# Unit testing G1 Pet Adoption System – HappyTails

**Testing Framework:** Mocha@10.8.2

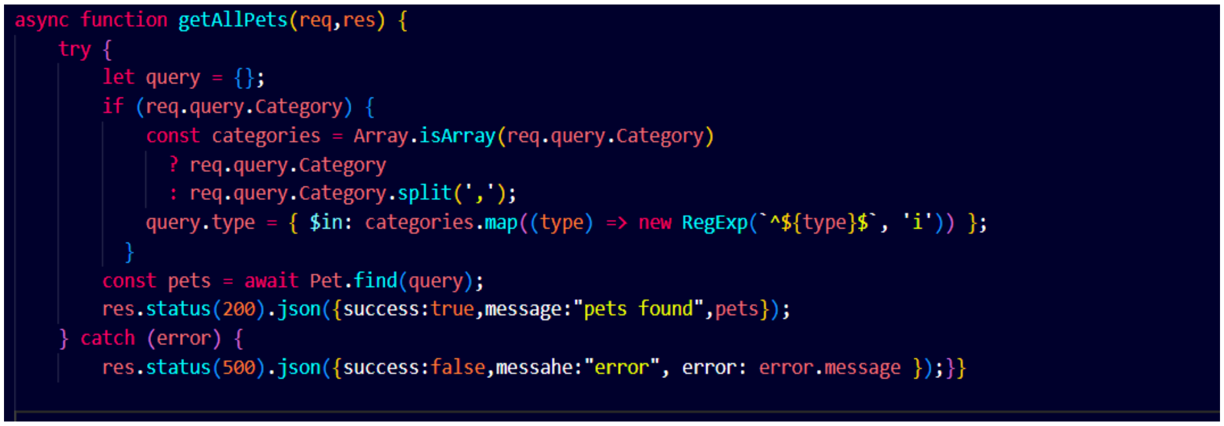
**Assertion library used:** Chai@4.3.4

**Other:** Sinon@19.0.2

## APIControllers

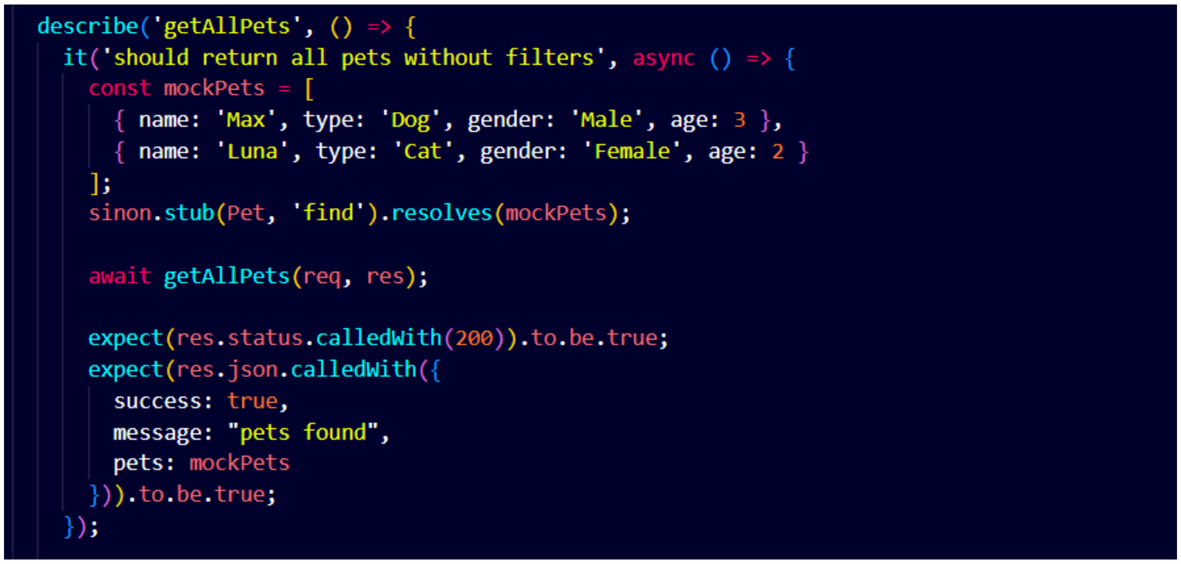
### a. getAllPets

This function is intended to fetch detailed information about a specific pet.



```
async function getAllPets(req,res) {
    try {
        let query = {};
        if (req.query.Category) {
            const categories = Array.isArray(req.query.Category)
              ? req.query.Category
              : req.query.Category.split(',');
            query.type = { $in: categories.map((type) => new RegExp(`^${type}$`, 'i')) };
        }
        const pets = await Pet.find(query);
        res.status(200).json({success:true,message:"pets found",pets});
    } catch (error) {
        res.status(500).json({success:false,messahe:"error", error: error.message });}}
```

**Test Cases:**

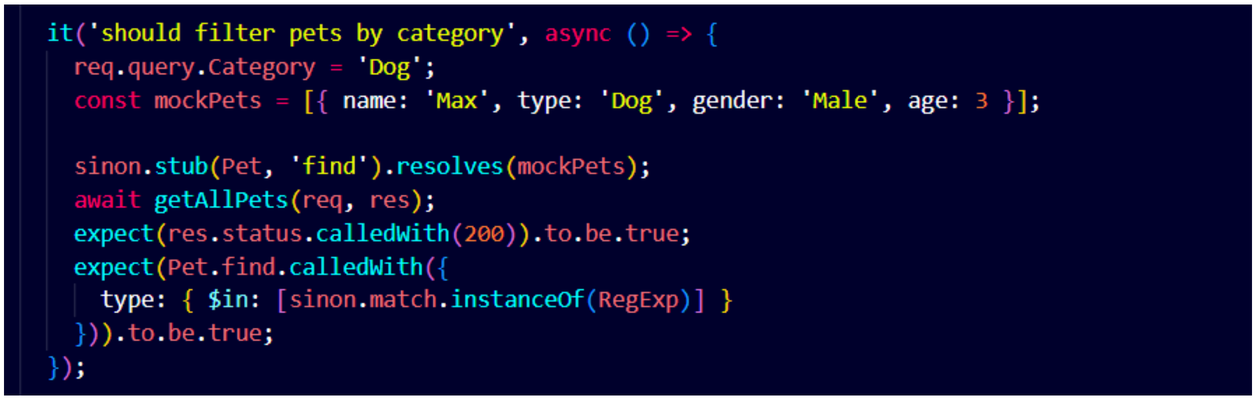


```
describe('getAllPets', () => {
  it('should return all pets without filters', async () => {
    const mockPets = [
      { name: 'Max', type: 'Dog', gender: 'Male', age: 3 },
      { name: 'Luna', type: 'Cat', gender: 'Female', age: 2 }
    ];
    sinon.stub(Pet, 'find').resolves(mockPets);

    await getAllPets(req, res);

    expect(res.status.calledWith(200)).to.be.true;
    expect(res.json.calledWith({
      success: true,
      message: "pets found",
      pets: mockPets
    })).to.be.true;
  });
```

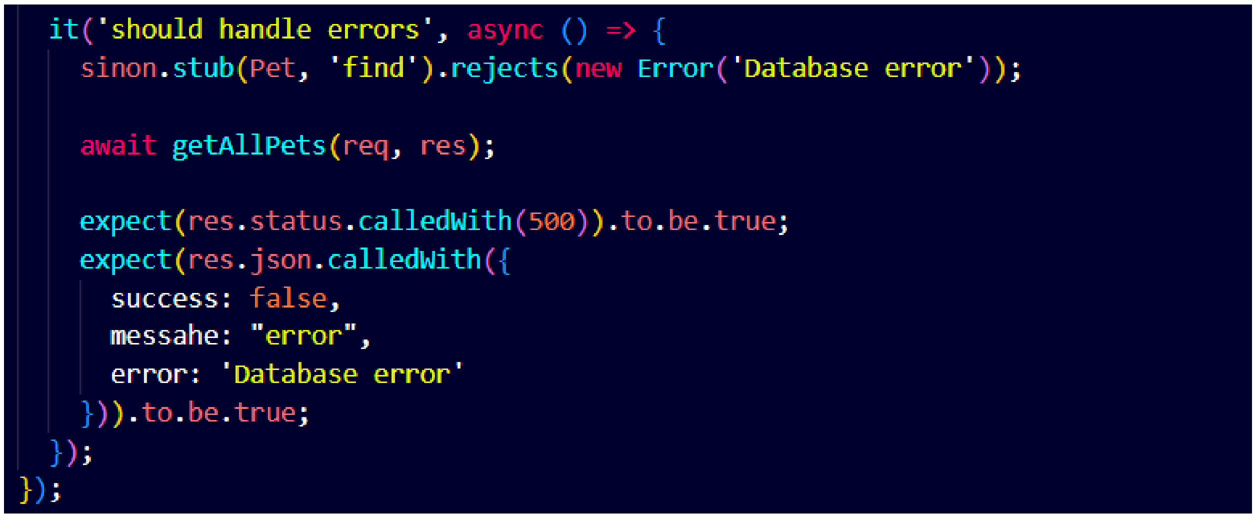
Checks if the controller fetches all pets from the database, without using filters.



```
it('should filter pets by category', async () => {
  req.query.Category = 'Dog';
  const mockPets = [{ name: 'Max', type: 'Dog', gender: 'Male', age: 3 }];

  sinon.stub(Pet, 'find').resolves(mockPets);
  await getAllPets(req, res);
  expect(res.status.calledWith(200)).to.be.true;
  expect(Pet.find.calledWith({
    type: { $in: [sinon.match.instanceOf(RegExp)] }
  })).to.be.true;
});
```

Checks if the controller correctly filters pets by the category provided in the query parameters.



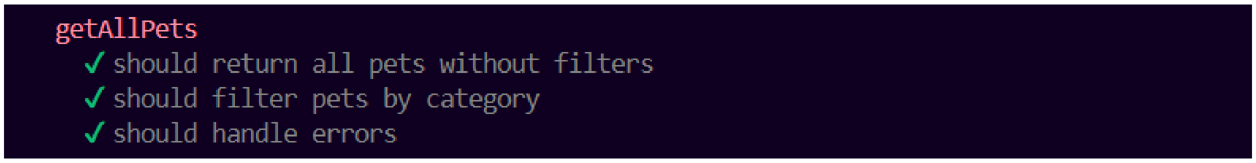
```
  it('should handle errors', async () => {
    sinon.stub(Pet, 'find').rejects(new Error('Database error'));

    await getAllPets(req, res);

    expect(res.status.calledWith(500)).to.be.true;
    expect(res.json.calledWith({
      success: false,
      messahe: "error",
      error: 'Database error'
    })).to.be.true;
  });
});
```

Checks if the controller correctly handles database errors when fetching pets. It ensures the correct status code and error message in the response.

**Output:**

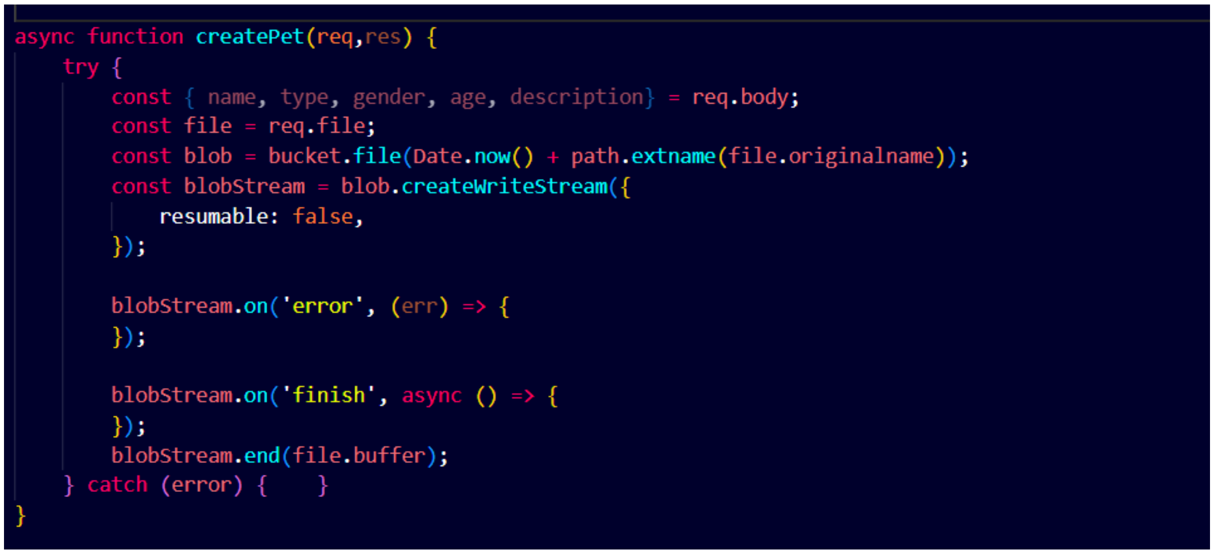


```
  getAllPets
    ✓ should return all pets without filters
    ✓ should filter pets by category
    ✓ should handle errors
```

Image shows all test cases running and functioning.

**b. createpet**

Creates a new pet entry in the database with details and an image uploaded to a cloud storage bucket.

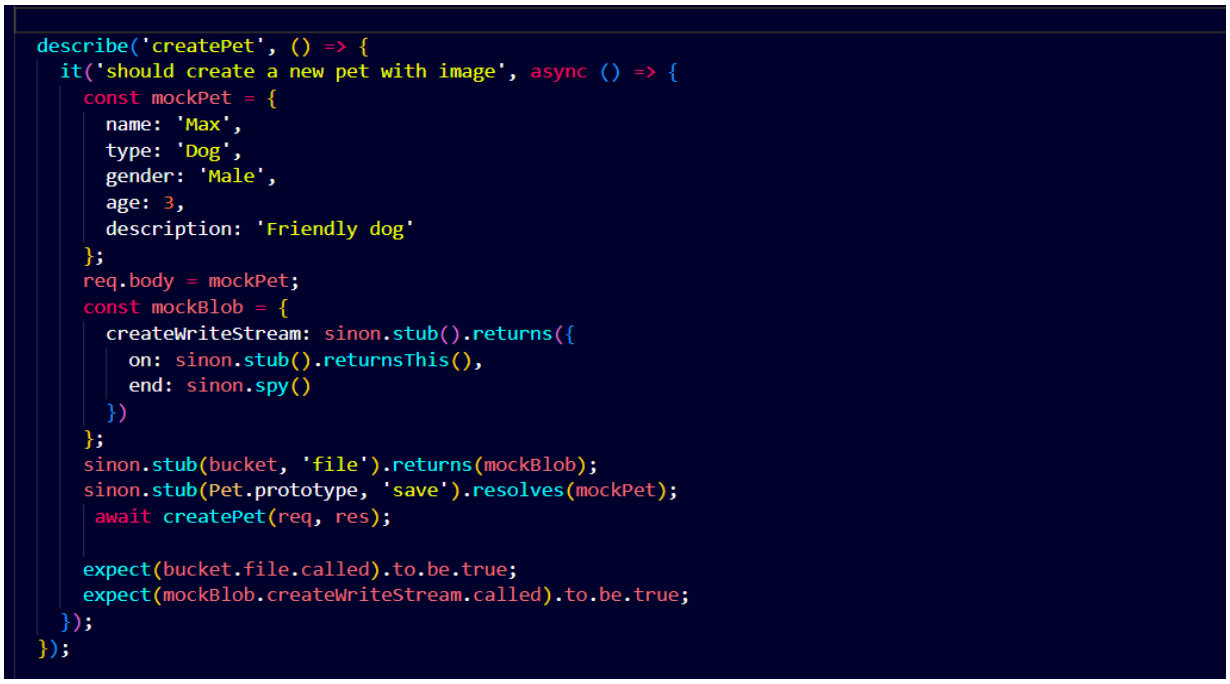


```
async function createPet(req,res) {
    try {
        const { name, type, gender, age, description} = req.body;
        const file = req.file;
        const blob = bucket.file(Date.now() + path.extname(file.originalname));
        const blobStream = blob.createWriteStream({
            resumable: false,
        });

        blobStream.on('error', (err) => {
        });

        blobStream.on('finish', async () => {
        });
        blobStream.end(file.buffer);
    } catch (error) {    }
}
```

**Test Cases:**

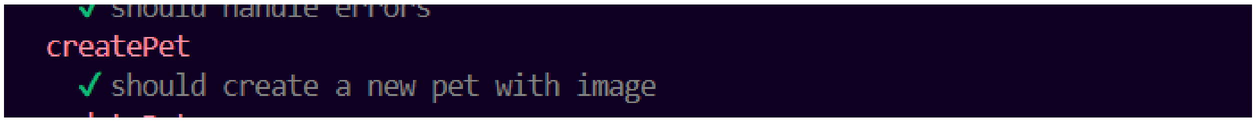


```
describe('createPet', () => {
  it('should create a new pet with image', async () => {
    const mockPet = {
      name: 'Max',
      type: 'Dog',
      gender: 'Male',
      age: 3,
      description: 'Friendly dog'
    };
    req.body = mockPet;
    const mockBlob = {
      createWriteStream: sinon.stub().returns({
        on: sinon.stub().returnsThis(),
        end: sinon.spy()
      })
    };
    sinon.stub(bucket, 'file').returns(mockBlob);
    sinon.stub(Pet.prototype, 'save').resolves(mockPet);
     await createPet(req, res);

    expect(bucket.file.called).to.be.true;
    expect(mockBlob.createWriteStream.called).to.be.true;
  });
});
```

Should create a new pet with image, i.e checks if the controller creates a new pet entry with an uploaded image.

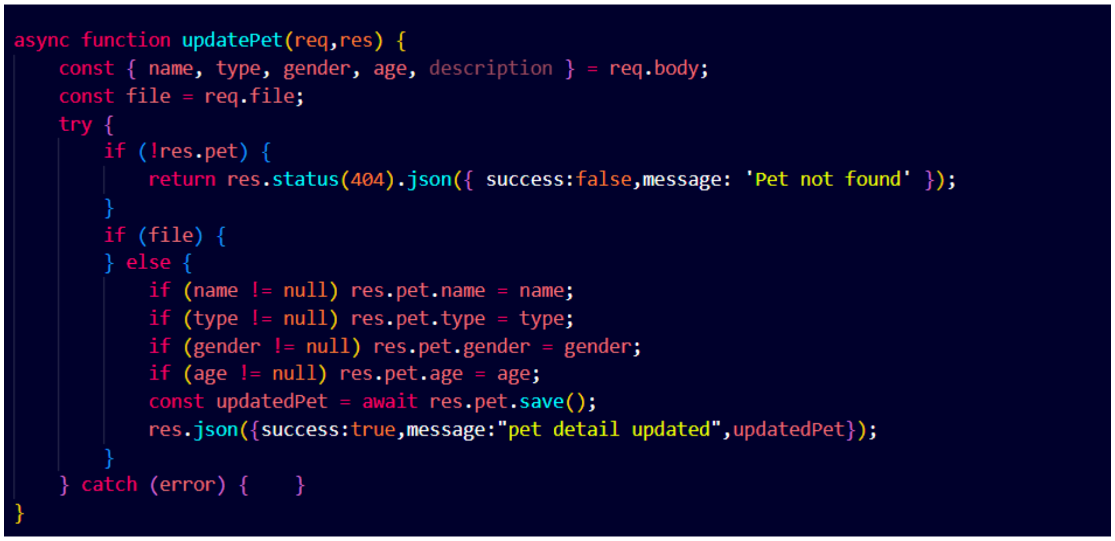
Output:



```
  ✓ should handle errors
createPet
  ✓ should create a new pet with image
```

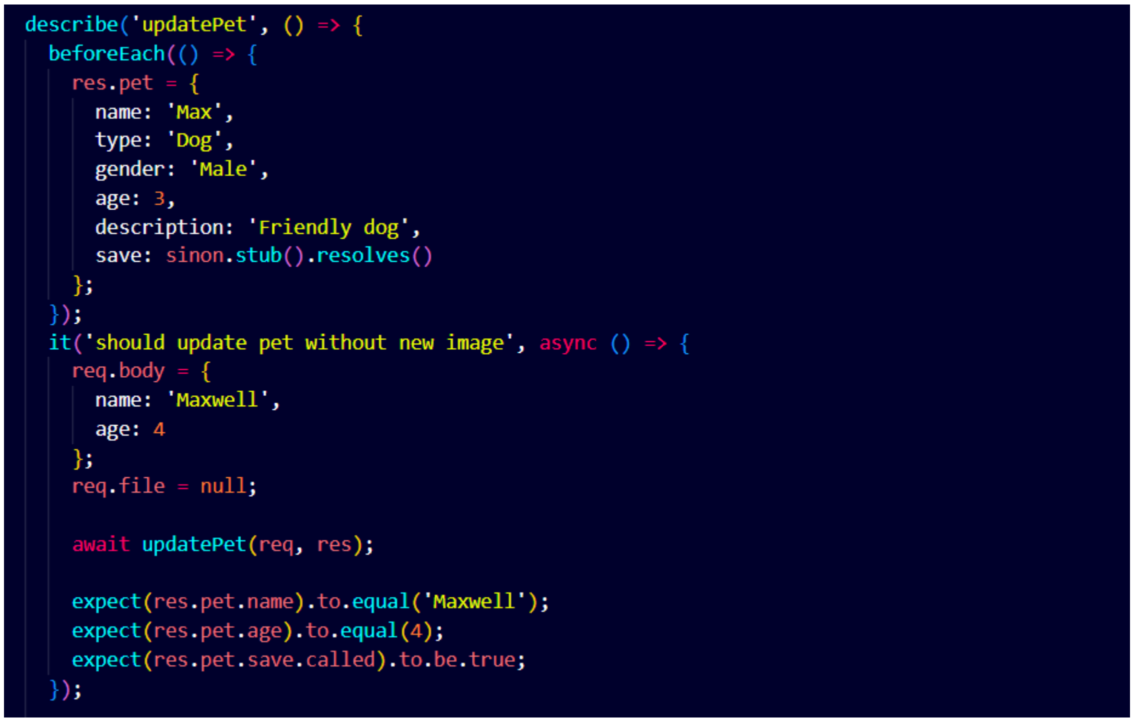
c. **updatePet**

Updates the details of an existing pet, optionally including a new image.



```
async function updatePet(req,res) {
    const { name, type, gender, age, description } = req.body;
    const file = req.file;
    try {
        if (!res.pet) {
            return res.status(404).json({ success:false,message: 'Pet not found' });
        }
        if (file) {
        } else {
            if (name != null) res.pet.name = name;
            if (type != null) res.pet.type = type;
            if (gender != null) res.pet.gender = gender;
            if (age != null) res.pet.age = age;
            const updatedPet = await res.pet.save();
            res.json({success:true,message:"pet detail updated",updatedPet});
        }
    } catch (error) {    }
}
```

## Test Cases:

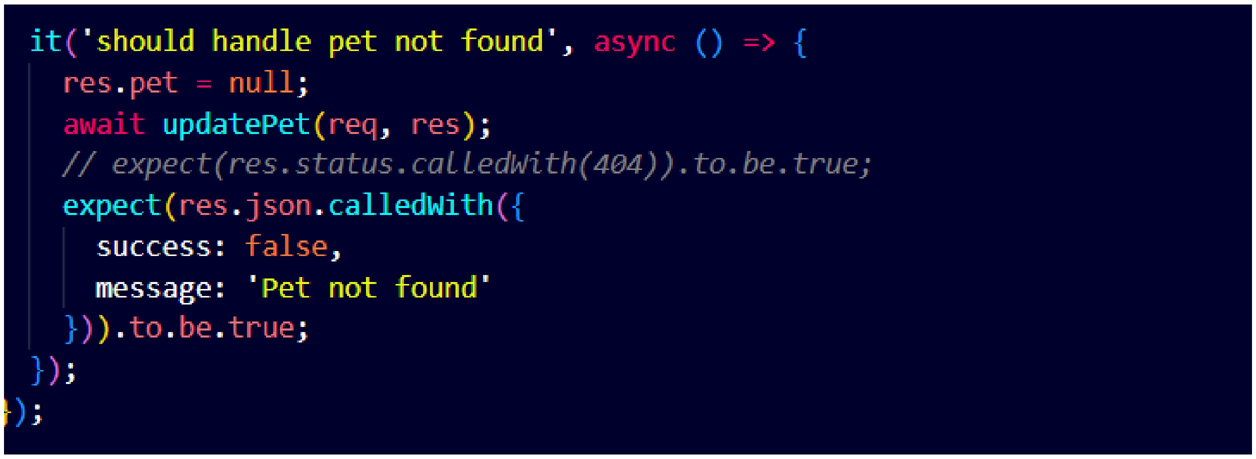


```
describe('updatePet', () => {
  beforeEach(() => {
    res.pet = {
      name: 'Max',
      type: 'Dog',
      gender: 'Male',
      age: 3,
      description: 'Friendly dog',
      save: sinon.stub().resolves()
    };
  });
  it('should update pet without new image', async () => {
    req.body = {
      name: 'Maxwell',
      age: 4
    };
    req.file = null;

    await updatePet(req, res);

    expect(res.pet.name).to.equal('Maxwell');
    expect(res.pet.age).to.equal(4);
    expect(res.pet.save.called).to.be.true;
  });
```

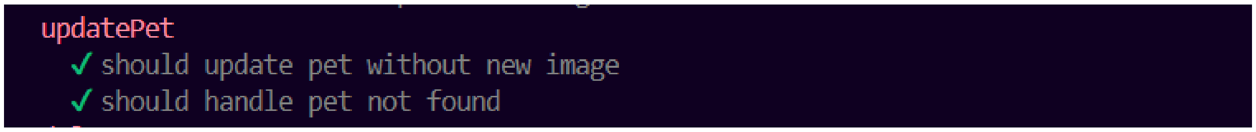
Checks If the controller updates the pet details (e.g., name and age) without uploading a new image.



```
it('should handle pet not found', async () => {
  res.pet = null;
  await updatePet(req, res);
  // expect(res.status.calledWith(404)).to.be.true;
  expect(res.json.calledWith({
    success: false,
    message: 'Pet not found'
  })).to.be.true;
});
});
```

Checks If the controller handles the case where the pet to update does not exist.

Output:

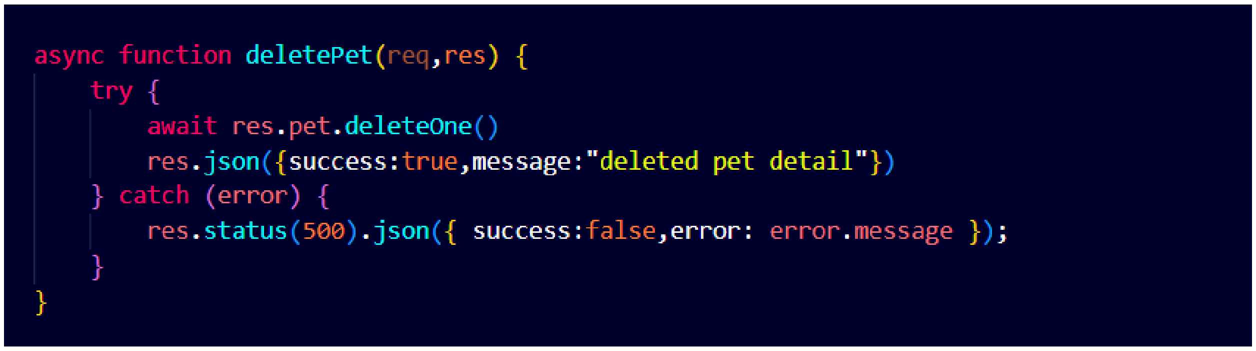


```
updatePet
  ✓ should update pet without new image
  ✓ should handle pet not found
```

Shows proper functioning and working of “updatePet” function.

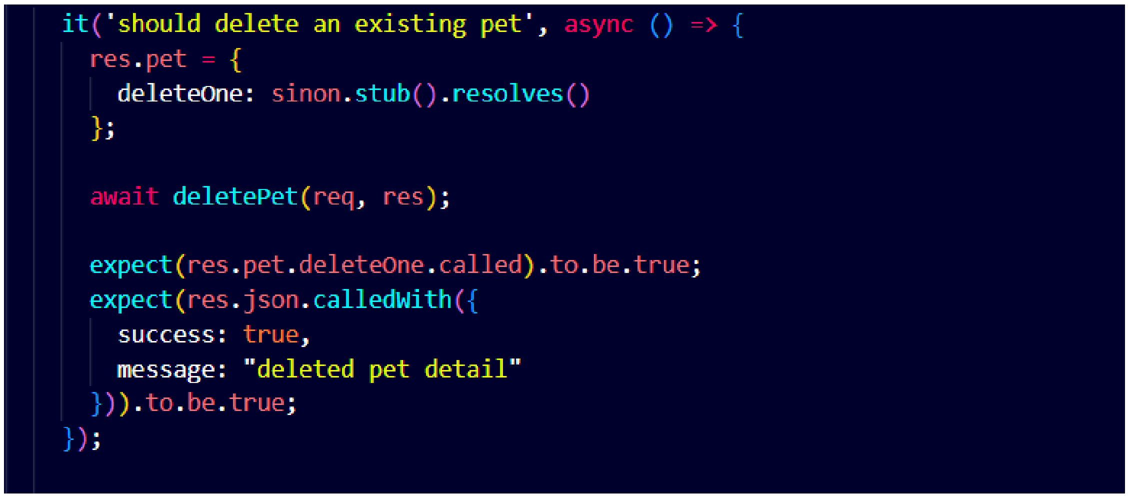
**d. deletePet**

Deletes an existing pet from the database.



```
async function deletePet(req,res) {
    try {
        await res.pet.deleteOne()
        res.json({success:true,message:"deleted pet detail"})
    } catch (error) {
        res.status(500).json({ success:false,error: error.message });
    }
}
```

**Test Cases:**

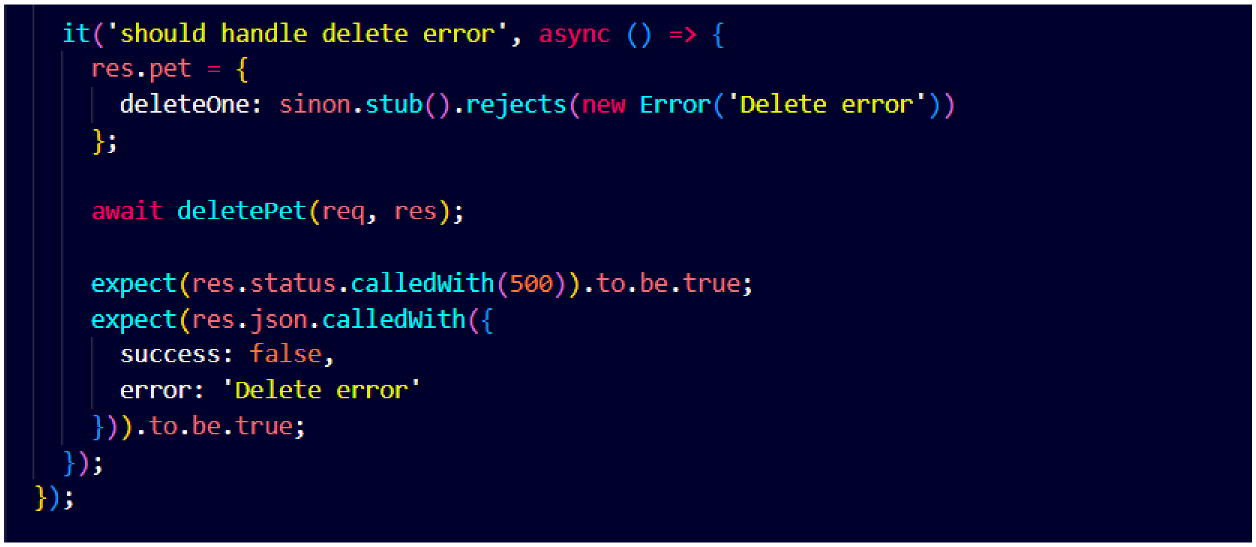


```
  it('should delete an existing pet', async () => {
    res.pet = {
      deleteOne: sinon.stub().resolves()
    };

    await deletePet(req, res);

    expect(res.pet.deleteOne.called).to.be.true;
    expect(res.json.calledWith({
      success: true,
      message: "deleted pet detail"
    })).to.be.true;
  });
```

Checks if the controller deletes a pet successively. Verifies that the “deleteOne” function is called and the response indicates success.



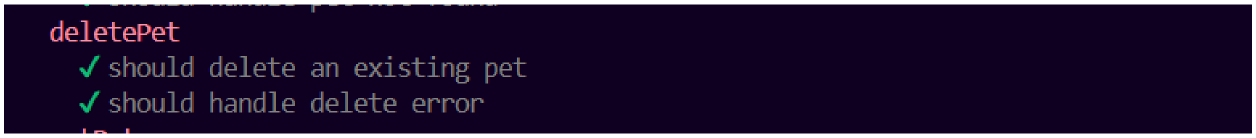
```
  it('should handle delete error', async () => {
    res.pet = {
      deleteOne: sinon.stub().rejects(new Error('Delete error'))
    };

    await deletePet(req, res);

    expect(res.status.calledWith(500)).to.be.true;
    expect(res.json.calledWith({
      success: false,
      error: 'Delete error'
    })).to.be.true;
  });
});
```

Checks if the controller handles errors during deletion, ensures the correct status code and error message are returned.

**Output:**

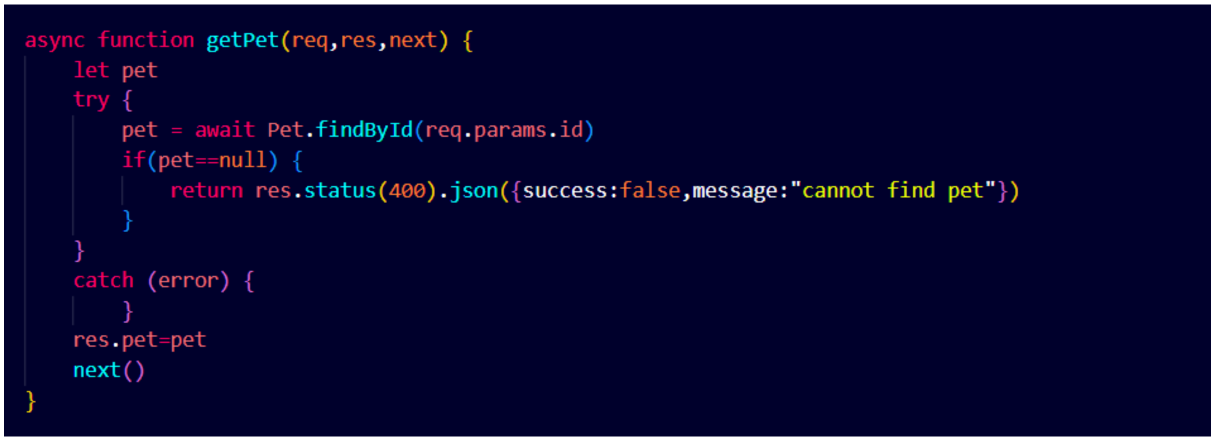


```
deletePet
  ✓ should delete an existing pet
  ✓ should handle delete error
```

Ensures proper functioning of the delete function and testcases.

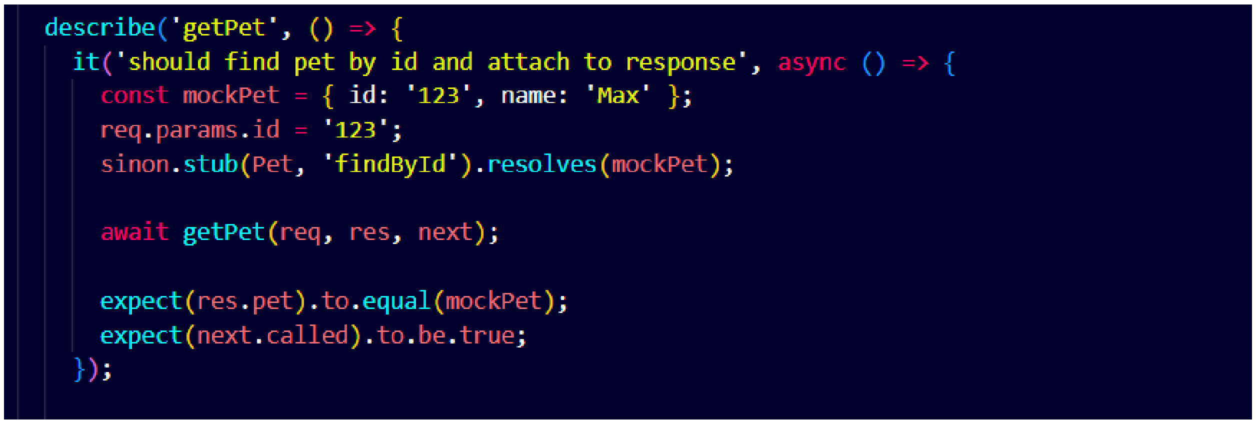
**e. getPet**

Retrieves a specific pet by its ID and attaches it to the res object for subsequent handlers.



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

**Test Cases:**

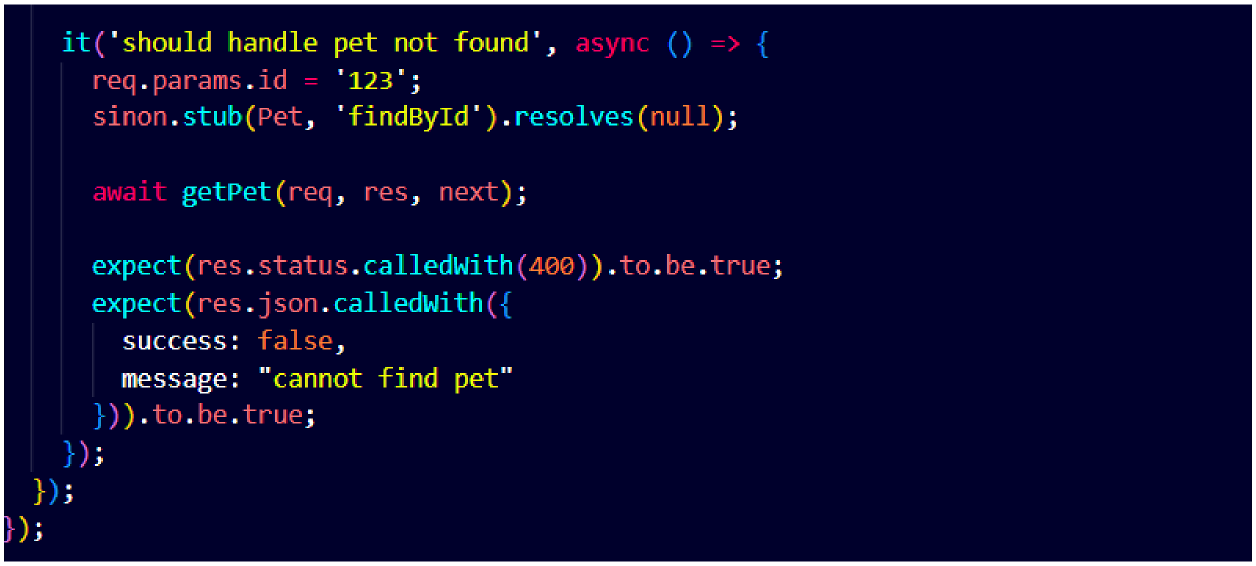


```
describe('getPet', () => {
  it('should find pet by id and attach to response', async () => {
    const mockPet = { id: '123', name: 'Max' };
    req.params.id = '123';
    sinon.stub(Pet, 'findById').resolves(mockPet);

    await getPet(req, res, next);

    expect(res.pet).to.equal(mockPet);
    expect(next.called).to.be.true;
  });
```

Checks if the controller retrieves a pet by ID and attaches it to the res object.



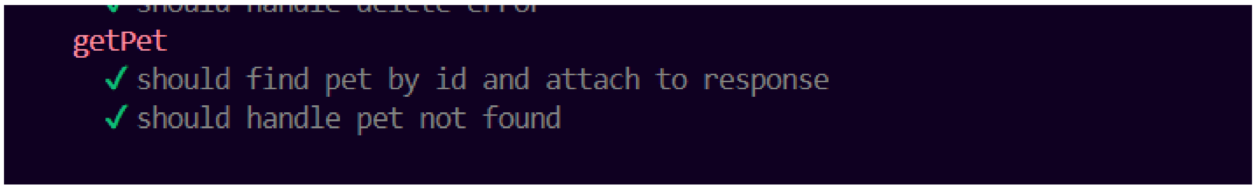
```
    it('should handle pet not found', async () => {
      req.params.id = '123';
      sinon.stub(Pet, 'findById').resolves(null);

      await getPet(req, res, next);

      expect(res.status.calledWith(400)).to.be.true;
      expect(res.json.calledWith({
        success: false,
        message: "cannot find pet"
      })).to.be.true;
    });
  });
});
```

Checks if the controller handles the case where no pet is found for the given ID. Ensures a 400 response with an appropriate error message.

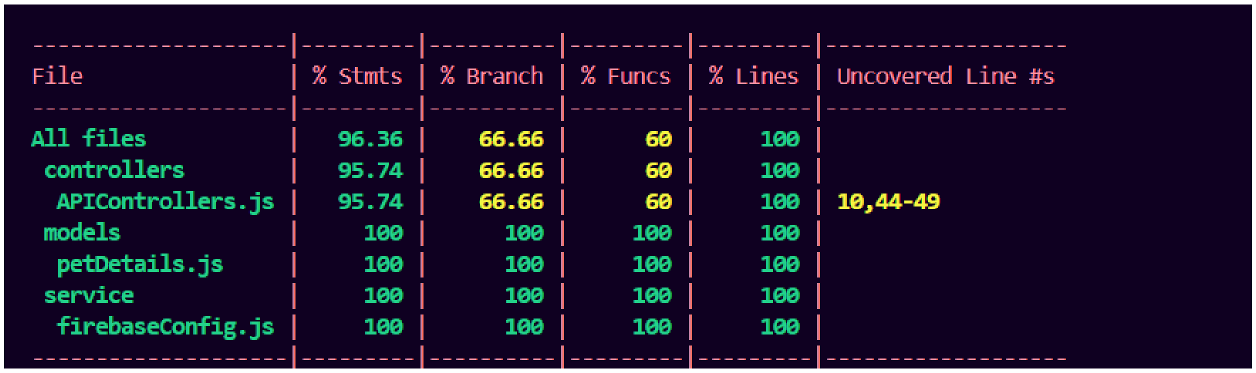
**Output:**



```
  getPet
    ✓ should find pet by id and attach to response
    ✓ should handle pet not found
```

Ensures correct functioning of getPet function and shows proper outputs.

**Overall Coverage of Code:**



```
--------------------|---------|----------|---------|---------|-------------------
File                | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------------|---------|----------|---------|---------|-------------------
All files           |   96.36 |    66.66 |      60 |     100 |
 controllers        |   95.74 |    66.66 |      60 |     100 |
  APIControllers.js |   95.74 |    66.66 |      60 |     100 | 10,44-49
 models             |     100 |      100 |     100 |     100 |
  petDetails.js     |     100 |      100 |     100 |     100 |
 service            |     100 |      100 |     100 |     100 |
  firebaseConfig.js |     100 |      100 |     100 |     100 |
--------------------|---------|----------|---------|---------|-------------------
```

**The coverage of this file, while not a 100% stands out to 95.74%. Along with this the branch coverage appears to be significantly lower, this is because of the uncovered lines both of which are "if-else" statements and hence do not execute all the path conditionals. I tried hard to write testcases for the uncovered lines but after covering these lines some of the other cases were resulting in error.**