UNIT TESTING G1 PET ADOPTION SYSTEM – HappyTails

Testing Framework: Mocha@10.8.2

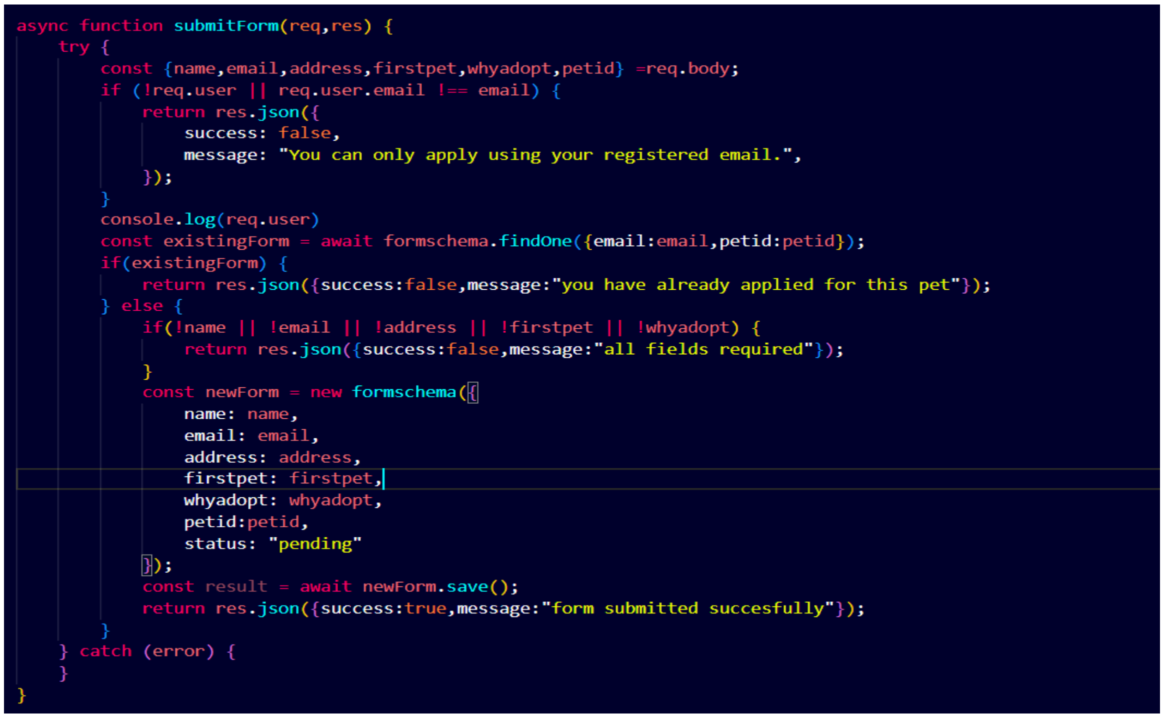
Assertion library used: Chai@4.3.4

Other: Sinon@19.0.2

FormControllers

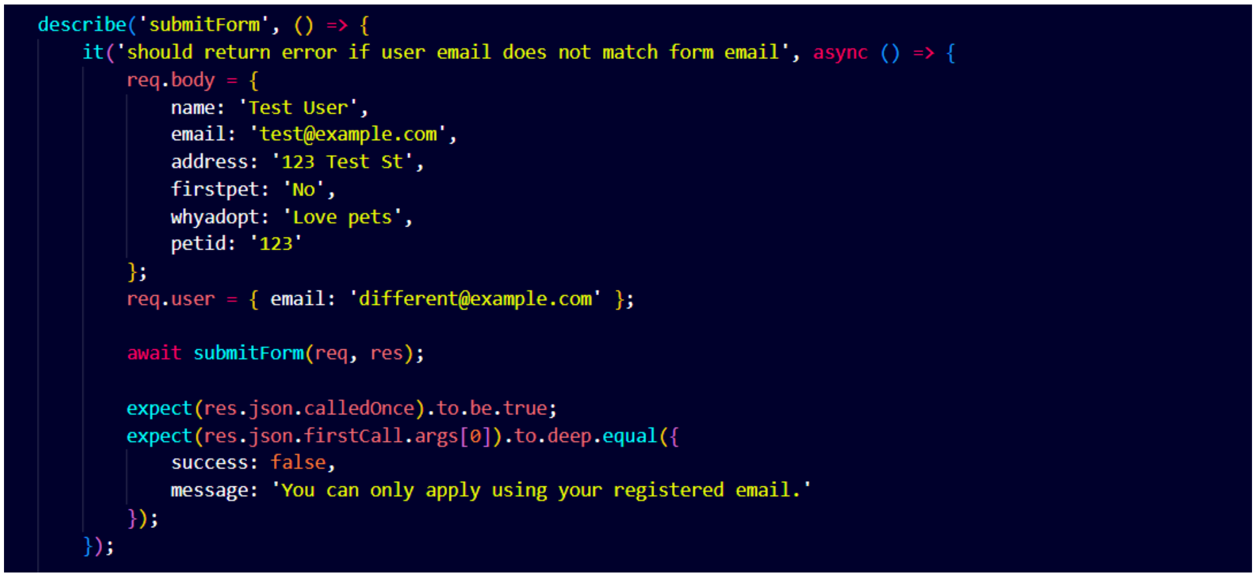
a. Submitform:

This function handles the submission of a form on the website.



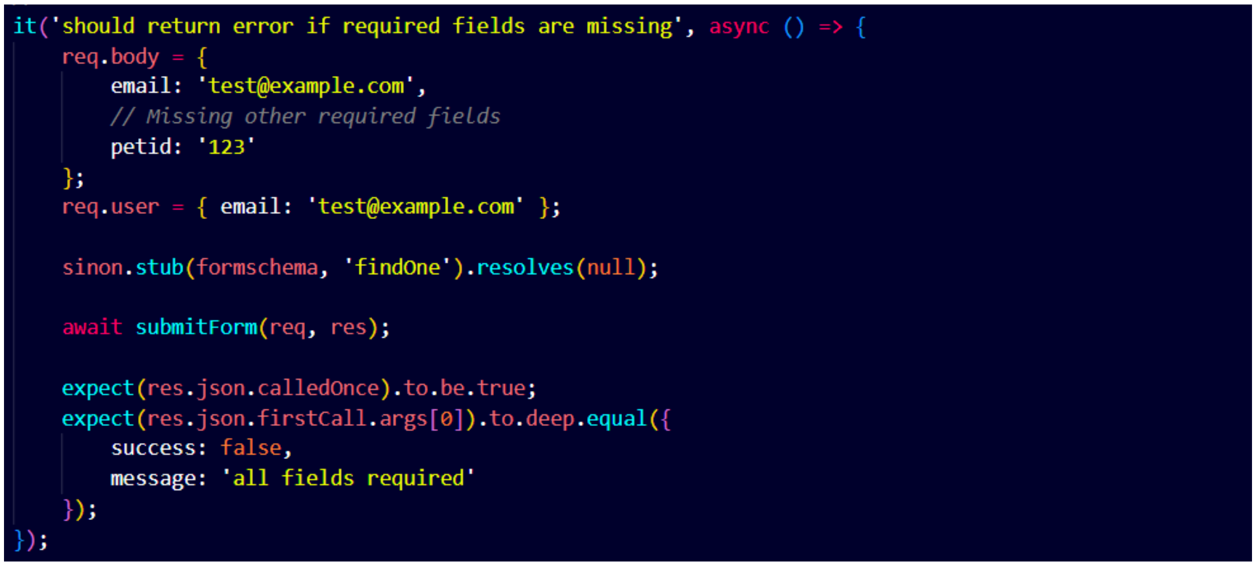
```
async function submitForm(req,res) {
    try {
        const {name,email,address,firstpet,whyadopt,petid} =req.body;
        if (!req.user || req.user.email !== email) {
            return res.json({
                success: false,
                message: "You can only apply using your registered email.",
            });
        }
        console.log(req.user)
        const existingForm = await formschema.findOne({email:email,petid:petid});
        if(existingForm) {
            return res.json({success:false,message:"you have already applied for this pet"});
        } else {
            if(!name || !email || !address || !firstpet || !whyadopt) {
                return res.json({success:false,message:"all fields required"});
            }
            const newForm = new formschema({
                name: name,
                email: email,
                address: address,
                firstpet: firstpet,
                whyadopt: whyadopt,
                petid:petid,
                status: "pending"
            });
            const result = await newForm.save();
            return res.json({success:true,message:"form submitted succesfully"});
        }
    } catch (error) {
    }
}
```

Test Cases:



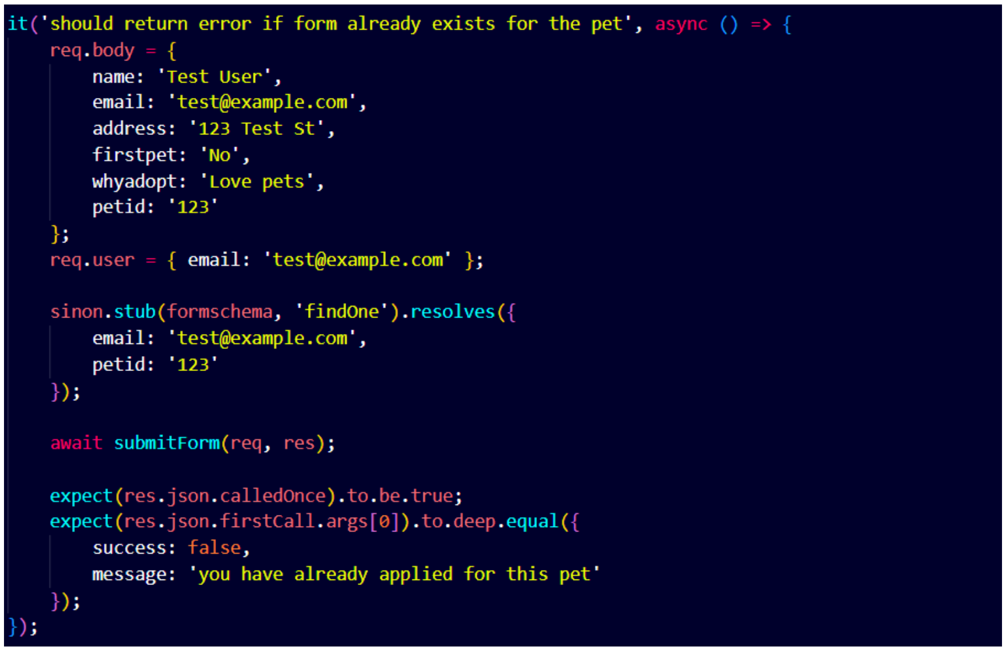
```javascript
describe('submitForm', () => {
    it('should return error if user email does not match form email', async () => {
        req.body = {
            name: 'Test User',
            email: 'test@example.com',
            address: '123 Test St',
            firstpet: 'No',
            whyadopt: 'Love pets',
            petid: '123'
        };
        req.user = { email: 'different@example.com' };

        await submitForm(req, res);

        expect(res.json.calledOnce).to.be.true;
        expect(res.json.firstCall.args[0]).to.deep.equal({
            success: false,
            message: 'You can only apply using your registered email.'
        });
    });
```

Checks if user email matches the registered email, if not returns an error.



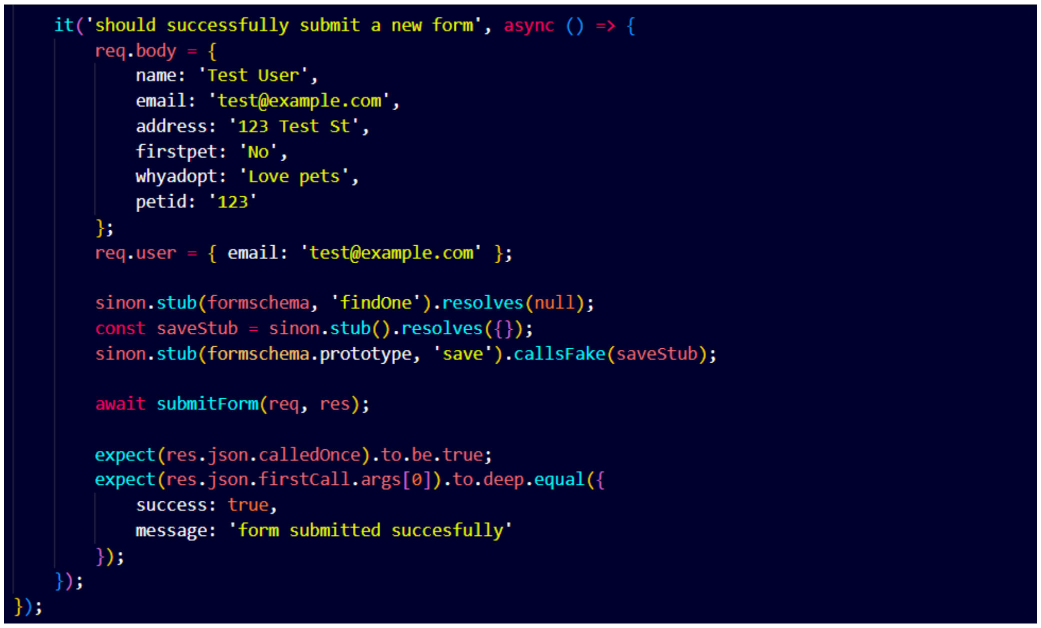
```javascript
it('should return error if required fields are missing', async () => {
    req.body = {
        email: 'test@example.com',
        // Missing other required fields
        petid: '123'
    };
    req.user = { email: 'test@example.com' };

    sinon.stub(formschema, 'findOne').resolves(null);

    await submitForm(req, res);

    expect(res.json.calledOnce).to.be.true;
    expect(res.json.firstCall.args[0]).to.deep.equal({
        success: false,
        message: 'all fields required'
    });
});
```

Verifies that all required fields (like name, address, firstpet) are present in the request body. If any field is missing, it should return an error message saying "all fields required."



```
it('should return error if form already exists for the pet', async () => {
    req.body = {
        name: 'Test User',
        email: 'test@example.com',
        address: '123 Test St',
        firstpet: 'No',
        whyadopt: 'Love pets',
        petid: '123'
    };
    req.user = { email: 'test@example.com' };

    sinon.stub(formschema, 'findOne').resolves({
        email: 'test@example.com',
        petid: '123'
    });

    await submitForm(req, res);

    expect(res.json.calledOnce).to.be.true;
    expect(res.json.firstCall.args[0]).to.deep.equal({
        success: false,
        message: 'you have already applied for this pet'
    });
});
```

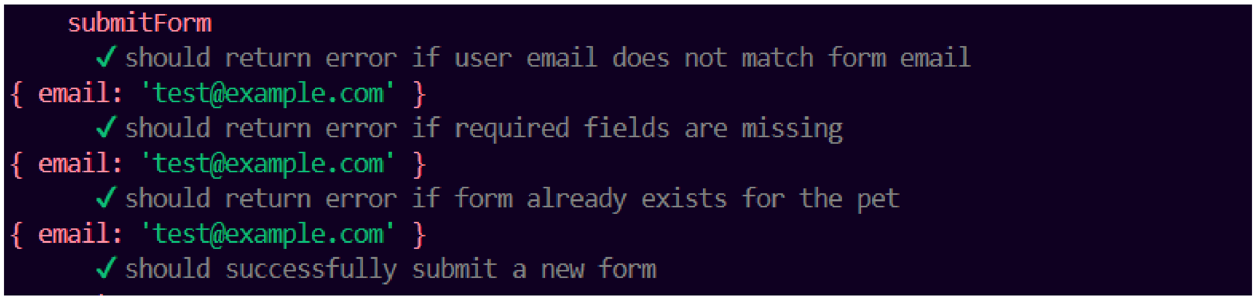
Checks that a user cannot submit the form for a pet that they have already applied for.



```
    it('should successfully submit a new form', async () => {
        req.body = {
            name: 'Test User',
            email: 'test@example.com',
            address: '123 Test St',
            firstpet: 'No',
            whyadopt: 'Love pets',
            petid: '123'
        };
        req.user = { email: 'test@example.com' };

        sinon.stub(formschema, 'findOne').resolves(null);
        const saveStub = sinon.stub().resolves({});
        sinon.stub(formschema.prototype, 'save').callsFake(saveStub);

        await submitForm(req, res);

        expect(res.json.calledOnce).to.be.true;
        expect(res.json.firstCall.args[0]).to.deep.equal({
            success: true,
            message: 'form submitted succesfully'
        });
    });
});
```

Verifies that a new form is saved to the database if all fields are valid and the user has not already applied.

Output:

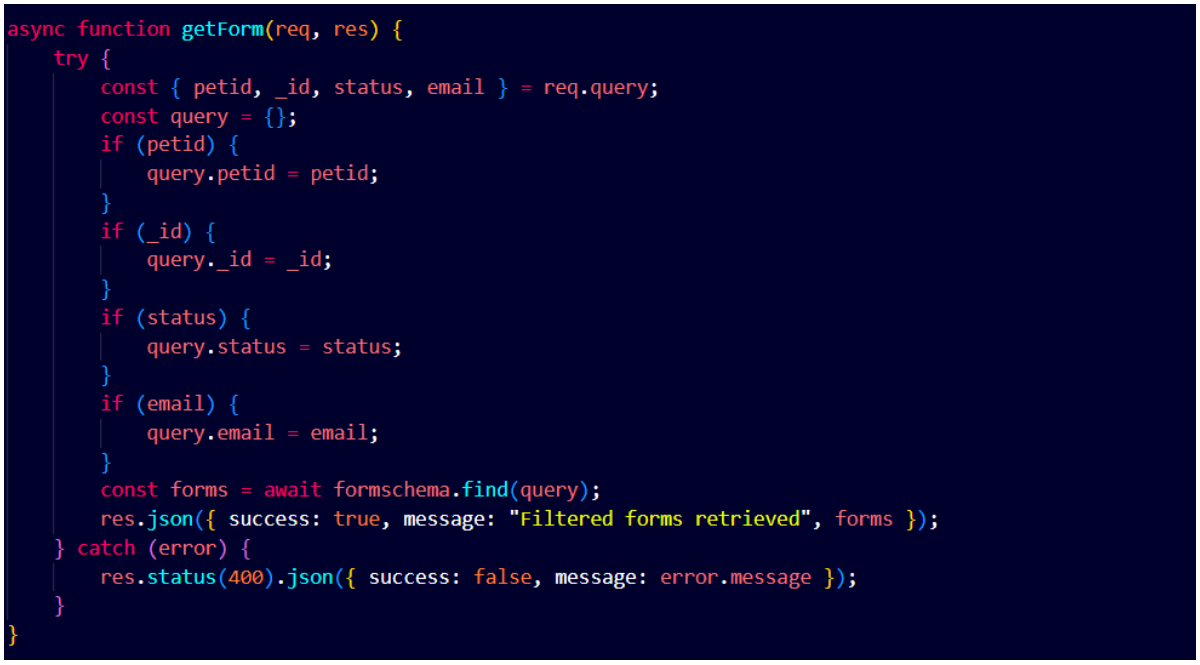


```
    submitForm
      ✓ should return error if user email does not match form email
{ email: 'test@example.com' }
      ✓ should return error if required fields are missing
{ email: 'test@example.com' }
      ✓ should return error if form already exists for the pet
{ email: 'test@example.com' }
      ✓ should successfully submit a new form
```

These functions work together to check all the functions and throw errors if there is a problem and otherwise display these messages in the case of an error.

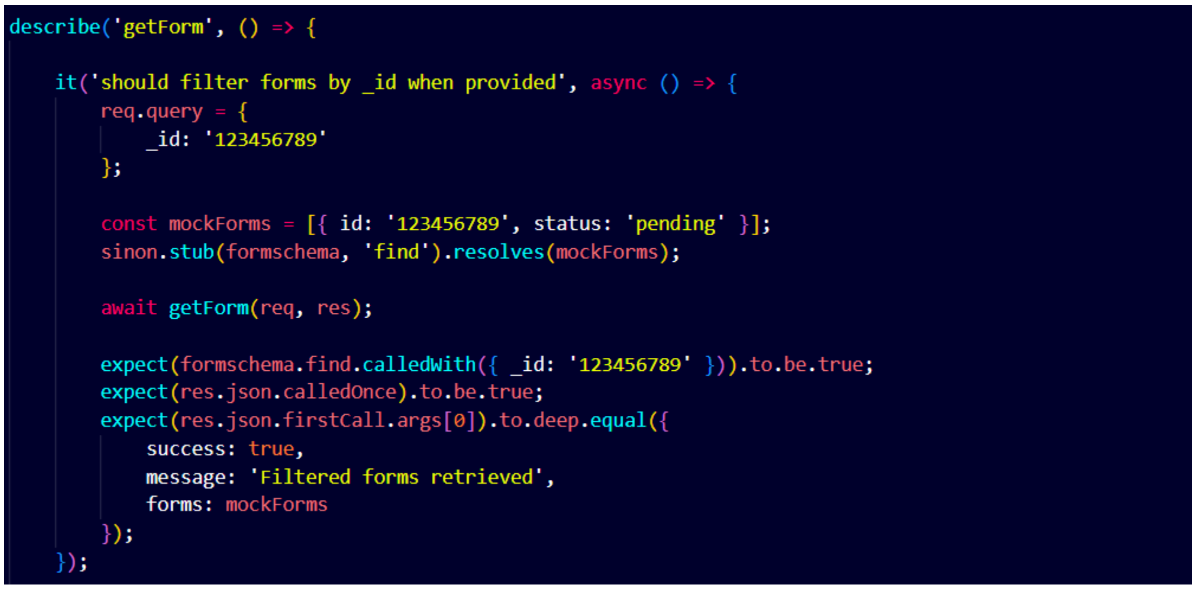
b. getform:

This function retrieves forms based on specific filters.



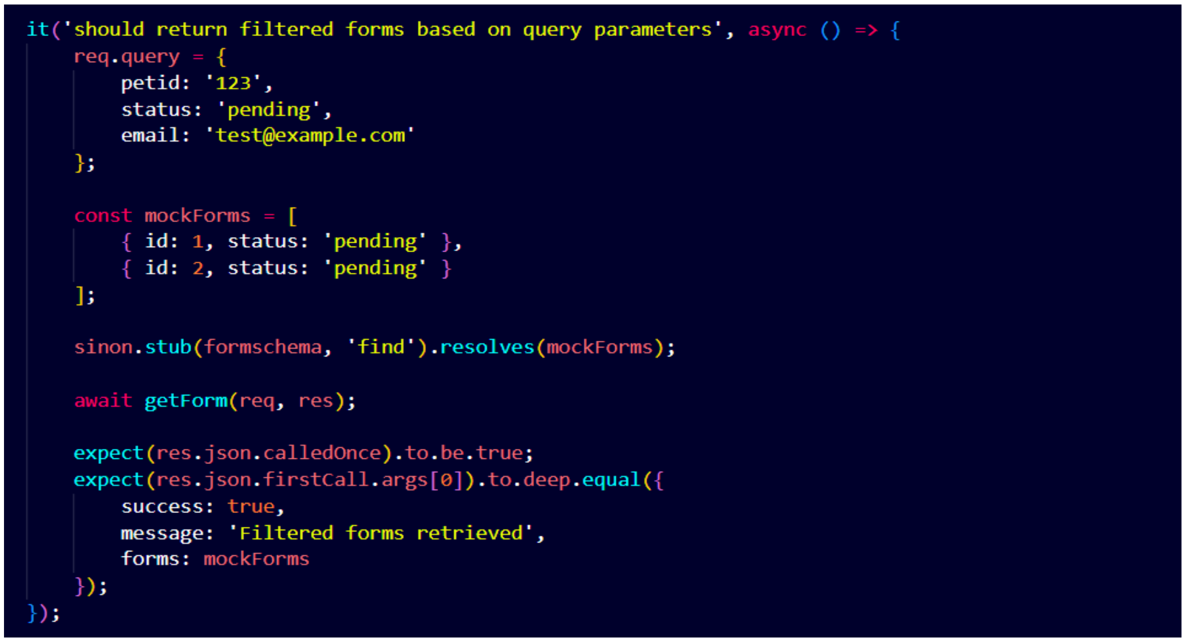
```
async function getForm(req, res) {
    try {
        const { petid, _id, status, email } = req.query;
        const query = {};
        if (petid) {
            query.petid = petid;
        }
        if (_id) {
            query._id = _id;
        }
        if (status) {
            query.status = status;
        }
        if (email) {
            query.email = email;
        }
        const forms = await formschema.find(query);
        res.json({ success: true, message: "Filtered forms retrieved", forms });
    } catch (error) {
        res.status(400).json({ success: false, message: error.message });
    }
}
```

Test cases:



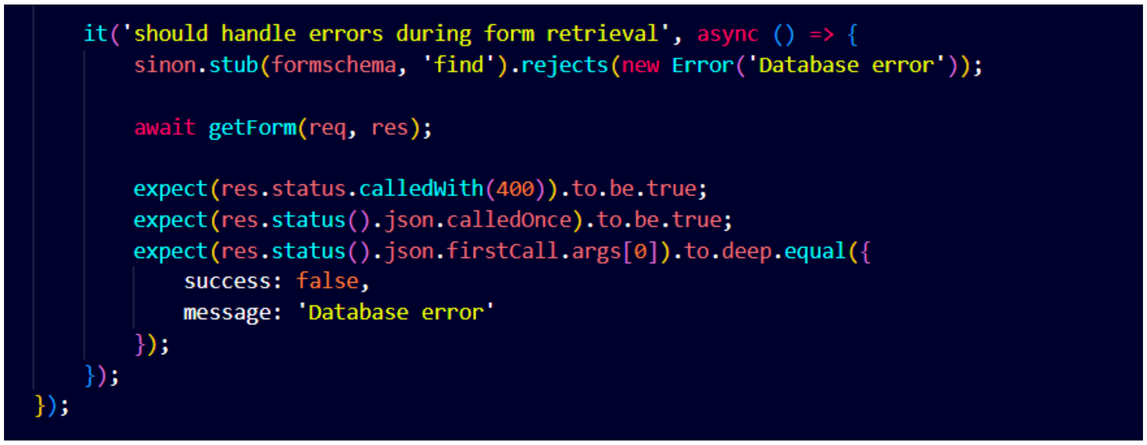
```javascript
describe('getForm', () => {

    it('should filter forms by _id when provided', async () => {
        req.query = {
            _id: '123456789'
        };

        const mockForms = [{ id: '123456789', status: 'pending' }];
        sinon.stub(formschema, 'find').resolves(mockForms);

        await getForm(req, res);

        expect(formschema.find.calledWith({ _id: '123456789' })).to.be.true;
        expect(res.json.calledOnce).to.be.true;
        expect(res.json.firstCall.args[0]).to.deep.equal({
            success: true,
            message: 'Filtered forms retrieved',
            forms: mockForms
        });
    });
```

Tests whether the function retrieves forms with a specific ID, when the ID is provided.



```javascript
it('should return filtered forms based on query parameters', async () => {
    req.query = {
        petid: '123',
        status: 'pending',
        email: 'test@example.com'
    };

    const mockForms = [
        { id: 1, status: 'pending' },
        { id: 2, status: 'pending' }
    ];

    sinon.stub(formschema, 'find').resolves(mockForms);

    await getForm(req, res);

    expect(res.json.calledOnce).to.be.true;
    expect(res.json.firstCall.args[0]).to.deep.equal({
        success: true,
        message: 'Filtered forms retrieved',
        forms: mockForms
    });
});
```

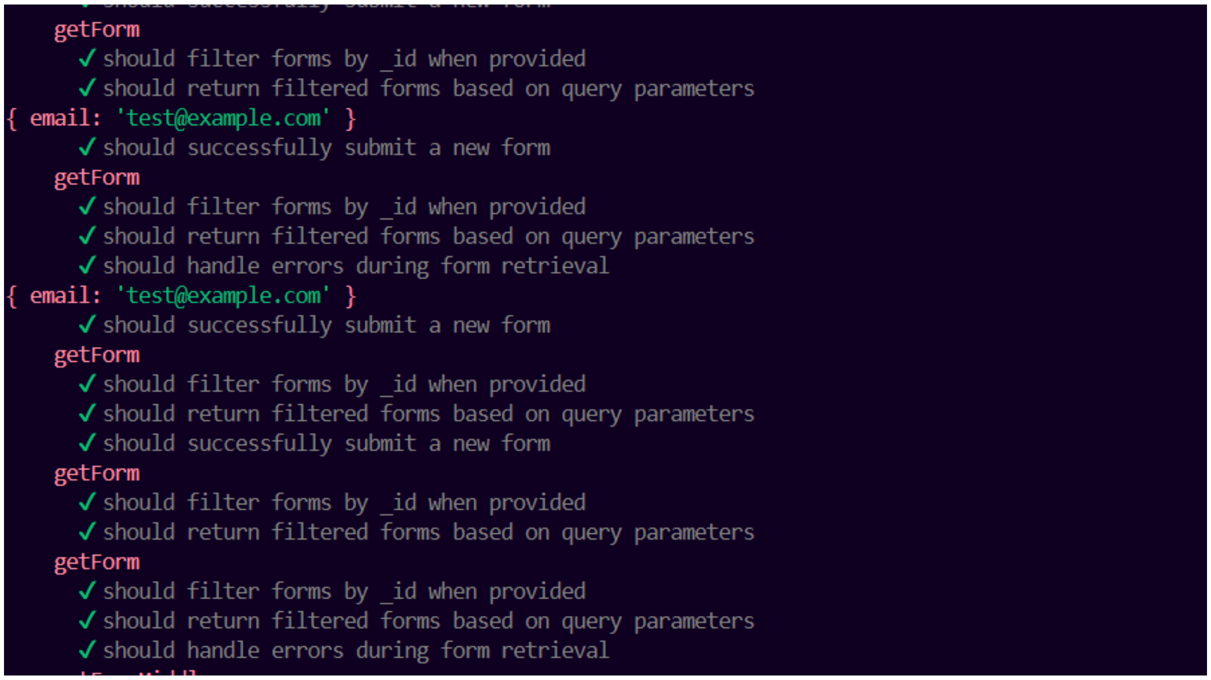
Checks that forms are filtered based on other query parameters like petid, status, and email



```
    it('should handle errors during form retrieval', async () => {
        sinon.stub(formschema, 'find').rejects(new Error('Database error'));

        await getForm(req, res);

        expect(res.status.calledWith(400)).to.be.true;
        expect(res.status().json.calledOnce).to.be.true;
        expect(res.status().json.firstCall.args[0]).to.deep.equal({
            success: false,
            message: 'Database error'
        });
    });
});
```

Verifies that the function gracefully handles errors (e.g., database connectivity issues) and returns an appropriate error message with status 400.

Output:

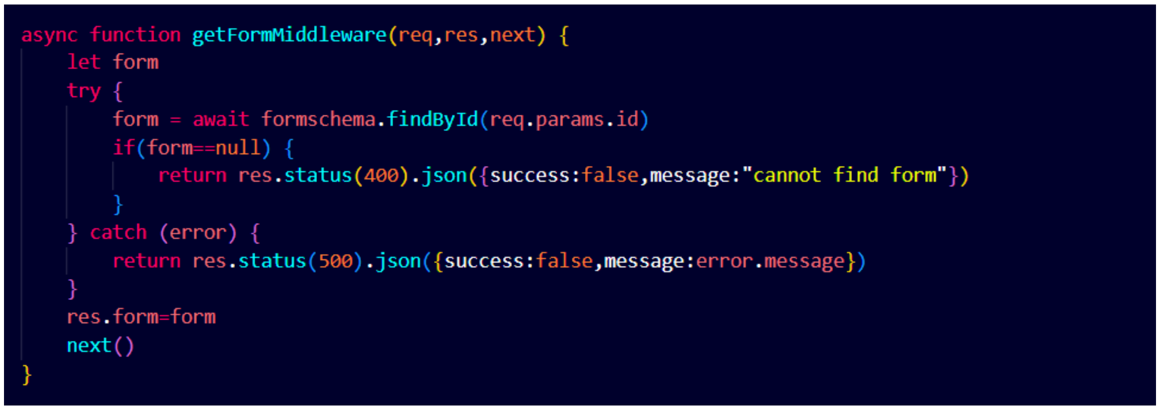


```
    getForm
      ✓ should filter forms by _id when provided
      ✓ should return filtered forms based on query parameters
{ email: 'test@example.com' }
      ✓ should successfully submit a new form
    getForm
      ✓ should filter forms by _id when provided
      ✓ should return filtered forms based on query parameters
      ✓ should handle errors during form retrieval
{ email: 'test@example.com' }
      ✓ should successfully submit a new form
    getForm
      ✓ should filter forms by _id when provided
      ✓ should return filtered forms based on query parameters
      ✓ should successfully submit a new form
    getForm
      ✓ should filter forms by _id when provided
      ✓ should return filtered forms based on query parameters
    getForm
      ✓ should filter forms by _id when provided
      ✓ should return filtered forms based on query parameters
      ✓ should handle errors during form retrieval
```

All these functions work together to ensure correctness, and display following messages.

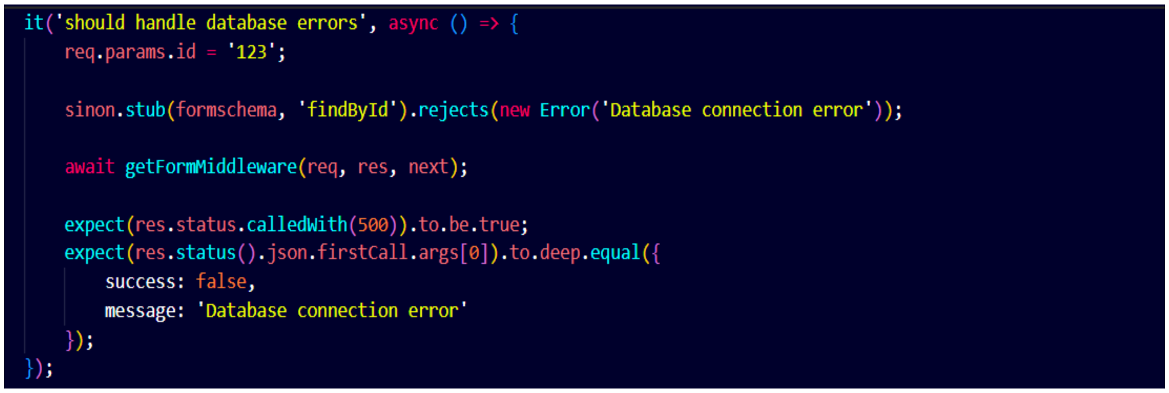
c. getFormMiddleware

This middleware function retrieves a form by its ID.



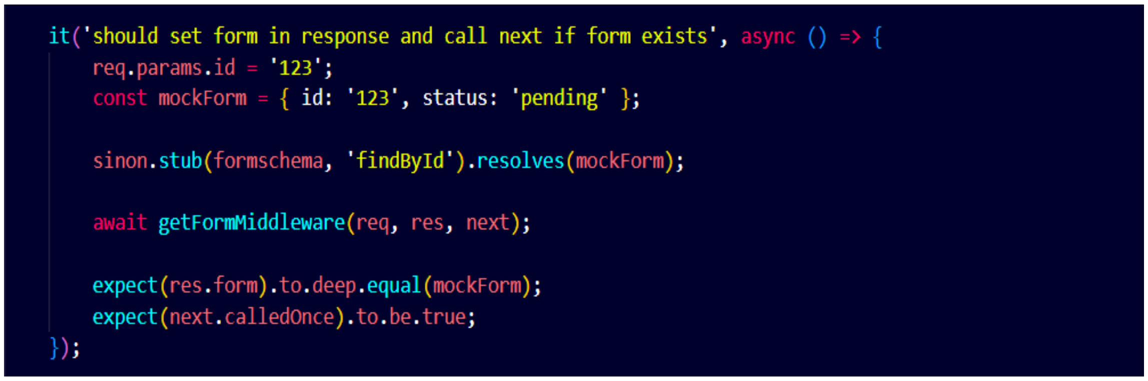
```
async function getFormMiddleware(req,res,next) {
    let form
    try {
        form = await formschema.findById(req.params.id)
        if(form==null) {
            return res.status(400).json({success:false,message:"cannot find form"})
        }
    } catch (error) {
        return res.status(500).json({success:false,message:error.message})
    }
    res.form=form
    next()
}
```

Test cases:



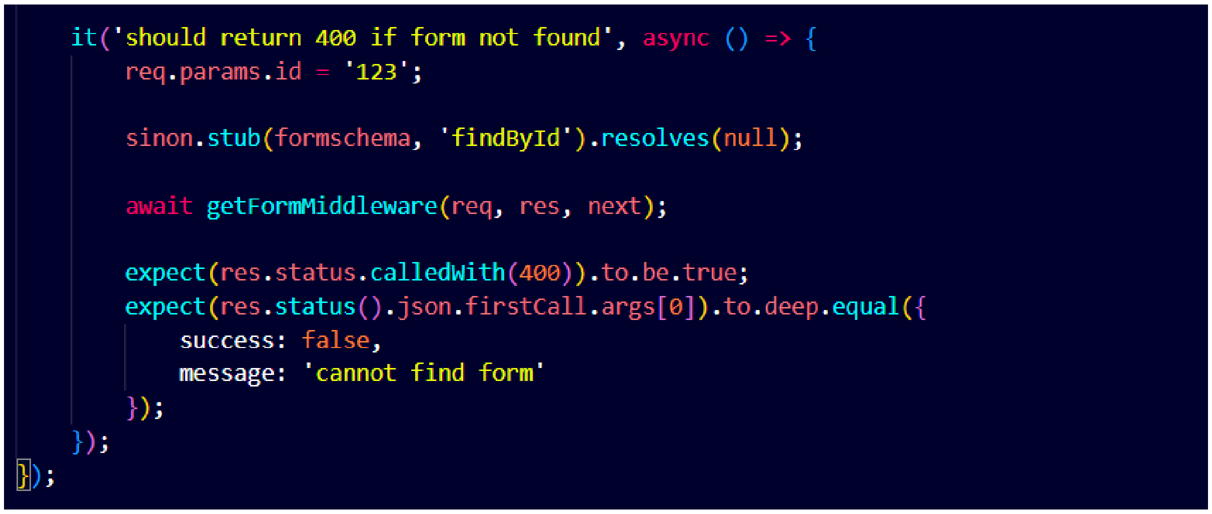
```
it('should handle database errors', async () => {
    req.params.id = '123';

    sinon.stub(formschema, 'findById').rejects(new Error('Database connection error'));

    await getFormMiddleware(req, res, next);

    expect(res.status.calledWith(500)).to.be.true;
    expect(res.status().json.firstCall.args[0]).to.deep.equal({
        success: false,
        message: 'Database connection error'
    });
});
```

Ensures that if there's a database error while retrieving a form by its id, an error response with a 500-status code is returned.



```
it('should set form in response and call next if form exists', async () => {
    req.params.id = '123';
    const mockForm = { id: '123', status: 'pending' };

    sinon.stub(formschema, 'findById').resolves(mockForm);

    await getFormMiddleware(req, res, next);

    expect(res.form).to.deep.equal(mockForm);
    expect(next.calledOnce).to.be.true;
});
```

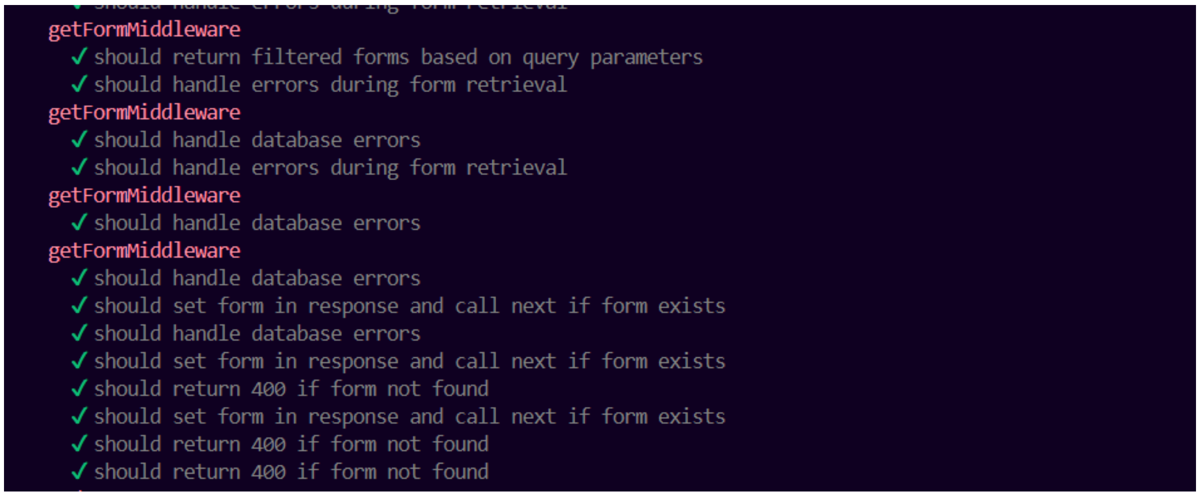
Tests that if a form is found, it is added to res.form, and the next middleware function is called.



```
    it('should return 400 if form not found', async () => {
        req.params.id = '123';

        sinon.stub(formschema, 'findById').resolves(null);

        await getFormMiddleware(req, res, next);

        expect(res.status.calledWith(400)).to.be.true;
        expect(res.status().json.firstCall.args[0]).to.deep.equal({
            success: false,
            message: 'cannot find form'
        });
    });
});
```

Checks that if no form is found for the given id, an error message saying "cannot find form" is returned with a 400 status.

Output:

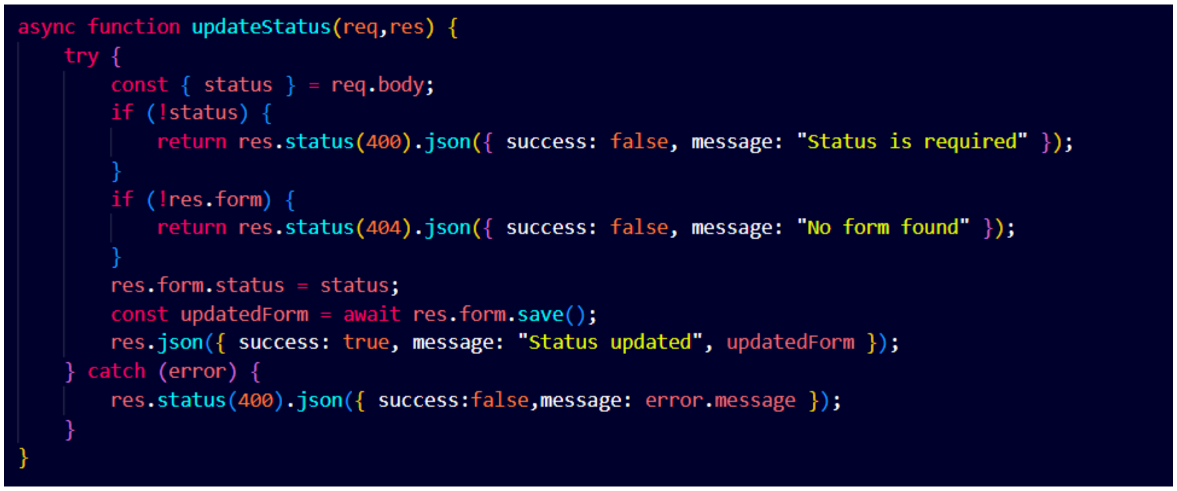


```
  getFormMiddleware
    ✓ should return filtered forms based on query parameters
    ✓ should handle errors during form retrieval
  getFormMiddleware
    ✓ should handle database errors
    ✓ should handle errors during form retrieval
  getFormMiddleware
    ✓ should handle database errors
  getFormMiddleware
    ✓ should handle database errors
    ✓ should set form in response and call next if form exists
    ✓ should handle database errors
    ✓ should set form in response and call next if form exists
    ✓ should return 400 if form not found
    ✓ should set form in response and call next if form exists
    ✓ should return 400 if form not found
    ✓ should return 400 if form not found
```

These functions thus, work together to ensure the code runs and display the following messages on success.

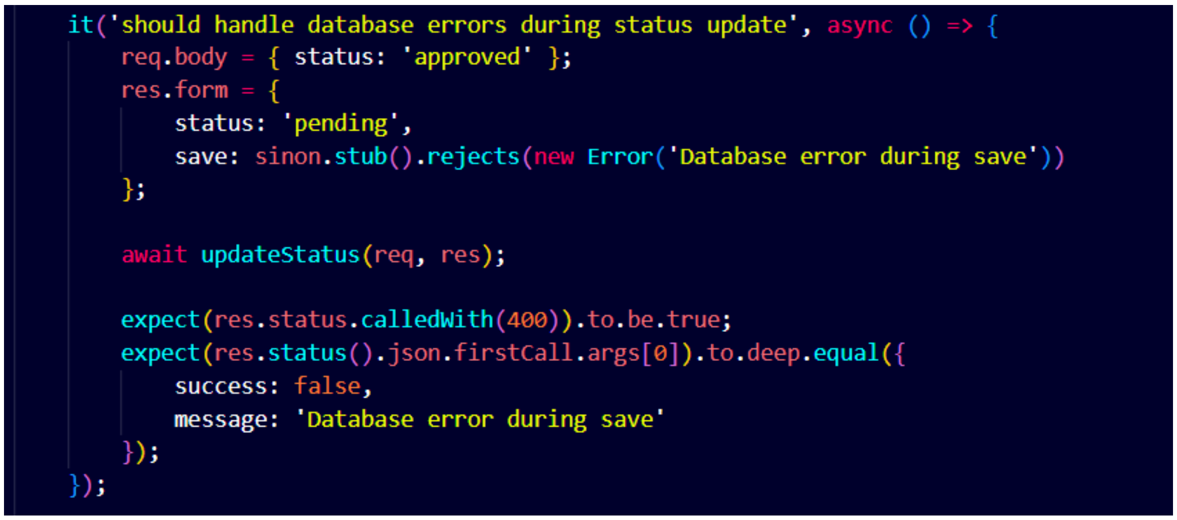
d. updateStatus

This function updates the status of a form.

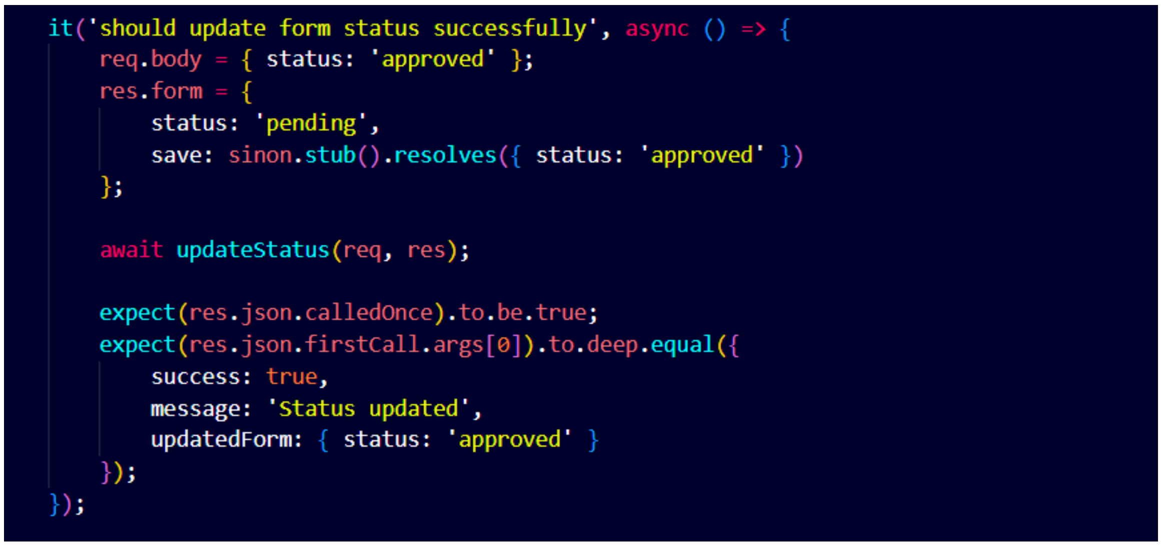


```
async function updateStatus(req,res) {
    try {
        const { status } = req.body;
        if (!status) {
            return res.status(400).json({ success: false, message: "Status is required" });
        }
        if (!res.form) {
            return res.status(404).json({ success: false, message: "No form found" });
        }
        res.form.status = status;
        const updatedForm = await res.form.save();
        res.json({ success: true, message: "Status updated", updatedForm });
    } catch (error) {
        res.status(400).json({ success:false,message: error.message });
    }
}
```

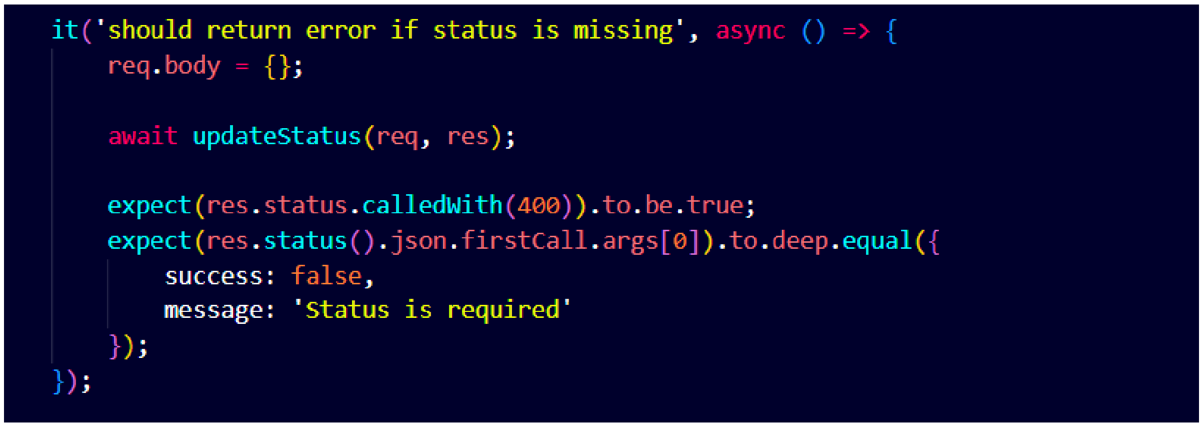
Test cases:


```
it('should handle database errors during status update', async () => {
    req.body = { status: 'approved' };
    res.form = {
        status: 'pending',
        save: sinon.stub().rejects(new Error('Database error during save'))
    };

    await updateStatus(req, res);

    expect(res.status.calledWith(400)).to.be.true;
    expect(res.status().json.firstCall.args[0]).to.deep.equal({
        success: false,
        message: 'Database error during save'
    });
});
```

Verifies that if a database error occurs while saving the updated form, an appropriate error response is returned with status 400.

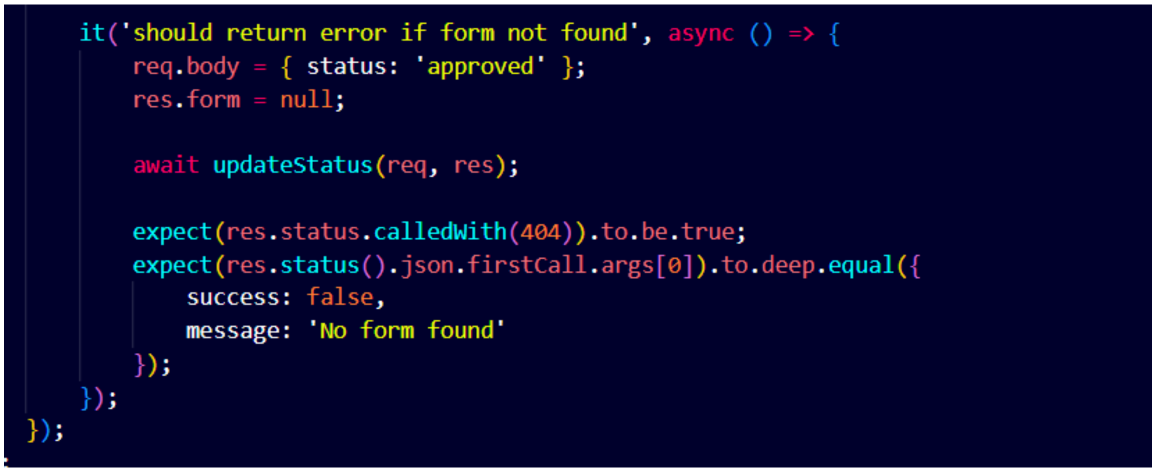

```
it('should update form status successfully', async () => {
    req.body = { status: 'approved' };
    res.form = {
        status: 'pending',
        save: sinon.stub().resolves({ status: 'approved' })
    };

    await updateStatus(req, res);

    expect(res.json.calledOnce).to.be.true;
    expect(res.json.firstCall.args[0]).to.deep.equal({
        success: true,
        message: 'Status updated',
        updatedForm: { status: 'approved' }
    });
});
```

Tests that when a valid status is provided, the form's status is updated successfully in the database. Ensures the response includes the updated form and a success message.



```
it('should return error if status is missing', async () => {
    req.body = {};

    await updateStatus(req, res);

    expect(res.status.calledWith(400)).to.be.true;
    expect(res.status().json.firstCall.args[0]).to.deep.equal({
        success: false,
        message: 'Status is required'
    });
});
```

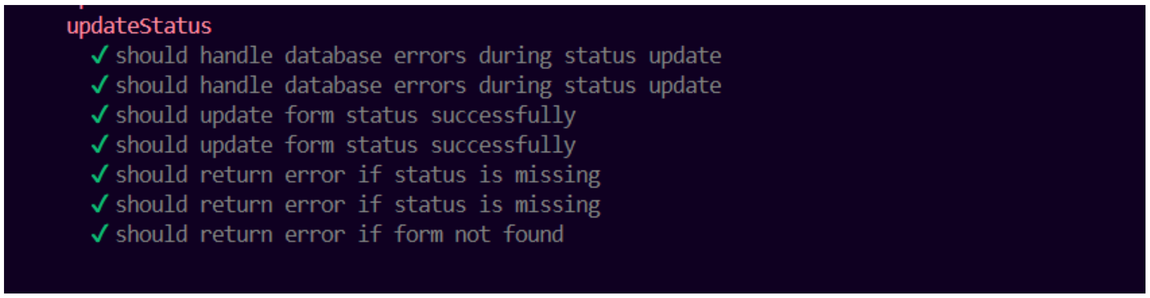
Checks that an error is returned if the status field is missing in the request body. The error message should state "Status is required."



```
    it('should return error if form not found', async () => {
        req.body = { status: 'approved' };
        res.form = null;

        await updateStatus(req, res);

        expect(res.status.calledWith(404)).to.be.true;
        expect(res.status().json.firstCall.args[0]).to.deep.equal({
            success: false,
            message: 'No form found'
        });
    });
});
```

Ensures that if no form exists in res.form (set by the middleware), an error message is returned with status 404.

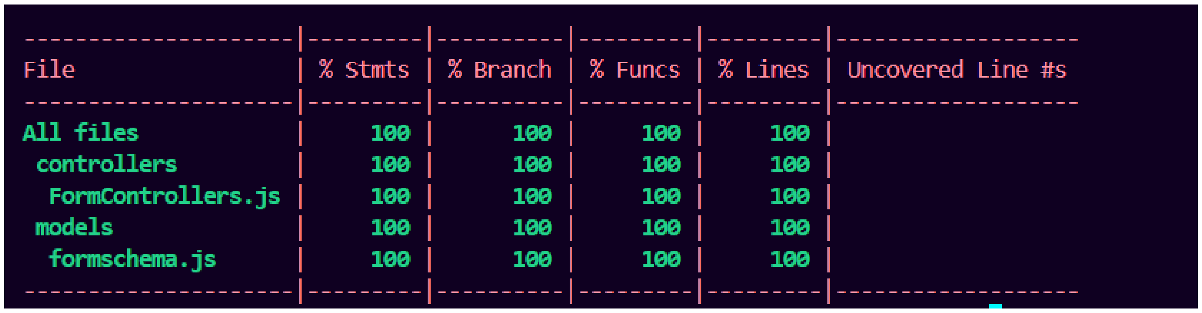
Output:



```
    updateStatus
      ✓ should handle database errors during status update
      ✓ should handle database errors during status update
      ✓ should update form status successfully
      ✓ should update form status successfully
      ✓ should return error if status is missing
      ✓ should return error if status is missing
      ✓ should return error if form not found
```

These statements display no error and proper functioning of the code.

Overall coverage of code:



```
---------------------|---------|----------|---------|---------|-------------------
File                 | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------------|---------|----------|---------|---------|-------------------
All files            |     100 |      100 |     100 |     100 |
 controllers         |     100 |      100 |     100 |     100 |
  FormControllers.js |     100 |      100 |     100 |     100 |
 models              |     100 |      100 |     100 |     100 |
  formschema.js      |     100 |      100 |     100 |     100 |
---------------------|---------|----------|---------|---------|-------------------
```

100% coverage was achieved with test cases written down for every line, and every line being tested.