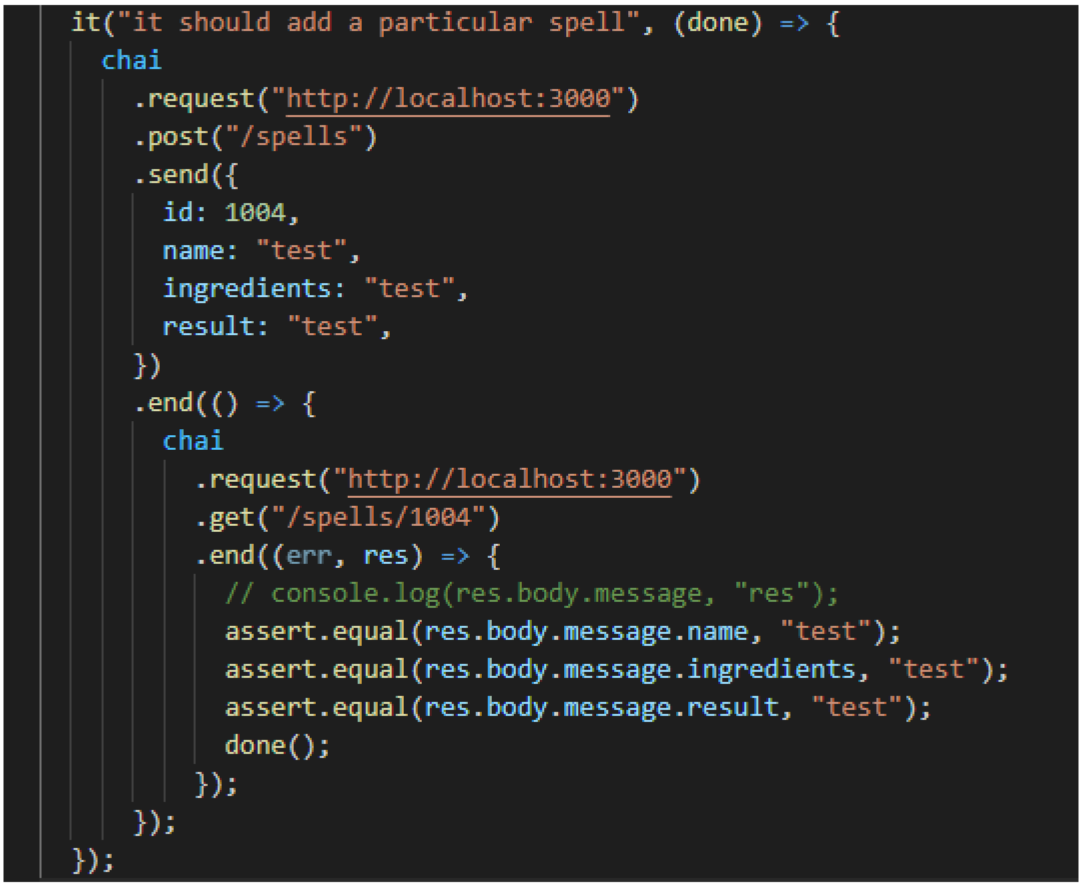
```
let spells =
    [
      {
        id: 1001,
        name: "Rabbit foot positivity",
        ingredients: [
          {name:"Foot of rabbit"},
          {name:"Juice of beetle"}],
        result: "Good luck"
      },

    ];
//get a specific spell
router.get('/:id', function(req, res){
  const spellId = req.params['id'];
  // console.log(req.params.id, "req.params.id", spells[spellId]);
  const spell = spells.filter(spell => spell.id == spellId)[0];
  res.json({"message":spell});
});
```

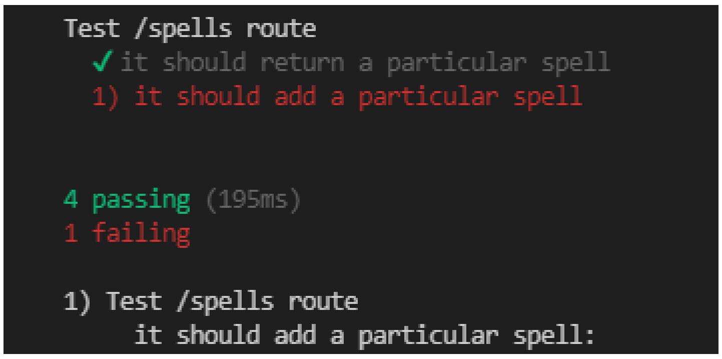

```
Test /user route
  ✓ it should create/register a new user account
  ✓ it should login a user
  ✓ it should logout a user

Test /spells route
  ✓ it should return a particular spell


4 passing (130ms)
```

- ## Test 2 - Add a spell (POST)

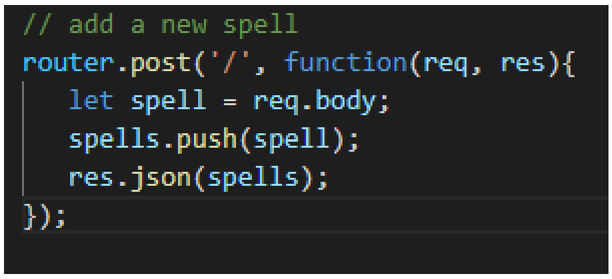
  This test checks that all the data added are all retrieved via the endpoint of test 1 above.

  BEFORE:

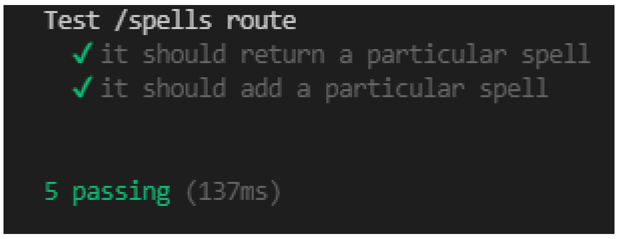

```
it("it should add a particular spell", (done) => {
  chai
    .request("http://localhost:3000")
    .post("/spells")
    .send({
      id: 1004,
      name: "test",
      ingredients: "test",
      result: "test",
    })
    .end(() => {
      chai
        .request("http://localhost:3000")
        .get("/spells/1004")
        .end((err, res) => {
          // console.log(res.body.message, "res");
          assert.equal(res.body.message.name, "test");
          assert.equal(res.body.message.ingredients, "test");
          assert.equal(res.body.message.result, "test");
          done();
        });
    });
});
```


```
Test /spells route
  ✓ it should return a particular spell
  1) it should add a particular spell


4 passing (195ms)
1 failing

1) Test /spells route
     it should add a particular spell:
```

AFTER:



```
// add a new spell
router.post('/', function(req, res){
    let spell = req.body;
    spells.push(spell);
    res.json(spells);
});
```

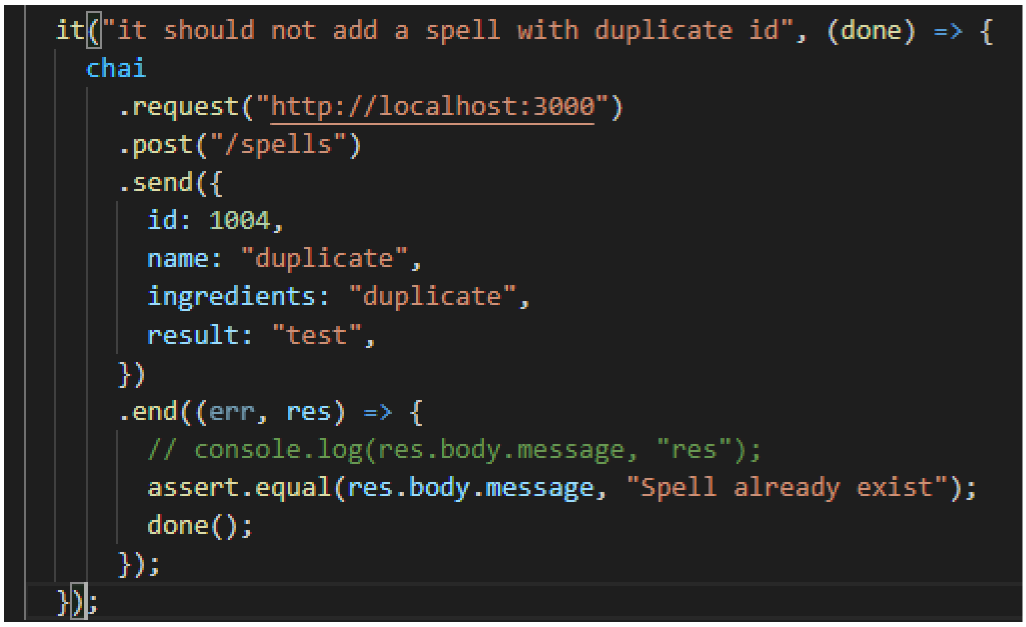


```
  Test /spells route
    ✓ it should return a particular spell
    ✓ it should add a particular spell


  5 passing (137ms)
```

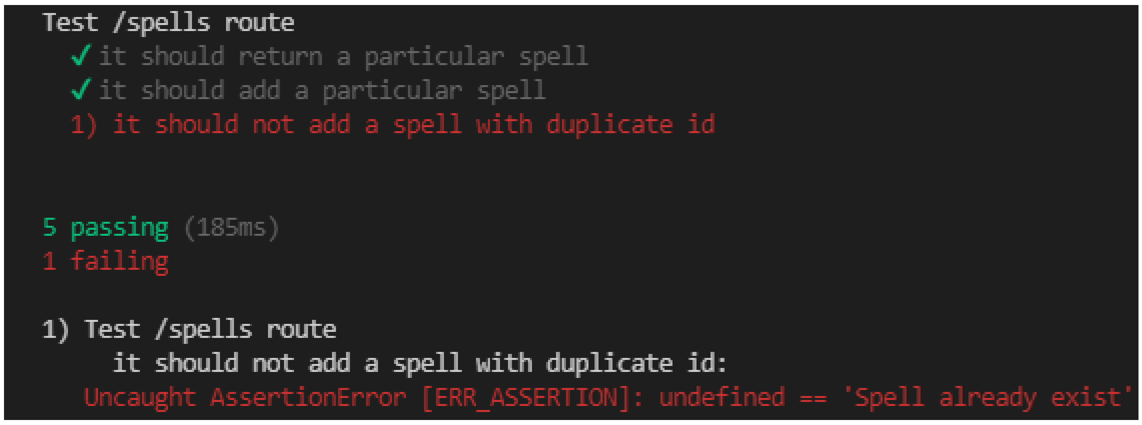
- Test 3 - Should not add a spell with duplicate ID (POST)

A spell with the same ID above was repeated, this test ensures that the server rejects it, and it has the right response message.

BEFORE:

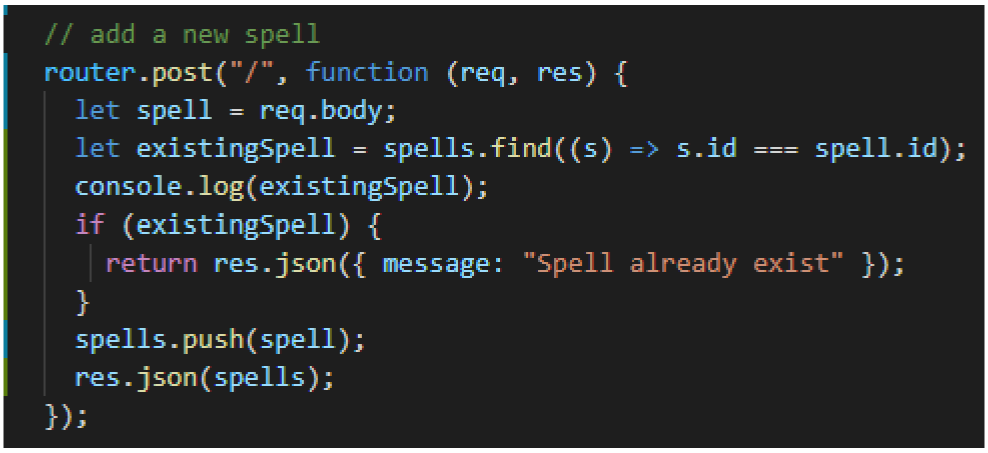


```
  it("it should not add a spell with duplicate id", (done) => {
    chai
      .request("http://localhost:3000")
      .post("/spells")
      .send({
        id: 1004,
        name: "duplicate",
        ingredients: "duplicate",
        result: "test",
      })
      .end((err, res) => {
        // console.log(res.body.message, "res");
        assert.equal(res.body.message, "Spell already exist");
        done();
      });
  });
```

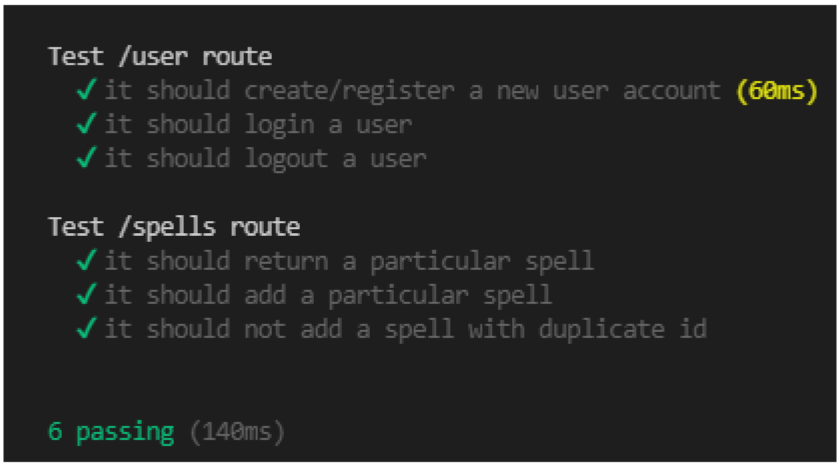


```
  Test /spells route
    ✓ it should return a particular spell
    ✓ it should add a particular spell
    1) it should not add a spell with duplicate id


  5 passing (185ms)
  1 failing

  1) Test /spells route
       it should not add a spell with duplicate id:
     Uncaught AssertionError [ERR_ASSERTION]: undefined == 'Spell already exist'
```

AFTER:



```
// add a new spell
router.post("/", function (req, res) {
  let spell = req.body;
  let existingSpell = spells.find((s) => s.id === spell.id);
  console.log(existingSpell);
  if (existingSpell) {
    return res.json({ message: "Spell already exist" });
  }
  spells.push(spell);
  res.json(spells);
});
```



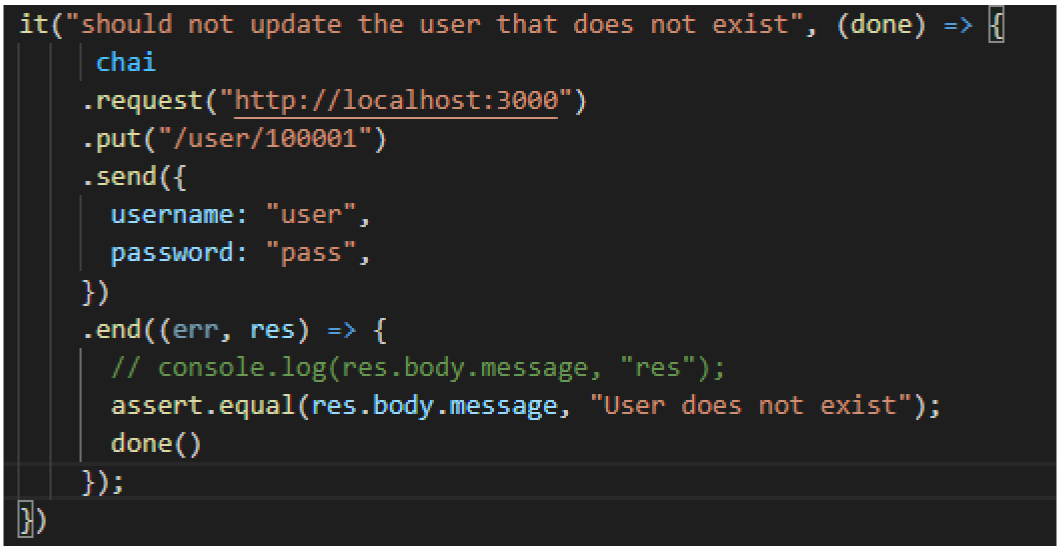
```
  Test /user route
    ✓ it should create/register a new user account (60ms)
    ✓ it should login a user
    ✓ it should logout a user

  Test /spells route
    ✓ it should return a particular spell
    ✓ it should add a particular spell
    ✓ it should not add a spell with duplicate id


  6 passing (140ms)
```

Test set 3 - /user/:id route (UPDATE, PUT)

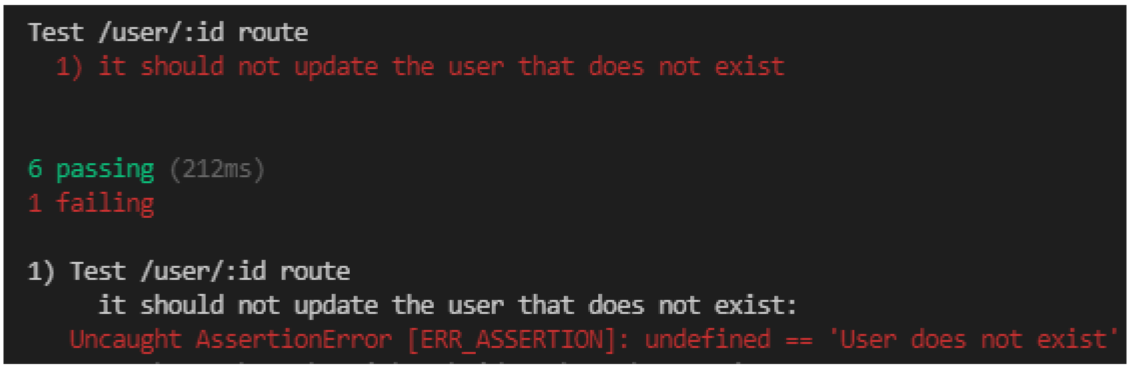
The /user/:id API path is only tested with the update method following a test driven development approach; A test the fails was firstly written and a code that pass the test followed, this cycle revolves on only one function *‘router.put(/:id …)’* ensuring that a user that does not exist is not updated, avoiding modifying a user details using incorrect password and finally editing the username of user with the right privilege(password)

- Test 1 - Should not update the user that does not exist

BEFORE:

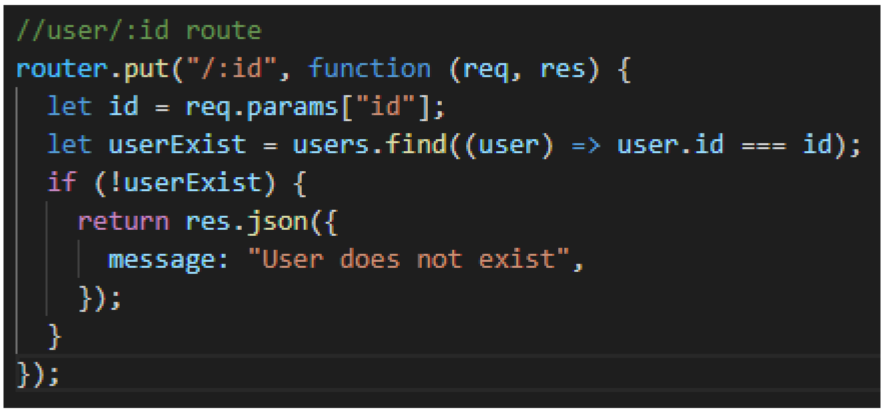


```
it("should not update the user that does not exist", (done) => {
    chai
    .request("http://localhost:3000")
    .put("/user/100001")
    .send({
      username: "user",
      password: "pass",
    })
    .end((err, res) => {
      // console.log(res.body.message, "res");
      assert.equal(res.body.message, "User does not exist");
      done()
    });
})
```

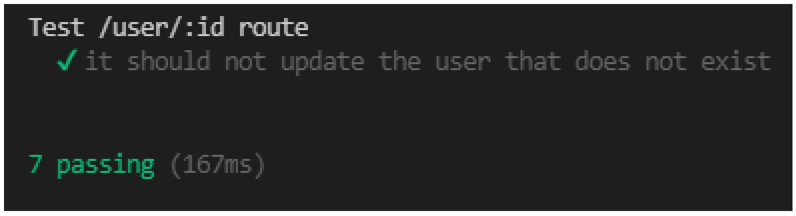


```
Test /user/:id route
  1) it should not update the user that does not exist


6 passing (212ms)
1 failing

1) Test /user/:id route
     it should not update the user that does not exist:
   Uncaught AssertionError [ERR_ASSERTION]: undefined == 'User does not exist'
```

AFTER:



```
//user/:id route
router.put("/:id", function (req, res) {
  let id = req.params["id"];
  let userExist = users.find((user) => user.id === id);
  if (!userExist) {
    return res.json({
      message: "User does not exist",
    });
  }
});
```

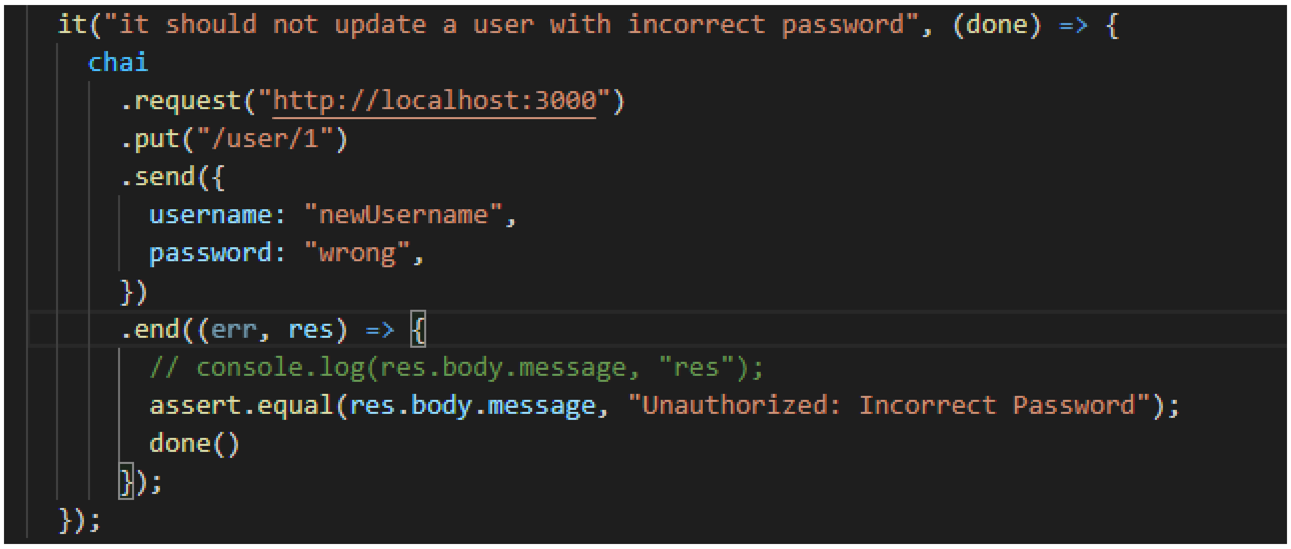


```
Test /user/:id route
  ✓ it should not update the user that does not exist


7 passing (167ms)
```

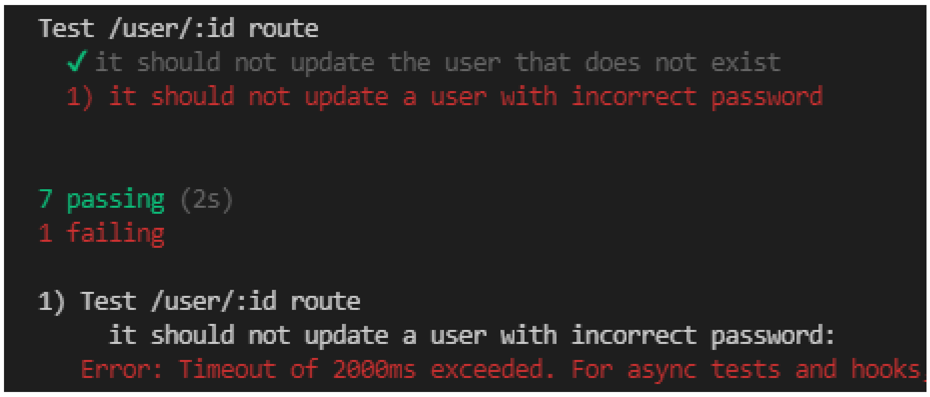
- Test 2 - Should not update a user with incorrect password

  ID of a user (like a token) was used to find user and the test check if it has the correct password

  BEFORE:

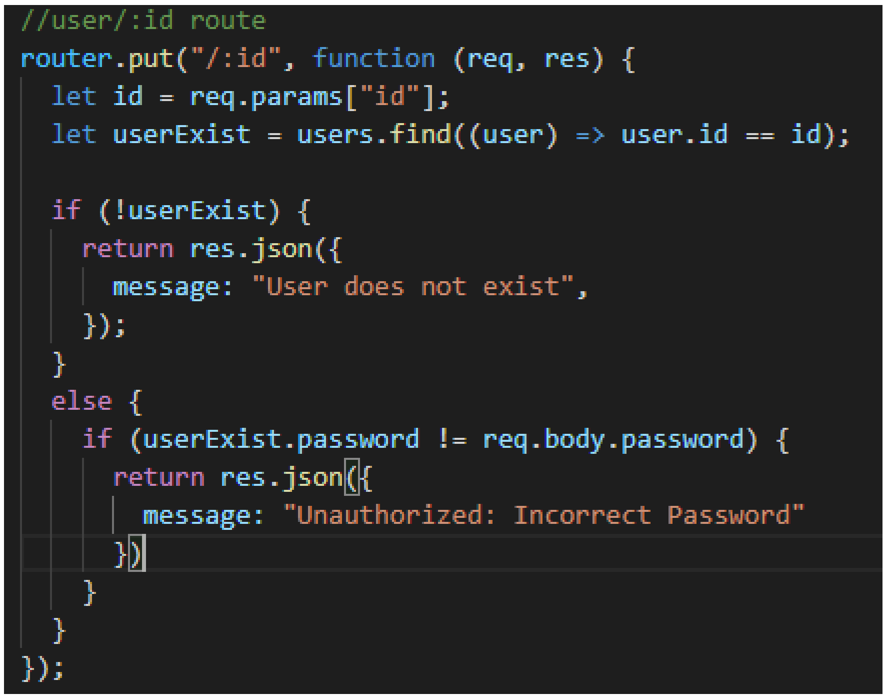


```
it("it should not update a user with incorrect password", (done) => {
  chai
    .request("http://localhost:3000")
    .put("/user/1")
    .send({
      username: "newUsername",
      password: "wrong",
    })
    .end((err, res) => {
      // console.log(res.body.message, "res");
      assert.equal(res.body.message, "Unauthorized: Incorrect Password");
      done()
    });
});
```

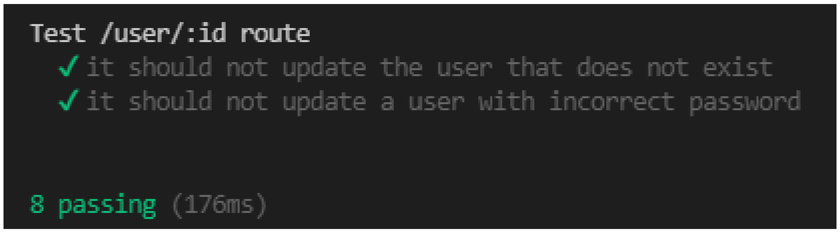


```
Test /user/:id route
  ✓ it should not update the user that does not exist
  1) it should not update a user with incorrect password


7 passing (2s)
1 failing

1) Test /user/:id route
     it should not update a user with incorrect password:
   Error: Timeout of 2000ms exceeded. For async tests and hooks
```

AFTER:



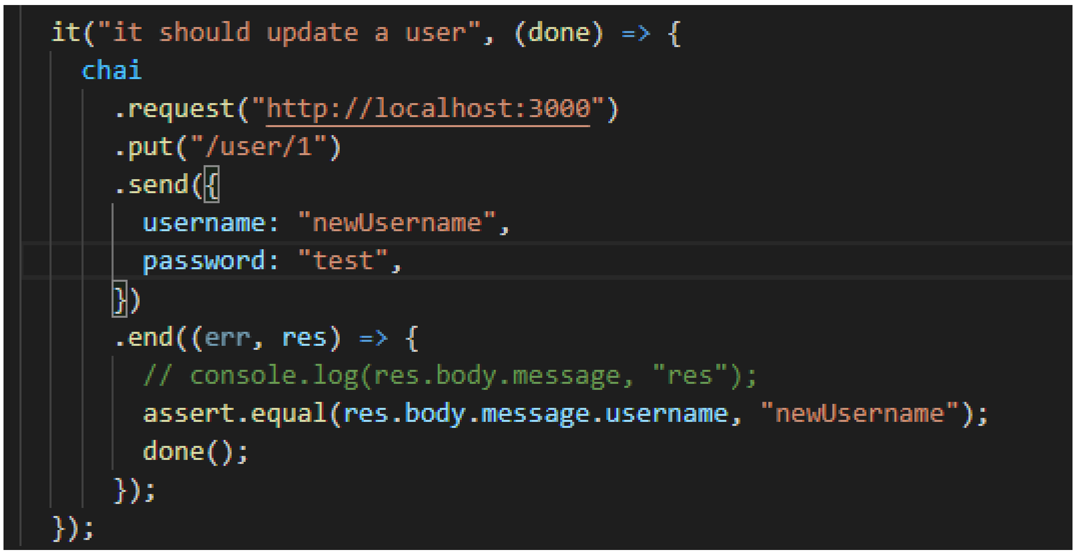
```
//user/:id route
router.put("/:id", function (req, res) {
  let id = req.params["id"];
  let userExist = users.find((user) => user.id == id);

  if (!userExist) {
    return res.json({
      message: "User does not exist",
    });
  }
  else {
    if (userExist.password != req.body.password) {
      return res.json({
        message: "Unauthorized: Incorrect Password"
      })
    }
  }
});
```



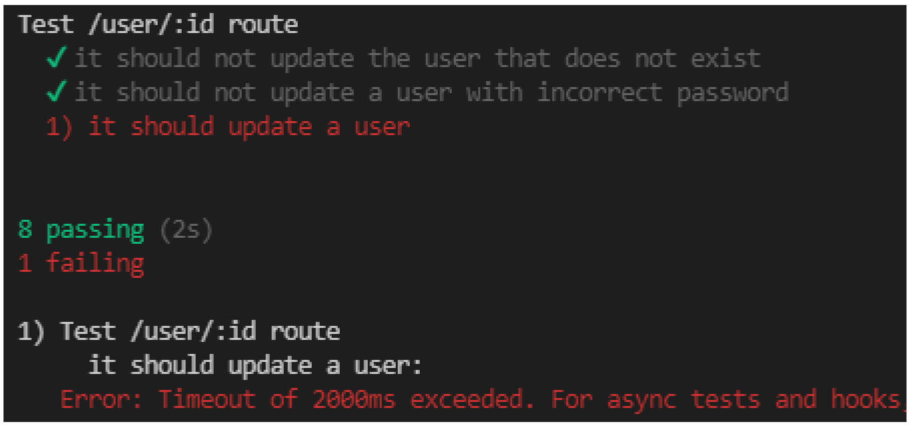
```
Test /user/:id route
  ✓ it should not update the user that does not exist
  ✓ it should not update a user with incorrect password


8 passing (176ms)
```

- Test 3 - Should update a user with correct password

  BEFORE:

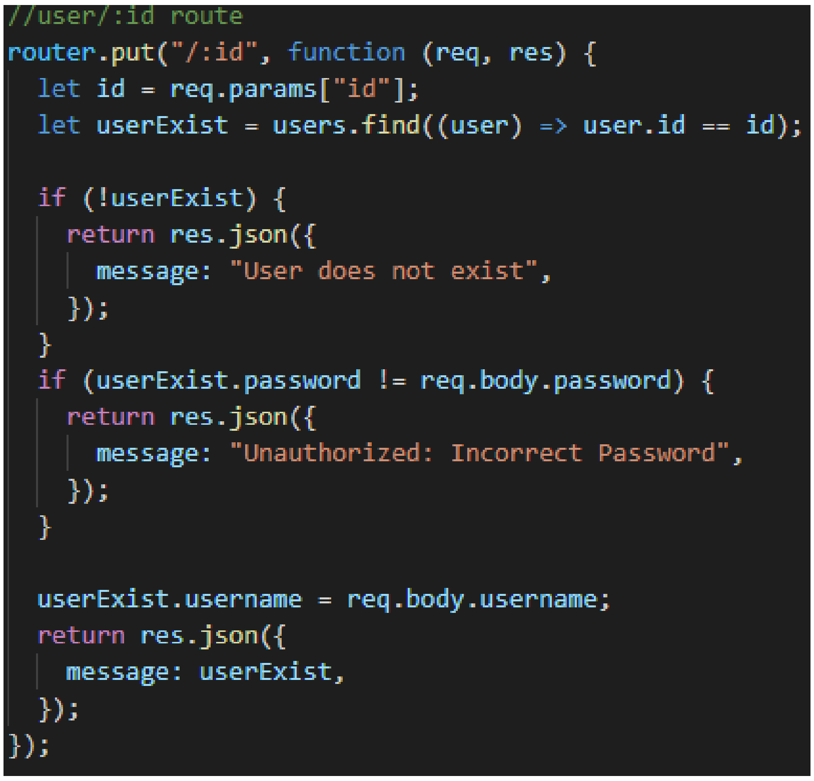


```
it("it should update a user", (done) => {
  chai
    .request("http://localhost:3000")
    .put("/user/1")
    .send({
      username: "newUsername",
      password: "test",
    })
    .end((err, res) => {
      // console.log(res.body.message, "res");
      assert.equal(res.body.message.username, "newUsername");
      done();
    });
});
```

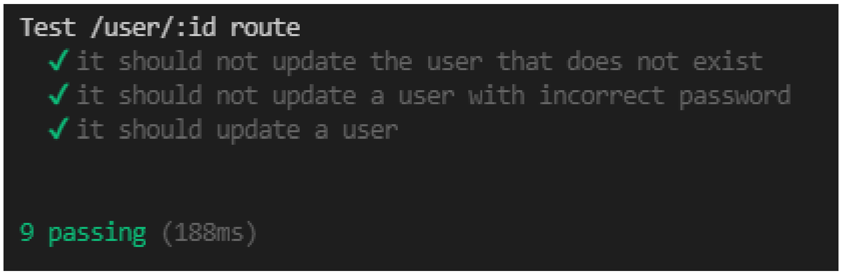


```
Test /user/:id route
  ✓ it should not update the user that does not exist
  ✓ it should not update a user with incorrect password
  1) it should update a user


8 passing (2s)
1 failing

1) Test /user/:id route
     it should update a user:
   Error: Timeout of 2000ms exceeded. For async tests and hooks,
```

AFTER:



```
//user/:id route
router.put("/:id", function (req, res) {
  let id = req.params["id"];
  let userExist = users.find((user) => user.id == id);

  if (!userExist) {
    return res.json({
      message: "User does not exist",
    });
  }
  if (userExist.password != req.body.password) {
    return res.json({
      message: "Unauthorized: Incorrect Password",
    });
  }

  userExist.username = req.body.username;
  return res.json({
    message: userExist,
  });
});
```



```
Test /user/:id route
  ✓ it should not update the user that does not exist
  ✓ it should not update a user with incorrect password
  ✓ it should update a user


9 passing (188ms)
```