Mutation Testing for EktaFund Donation Website

IT-314 Software Engineering
Group 19

Group Mentor: Diti Soni

Group Members:

202201315 Dani Jal Nileshbhai
202201281 Shah Sakshi Dharmeshkumar
202201354 Ram Kulkarni
202201292 Devamm Patel
202201275 Kunj Mahendra Bhuva
202201271 Rishabh Jain
202201362 Tanay Kewalramani
202201303 Pandya Ansh Ashutoshbhai
202201264 Yadav Alpeshkumar Banshbahadur
202201363 Mangukiya Harshkumar Ashwinbhai

Contents

Chapter 1

Introduction

1.1 Purpose of the Document

Mutation testing is a software testing technique that evaluates the effectiveness of unit tests by introducing small, intentional changes (called mutations) into the source code. These mutations are typically simple changes, such as altering operators, changing values, or modifying control flow. The goal is to assess whether the existing unit tests can detect these changes and fail appropriately. If a test case fails when a mutation is introduced, it means that the test case is effective in detecting potential issues in the code.

1.2 Overview of Ekta Group Donation System

The Ekta Group Donation Website aims to connect donors with individuals and organizations in need of donations. The platform allows donors to browse various donation categories, make secure donations, and track their donation history. Beneficiaries can list their needs, and admins can manage requests, ensuring donations are allocated effectively.

Chapter 2

Mutation Testing Requirements

2.1 Unit Test Framework

A robust framework (e.g., Jest, Mocha, JUnit) to run and validate tests on mutated code.

2.2 Mutation Testing Tool

A tool like Stryker, Pitest, or MutPy to automate the mutation and testing process.

2.3 Comprehensive Unit Tests

A well-covered test suite to detect code changes (mutations).

2.4 Logging and Error Handling

Proper logging to track results and handle errors during mutation testing.

2.5 Mutant Kill Strategy

Clear criteria to determine if a mutant is "killed" (test catches the mutation).

Chapter 3

Mutation Testing using Stryker Mutator

3.0.1 Overview of StrykerMutator

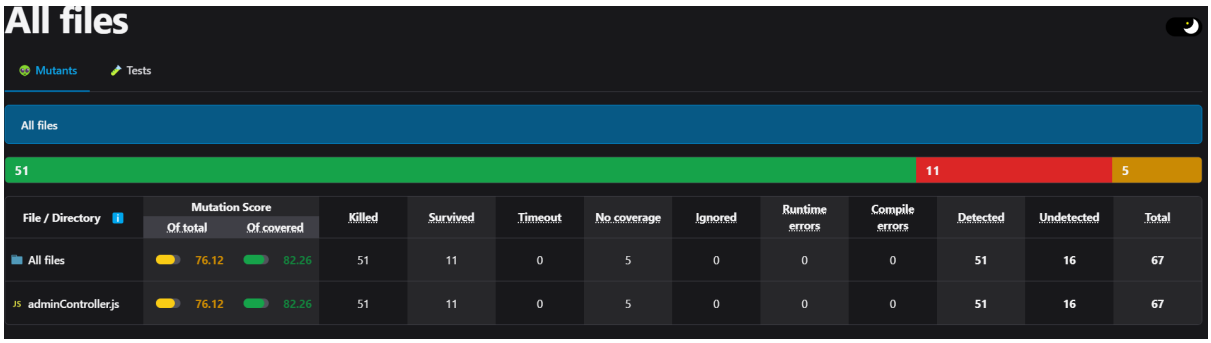
StrykerMutator is an open-source mutation testing framework for JavaScript, TypeScript, and other programming languages. It works by introducing small, controlled changes (mutations) to the codebase and then running the test suite to check whether the tests can detect these mutations. The goal of mutation testing is to assess the effectiveness and coverage of the test suite. StrykerMutator provides detailed reports on the mutation score, helping developers improve their test suite and, ultimately, the reliability and quality of their code.

Key Features of JMeter:

- StrykerMutator helps in identifying weak points in a test suite by introducing small changes (mutations) to the code and checking if the test suite detects these changes.
- Stryker provides various mutation operators, such as changing arithmetic operators or modifying method names, allowing users to simulate different types of faults in the code.
- It provides real-time feedback on how effective a test suite is at detecting introduced mutations, giving an overall mutation score.
- It generates detailed reports on the mutation score, providing insights into how well the tests are performing and which parts of the code require more test coverage.
- The tool is highly configurable, allowing users to set their mutation testing preferences, such as mutation operators, thresholds for test failure, and more.

3.1 Admin Controller

The following report shows the results of mutation testing conducted on the `adminController.js` file.



File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
All files	76.12	82.26	51	11	0	5	0	0	0	51	16	67
adminController.js	76.12	82.26	51	11	0	5	0	0	0	51	16	67

Figure 3.1: Mutation Testing Results for Admin Controller

3.1.1 Mutation Score

In the following test the mutation score is 76.12%

3.1.2 Mutation Breakdown

```
const NGO = require("../models/NGO");
const { sendNotification } = require("../utils/notifications");
const jwt = require("jsonwebtoken");

// Admin login function
exports.adminLogin = async (req, res) => {
  const { email, password } = req.body;

  // Check if email and password are provided
  if (!email || !password) {
    return res.status(400).json({
      message: `${!email ? "Email" : "Password"} is required`,
    });
  }

  // Check if credentials are correct
  if (email === process.env.ADMIN_EMAIL && password === process.env.ADMIN_PASSWORD) {
    try {
      const token = jwt.sign({ role: "admin" }, process.env.JWT_SECRET, {
        expiresIn: "5h",
      });
      res.status(200).json({ message: "Admin login successful", token });
    } catch (err) {
      res.status(500).json({ message: "Error during admin login", error: err });
    }
  } else {
    res.status(401).json({ message: "Invalid admin credentials" });
  }
};
```

Figure 3.2: **Killed Mutants**

```
// Check if credentials are correct
if (email === process.env.ADMIN_EMAIL && password === process.env.ADMIN_PASSWORD) {
  try {
    const token = jwt.sign({ role: "admin" }, process.env.JWT_SECRET, {
      expiresIn: "5h",
    });
    res.status(200).json({ message: "Admin login successful", token });
  } catch (err) {
    res.status(500).json({ message: "Error during admin login", error: err });
  }
} else {
  res.status(401).json({ message: "Invalid admin credentials" });
}
};
```

Figure 3.3: **Survived Mutants**

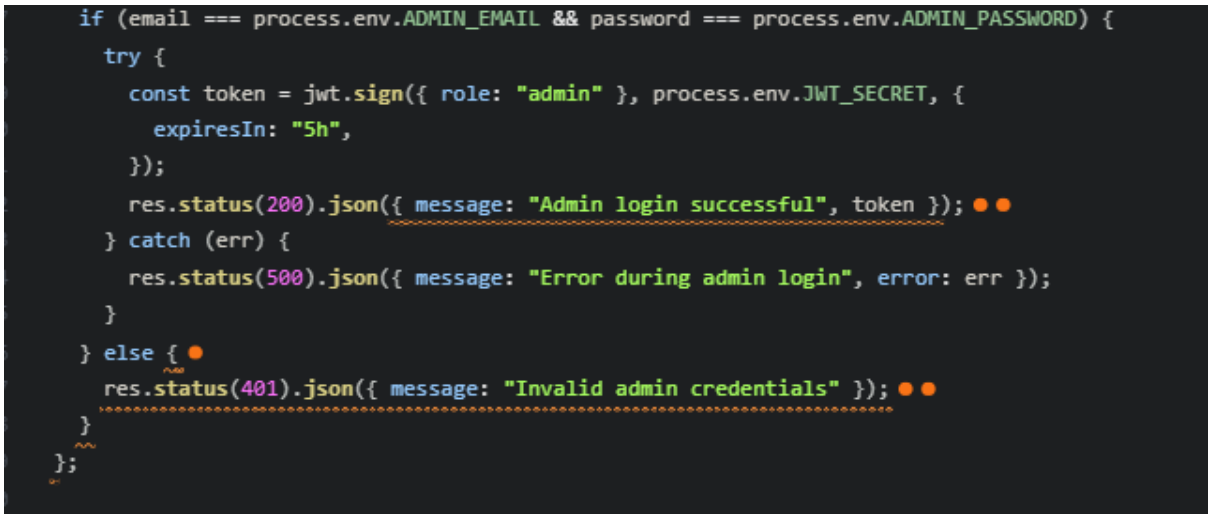


Figure 3.4: **No Coverage**

3.2 Billing Controller

The following report shows the results of mutation testing conducted on the `billingController.js` file.

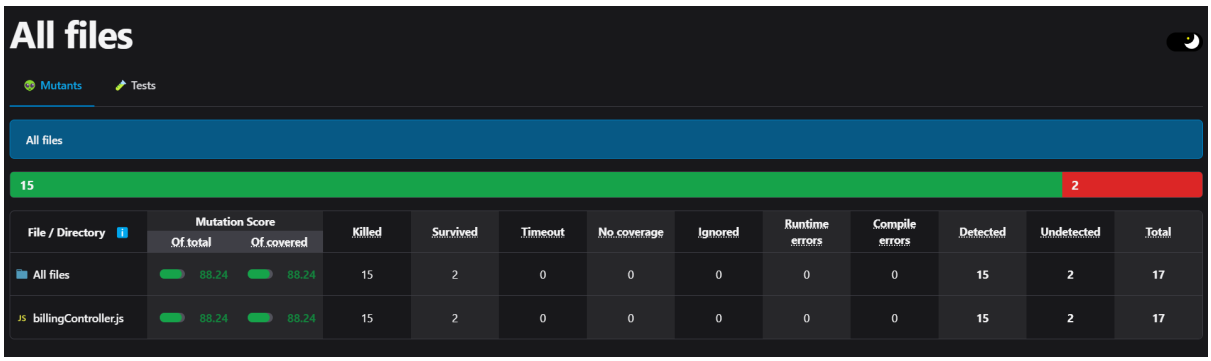


Figure 3.5: Mutation Testing Results for Billing Controller

3.2.1 Mutation Score

In the following test the mutation score is 88.24%

3.2.2 Mutation Breakdown

```
const billingService = require("../services/billing");
const reportGenerator = require("../services/reportGenerator");
const Transaction = require("../models/Transaction");

exports.generateBill = async (req, res) => {
  try {
    const { amount, ngoId, ngoName } = req.body;

    // Create and store a new transaction record with ngoName
    const transaction = await Transaction.create({
      amount,
      ngoId,
      ngoName,
      status: "Completed",
      transactionDate: new Date(),
    });

    res.status(201).json({ message: "Transaction recorded successfully", transaction });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Failed to generate bill" });
  }
};

exports.downloadDonationReport = async (req, res) => {
  try {
    const { ngoId } = req.params;

    // Generate report
    const reportBuffer = await reportGenerator.generateDonationReport(ngoId);

    // Send downloadable PDF file
    res.setHeader("Content-Disposition", "attachment; filename=donation_report.pdf");
    res.contentType("application/pdf");
    res.send(reportBuffer);
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Could not generate donation report" });
  }
```

Figure 3.6: **Killed Mutants**

```
// controllers/billingController.js
const billingService = require("../services/billing");
const reportGenerator = require("../services/reportGenerator");
const Transaction = require("../models/Transaction");

exports.generateBill = async (req, res) => {
  try {
    const { amount, ngoId, ngoName } = req.body;

    // Create and store a new transaction record with ngoName
    const transaction = await Transaction.create({
      amount,
      ngoId,
      ngoName,
      status: "Completed",
      transactionDate: new Date(),
    });

    res.status(201).json({ message: "Transaction recorded successfully", transaction });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Failed to generate bill" });
  }
};
```

Figure 3.7: **Survived Mutants**

3.3 Donation Controller

The following report shows the results of mutation testing conducted on the `donationController.js` file.

Figure 3.8: Mutation Testing Results for Donation Controller

3.3.1 Mutation Score

In the following test the mutation score is 69.7%

3.3.2 Mutation Breakdown

```
const paymentService = require("../services/payment"); // Use the mock payment service
const receiptGenerator = require("../services/receiptGenerator");
const Donation = require("../models/Donation");

exports.processDonation = async (req, res) => {
  try {
    // Destructure required fields from request body
    const { amount, donorId, name, mobileNumber, ngoName } = req.body;

    // Ensure all required fields are present
    if (
      !amount ||
      !donorId ||
      !name ||
      !mobileNumber ||
      !ngoName
    ) {
      return res.status(400).json({ error: "All fields are required" });
    }

    // Generate a mock transaction token based on user-provided details
    const paymentResult = await paymentService.createMockTransaction({
      name,
      mobileNumber,
      ngoName,
      amount,
    });

    if (!paymentResult.transactionToken) {
      return res.status(400).json({ error: "Payment processing failed." });
    }

    // Save donation details, including transaction token, to the database
    const donation = await Donation.create({
      amount,
      donorId,
      transactionId: paymentResult.transactionToken,
      status: "Completed",
    });

    // Generate a receipt for the donation
    const receipt = await receiptGenerator.generateReceipt(donation);

    res
      .status(201)
      .json({ message: "Donation processed successfully", receipt });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Internal Server Error" });
  }
};
```

Figure 3.9: **Killed Mutants**

```
const paymentService = require("../services/payment"); // Use the mock payment servi
const receiptGenerator = require("../services/receiptGenerator");
const Donation = require("../models/Donation");

exports.processDonation = async (req, res) => {
  try {
    // Destructure required fields from request body
    const { amount, donorId, name, mobileNumber, ngoName } = req.body;

    // Ensure all required fields are present
    if (
      !amount || ●●●●●●●
      !donorId ||
      !name ||
      !mobileNumber ||
      !ngoName
    ) {
      return res.status(400).json({ error: "All fields are required" });
    }

    // Generate a mock transaction token based on user-provided details
    const paymentResult = await paymentService.createMockTransaction({ ●
      name,
      mobileNumber,
      ngoName,
      amount,
    });

    if (!paymentResult.transactionToken) {
      return res.status(400).json({ error: "Payment processing failed." });
    }

    // Save donation details, including transaction token, to the database
    const donation = await Donation.create({ ●
      amount,
      donorId,
      transactionId: paymentResult.transactionToken,
      status: "Completed", ●
    });
```

Figure 3.10: **Survived Mutants**

3.4 Billing Controller

The following report shows the results of mutation testing conducted on the `billingController.js` file.

Figure 3.11: Mutation Testing Results for Billing Controller

3.4.1 Mutation Score

In the following test the mutation score is 88.24%

3.4.2 Mutation Breakdown

```
const billingService = require("../services/billing");
const reportGenerator = require("../services/reportGenerator");
const Transaction = require("../models/Transaction");

exports.generateBill = async (req, res) => {
  try {
    const { amount, ngoId, ngoName } = req.body;

    // Create and store a new transaction record with ngoName
    const transaction = await Transaction.create({
      amount,
      ngoId,
      ngoName,
      status: "Completed",
      transactionDate: new Date(),
    });

    res.status(201).json({ message: "Transaction recorded successfully", transaction });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Failed to generate bill" });
  }
};

exports.downloadDonationReport = async (req, res) => {
  try {
    const { ngoId } = req.params;

    // Generate report
    const reportBuffer = await reportGenerator.generateDonationReport(ngoId);

    // Send downloadable PDF file
    res.setHeader("Content-Disposition", "attachment; filename=donation_report.pdf");
    res.contentType("application/pdf");
    res.send(reportBuffer);
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Could not generate donation report" });
  }
```

Figure 3.12: **Killed Mutants**

```
// controllers/billingController.js
const billingService = require("../services/billing");
const reportGenerator = require("../services/reportGenerator");
const Transaction = require("../models/Transaction");

exports.generateBill = async (req, res) => {
  try {
    const { amount, ngoId, ngoName } = req.body;

    // Create and store a new transaction record with ngoName
    const transaction = await Transaction.create({
      amount,
      ngoId,
      ngoName,
      status: "Completed",
      transactionDate: new Date(),
    });

    res.status(201).json({ message: "Transaction recorded successfully", transaction });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Failed to generate bill" });
  }
};
```

Figure 3.13: **Survived Mutants**

3.5 Donor Controller

The following report shows the results of mutation testing conducted on the `donorController.js` file.

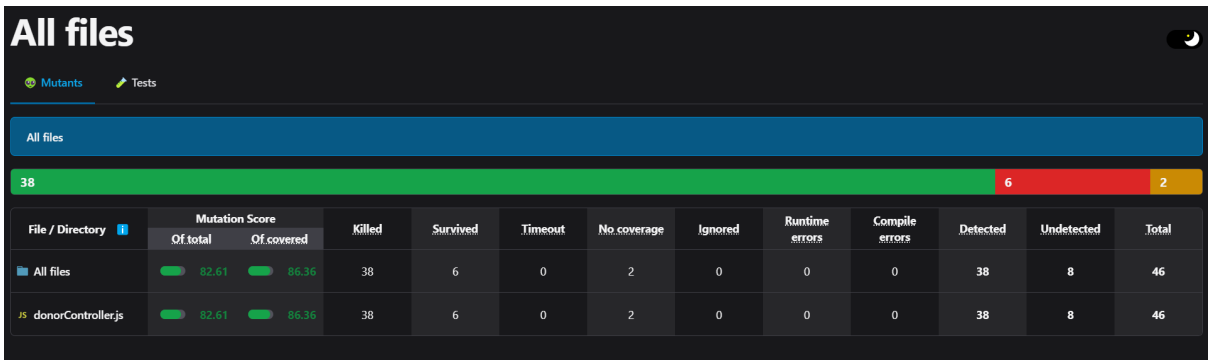


Figure 3.14: Mutation Testing Results for Donor Controller

3.5.1 Mutation Score

In the following test the mutation score is 82.61%

3.5.2 Mutation Breakdown

```
const jwt = require('jsonwebtoken');
const NGO = require('../models/NGO');

// Register a new donor
exports.registerDonor = async (req, res) => {
    try {
        const { name, email, password, contactNumber } = req.body;

        // Check if donor already exists
        let donor = await Donor.findOne({ email });
        if (donor) return res.status(400).json({ message: 'Donor already registered' });

        // Create and save new donor
        donor = new Donor({ name, email, password, contactNumber });
        await donor.save();

        res.status(201).json({ message: 'Donor registered successfully' });
    } catch (error) {
        res.status(500).json({ error: 'Server error' });
    }
};

// Authenticate donor and generate token
exports.loginDonor = async (req, res) => {
    try {
        const { email, password } = req.body;
        const donor = await Donor.findOne({ email });
        if (!donor) return res.status(400).json({ message: 'Invalid email or password' });

        // Check password
        const isMatch = await bcrypt.compare(password, donor.password);
        if (!isMatch) return res.status(400).json({ message: 'Invalid email or password' });

        // Generate JWT
        const token = jwt.sign({ id: donor._id, role: donor.role }, process.env.JWT_SECRET, { expiresIn: '1h' });
        res.json({ token });
    } catch (error) {
        res.status(500).json({ error: 'Server error' });
    }
};

// Function to retrieve and filter NGOs
exports.getFilteredNGOs = async (req, res) => {
    try {
      // Extract query parameters for filtering
      const { location } = req.body ;

      // Build filter criteria based on query params
      let filterCriteria = {};
      if (location) filterCriteria.location = location;
```

Figure 3.15: **Killed Mutants**

```
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const NGO = require('../models/NGO');

// Register a new donor
exports.registerDonor = async (req, res) => {
    try {
        const { name, email, password, contactNumber } = req.body;

        // Check if donor already exists
        let donor = await Donor.findOne({ email });
        if (donor) return res.status(400).json({ message: 'Donor already registered' });

        // Create and save new donor
        donor = new Donor({ name, email, password, contactNumber });
        await donor.save();

        res.status(201).json({ message: 'Donor registered successfully' });
    } catch (error) {
        res.status(500).json({ error: 'Server error' });
    }
};

// Authenticate donor and generate token
exports.loginDonor = async (req, res) => {
    try {
        const { email, password } = req.body;
        const donor = await Donor.findOne({ email });
        if (!donor) return res.status(400).json({ message: 'Invalid email or password' });

        // Check password
        const isMatch = await bcrypt.compare(password, donor.password);
        if (!isMatch) return res.status(400).json({ message: 'Invalid email or password' });

        // Generate JWT
        const token = jwt.sign({ id: donor._id, role: donor.role }, process.env.JWT_SECRET, { expiresIn: '1h' });
        res.json({ token });
    } catch (error) {
        res.status(500).json({ error: 'Server error' });
    }
};
```

Figure 3.16: **Survived Mutants**

```
// Authenticate donor and generate token
exports.loginDonor = async (req, res) => {
    try {
        const { email, password } = req.body;
        const donor = await Donor.findOne({ email });
        if (!donor) return res.status(400).json({ message: 'Invalid email or password' });

        // Check password
        const isMatch = await bcrypt.compare(password, donor.password);
        if (!isMatch) return res.status(400).json({ message: 'Invalid email or password' });

        // Generate JWT
        const token = jwt.sign({ id: donor._id, role: donor.role }, process.env.JWT_SECRET, { expiresIn: '1h' });
        res.json({ token });
    } catch (error) {
        res.status(500).json({ error: 'Server error' });
    }
};
```

Figure 3.17: **No Coverage**

3.6 NGO Controller

The following report shows the results of mutation testing conducted on the `ngoController.js` file.

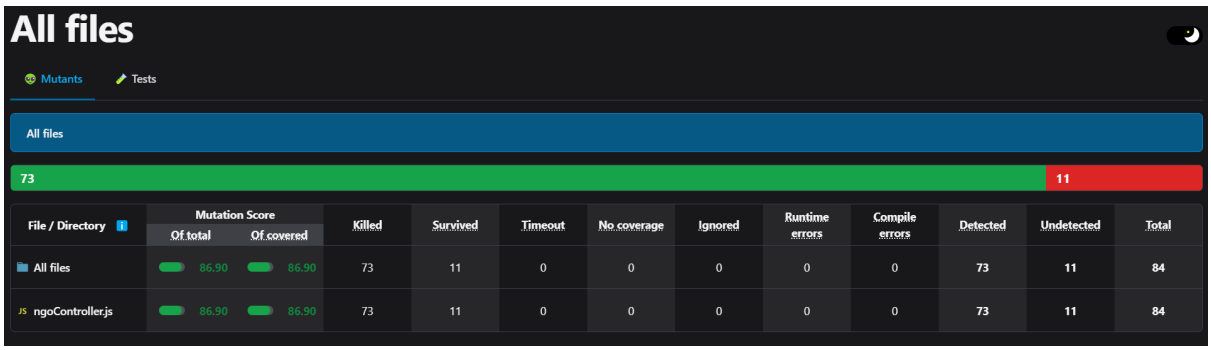


Figure 3.18: Mutation Testing Results for NGO Controller

3.6.1 Mutation Score

In the following test the mutation score is 86.9%

3.6.2 Mutation Breakdown

```
  const ngo = new NGO({
    name,
    location,
    causeArea,
    contactPerson,
    mobileNumber,
    email,
    address,
    vision,
    mission,
    password
  });

  await ngo.save();
  res.status(201).json({
    message: "NGO registered successfully",
    ngo: {
      ...ngo.toObject(),
      password: undefined, // Don't send the password back in the response
    },
  });
} catch (error) {
  res.status(500).json({ error: "Server error", details: error.message });
}


xports.loginNGO = async (req, res) => {
try {
  const { email, password } = req.body;

  if (!email || !password) {
    return res.status(400).json({ message: "Email and password are required" });
  }

  const ngo = await NGO.findOne({ email });
  if (!ngo) {
    return res.status(400).json({ message: "Invalid email" });
  }

  const isMatch = await bcrypt.compare(password, ngo.password);
  if (!isMatch) {
    return res.status(400).json({ message: "Invalid password" });
  }
```

Figure 3.19: **Killed Mutants**

```
    if (!password) {
      return res.status(400).json({ message: "Password is required to update the NGO profile" });
    }

    const ngo = await NGO.findOne({ email });

    if (!ngo) {
      return res.status(404).json({ message: "NGO with the specified email not found" });
    }

    const updateData = {
      name,
      location,
      causeArea,
      contactPerson,
      mobileNumber,
      address,
      vision,
      mission,
    };



    const updatedNGO = await NGO.findByIdAndUpdate(ngo._id, updateData, { new: true });

    res.status(200).json({ message: "NGO profile updated successfully", updatedNGO });
  } catch (error) {
    console.error("Error updating NGO profile:", error.message);
    res.status(500).json({ message: "Error updating NGO profile", details: error.message });
  }
};

exports.viewPendingRequests = async (req, res) => {
  try {
    const pendingNGOs = await NGO.find({ verificationStatus: "pending" });

    if (pendingNGOs.length === 0) {
      return res.status(404).json({ message: "No pending verification requests found" });
    }

    res.status(200).json({ pendingNGOs });
  } catch (error) {
    res.status(500).json({ message: "Server error" });
  }
};
```

Figure 3.20: **Survived Mutants**

3.7 Payment Controller

The following report shows the results of mutation testing conducted on the `paymentController.js` file.

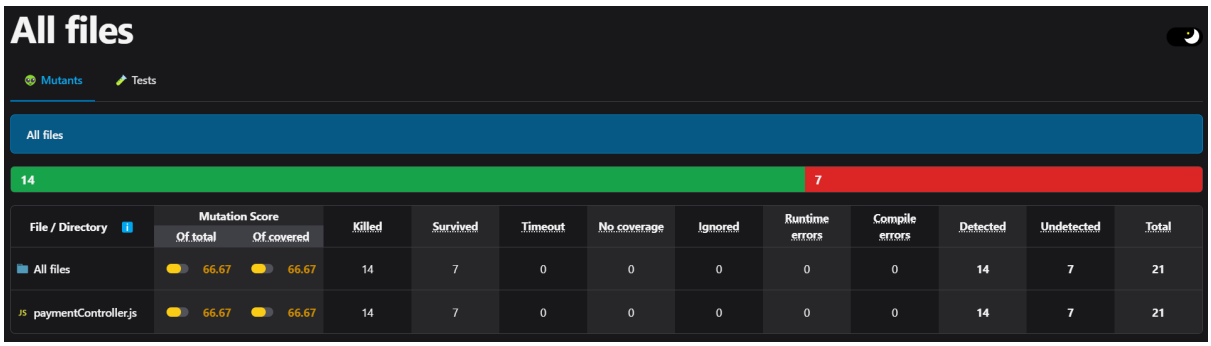


Figure 3.21: Mutation Testing Results for Payment Controller

3.7.1 Mutation Score

In the following test the mutation score is 66.67%

3.7.2 Mutation Breakdown

```
const paymentService = require('../services/payment');

// Endpoint to create a "mock" payment and generate a transaction token
exports.createMockPayment = async (req, res) => {
    try {
        const { name, mobileNumber, ngoName, amount } = req.body;

        // Check if all fields are provided
        if (!name || !mobileNumber || !ngoName || !amount) {
            return res.status(400).json({ error: 'All fields are required' });
        }

        // Generate a transaction token and save payment details
        const mockPaymentDetails = await paymentService.createMockTransaction({
            name,
            mobileNumber,
            ngoName,
            amount
        });

        // Respond with the mock transaction token and details
        res.status(200).json(mockPaymentDetails);
    } catch (error) {
        console.error('Error creating mock payment:', error);
        res.status(500).json({ error: 'Failed to initiate mock payment' });
    }
};
```

Figure 3.22: **Killed Mutants**

```
const paymentService = require('../services/payment');

// Endpoint to create a "mock" payment and generate a transaction token
exports.createMockPayment = async (req, res) => {
    try {
        const { name, mobileNumber, ngoName, amount } = req.body;

        // Check if all fields are provided
        if (!name || !mobileNumber || !ngoName || !amount) {
            return res.status(400).json({ error: 'All fields are required' });
        }

        // Generate a transaction token and save payment details
        const mockPaymentDetails = await paymentService.createMockTransaction({
            name,
            mobileNumber,
            ngoName,
            amount
        });

        // Respond with the mock transaction token and details
        res.status(200).json(mockPaymentDetails);
    } catch (error) {
        console.error('Error creating mock payment:', error);
        res.status(500).json({ error: 'Failed to initiate mock payment' });
    }
};
```

Figure 3.23: **Survived Mutants**

3.8 Review Controller

The following report shows the results of mutation testing conducted on the `reviewController.js` file.

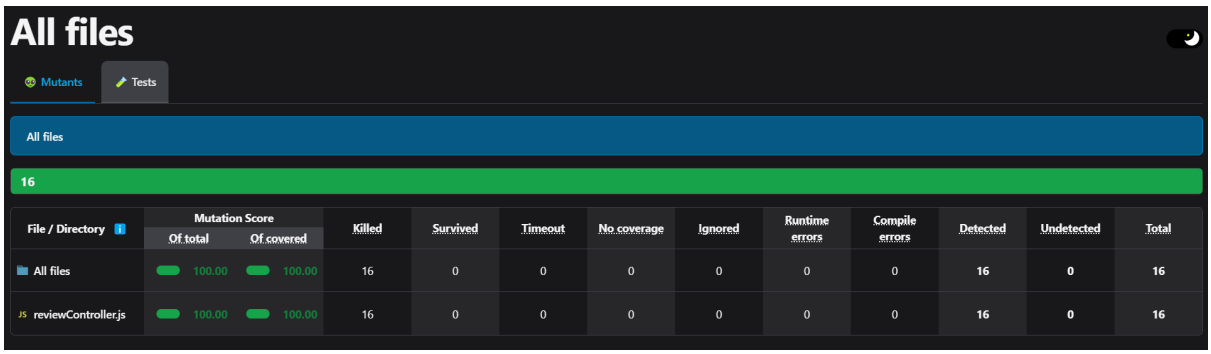


Figure 3.24: Mutation Testing Results for Review Controller

3.8.1 Mutation Score

In the following test the mutation score is 100.00%

3.8.2 Mutation Breakdown

```
const Review = require('../models/Review');
const WebsiteReview = require('../models/WebsiteReview');

// Controller for adding a donor review
exports.addReview = async (req, res) => {
  try {
    const { donorName, message, ngoName } = req.body;
    const review = new Review({ donorName, message, ngoName });
    await review.save();
    res.status(201).json({ message: 'Review submitted successfully' });
  } catch (error) {
    res.status(500).json({ message: 'Error submitting review', error });
  }
};

// Controller for adding a website review
exports.addWebsiteReview = async (req, res) => {
  try {
    const { reviewerName, reviewerType, message } = req.body;
    const websiteReview = new WebsiteReview({ reviewerName, reviewerType, message });
    await websiteReview.save();
    res.status(201).json({ message: 'Website review submitted successfully' });
  } catch (error) {
    res.status(500).json({ message: 'Error submitting website review', error });
  }
};
```

Figure 3.25: **Killed Mutants**