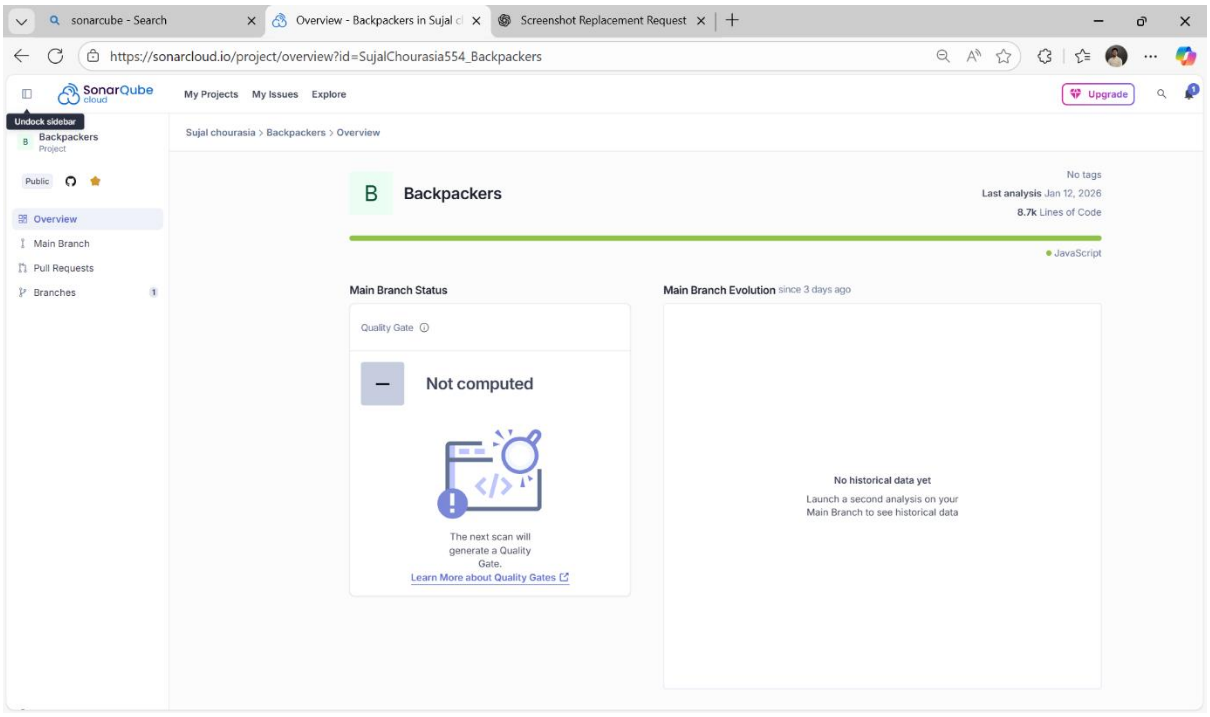
# Static Application Security Testing (SAST) Report

## 1. Introduction

Static Application Security Testing (SAST) is a security testing methodology used to analyze an application's source code without executing it. The goal of SAST is to identify security vulnerabilities, insecure coding practices, and potential risks early in the software development lifecycle.

This report documents the SAST analysis performed on the Studypilot application using SonarCloud.



https://sonarcloud.io/project/overview?id=SujalChourasia554_Backpackers

Backpackers
Project

Sujal chourasia > Backpackers > Overview

B Backpackers

No tags
Last analysis Jan 12, 2026
8.7k Lines of Code
● JavaScript

Main Branch Status

Not computed

The next scan will generate a Quality Gate.
Learn More about Quality Gates

Main Branch Evolution since 3 days ago

No historical data yet
Launch a second analysis on your Main Branch to see historical data

## 2. Tool Used

Tool Name: SonarCloud
Type: Static Application Security Testing (SAST)
Platform: Cloud-based
Integration: GitHub Analysis
Mode: Static source code analysis (no runtime execution)

## 3. Project Details

Project Name: GoTrip
Repository Owner: Sujal Chourasaia
Branch Analysed: main Lines of Code: ~8.7k
Scan Date: January 2026
Technology Stack: MERN Stack

## 4. Methodology

1. Source code hosted on GitHub
2. SonarCloud integrated using GitHub authentication
3. Repository imported and analysed automatically
4. Static analysis performed without executing the application
5. Results reviewed from SonarCloud dashboard

## 5. Scan Summary

Line of code Analuzed: ~8.7k

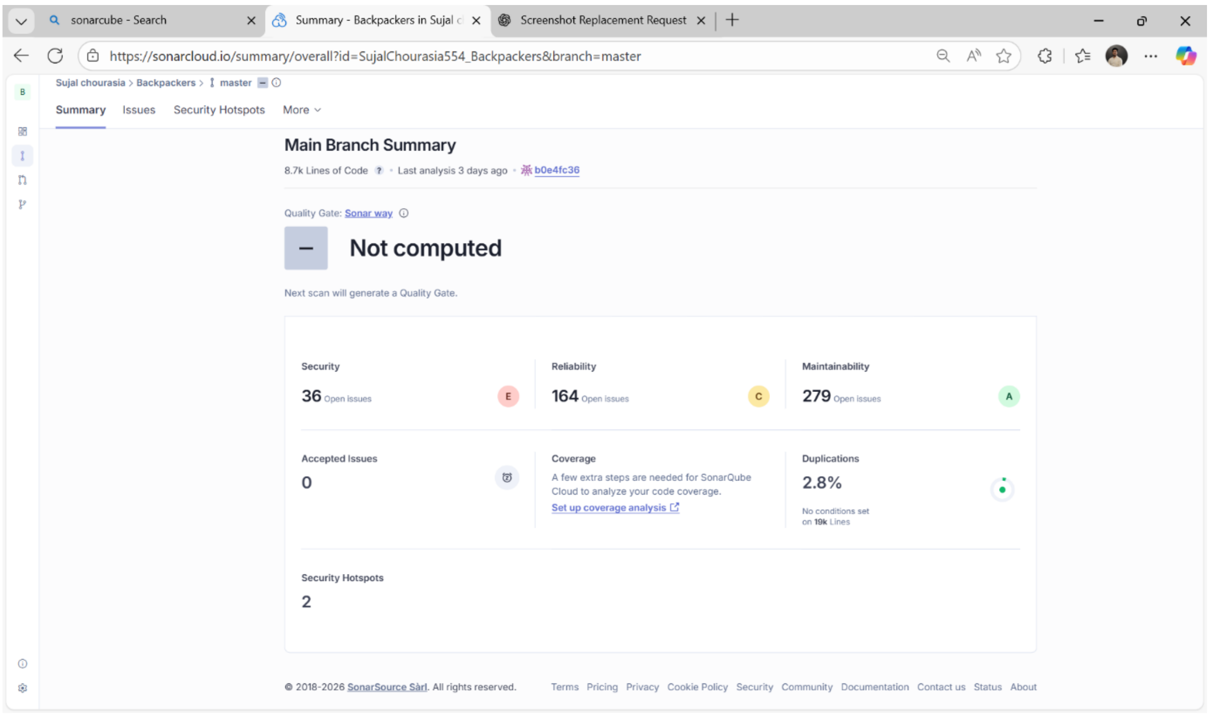
Quality gate: Not Compluted(Baseline scan)

Security Issues: 36

Reliability Issue: 164

Maintainability Issues: 279

Security Hotspots: 2


sonarcube - Search
Summary - Backpackers in Sujal c
Screenshot Replacement Request
Sujal chourasia > Backpackers > master
Summary
Issues
Security Hotspots
More

https://sonarcloud.io/summary/overall?id=SujalChourasia554_Backpackers&branch=master

Main Branch Summary

8.7k Lines of Code · Last analysis 3 days ago · b0e4fc36

Quality Gate: Sonar way

Not computed

Next scan will generate a Quality Gate.

Security
36 Open issues E

Reliability
164 Open issues C

Maintainability
279 Open issues A

Accepted Issues
0

Coverage
A few extra steps are needed for SonarQube Cloud to analyze your code coverage.
Set up coverage analysis

Duplications
2.8%
No conditions set on 19k Lines

Security Hotspots
2

© 2018-2026 SonarSource Sàrl. All rights reserved.

## 6. Identified Vulnerabilities

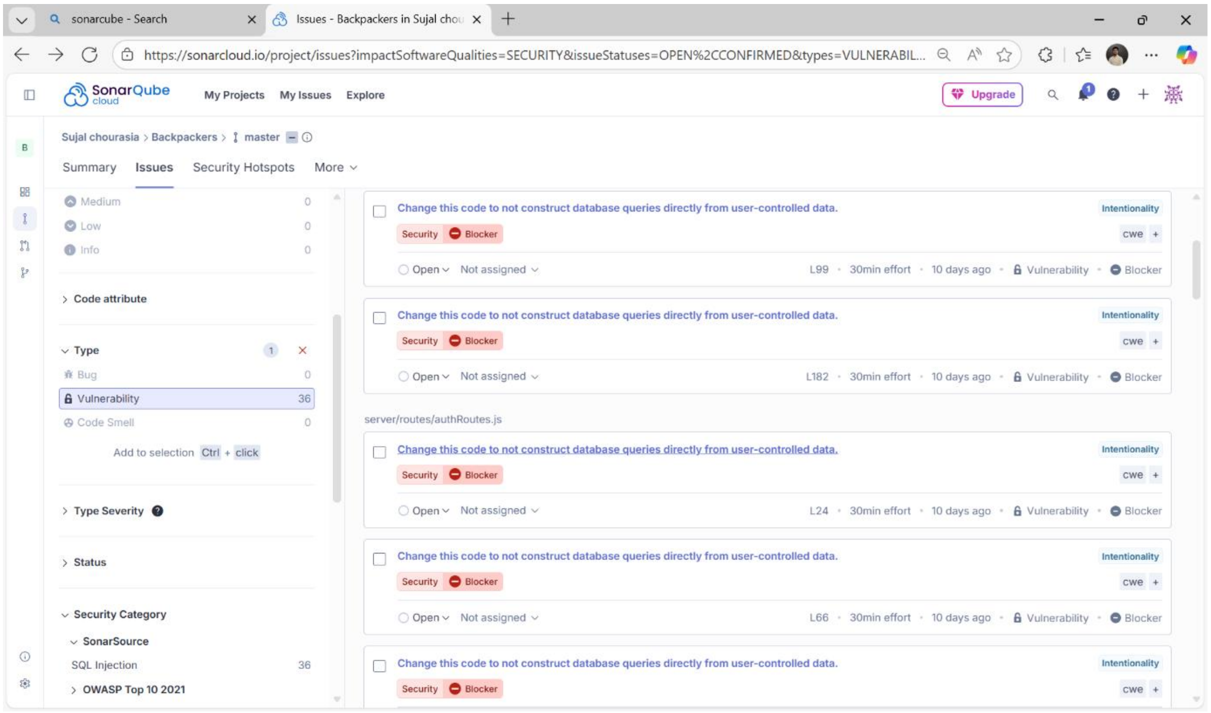
Example Vulnerability: NoSQL Injection
Severity: Blocker
File:

- server/routes/adminRoutes.js (Lines: 66, 99)
- server/routes/authRoutes.js (Lines: 24, 182)

Description:
SonarCloud detected multiple NoSQL Injection vulnerabilities where database operations are constructed directly using user-controlled input (req.body). Unvalidated data is passed into MongoDB query and create operations, allowing attackers to inject malicious NoSQL operators and manipulate database behaviour.

Impact:
This vulnerability can lead to unauthorized data access, data manipulation, and potential compromise of application integrity.

Recommendation:
Validate user input and avoid passing untrusted data directly into database queries


sonarcube - Search
Issues - Backpackers in Sujal chou
https://sonarcloud.io/project/issues?impactSoftwareQualities=SECURITY&issueStatuses=OPEN%2CCONFIRMED&types=VULNERABIL...
SonarQube cloud
My Projects
My Issues
Explore
Upgrade
Sujal chourasia > Backpackers > master
Summary
Issues
Security Hotspots
More

Medium 0
Low 0
Info 0

Code attribute

Type 1
Bug 0
Vulnerability 36
Code Smell 0
Add to selection Ctrl + click

Type Severity

Status

Security Category
SonarSource
SQL Injection 36
OWASP Top 10 2021

Change this code to not construct database queries directly from user-controlled data. Intentionality
Security Blocker cwe
Open Not assigned L99 · 30min effort · 10 days ago · Vulnerability · Blocker

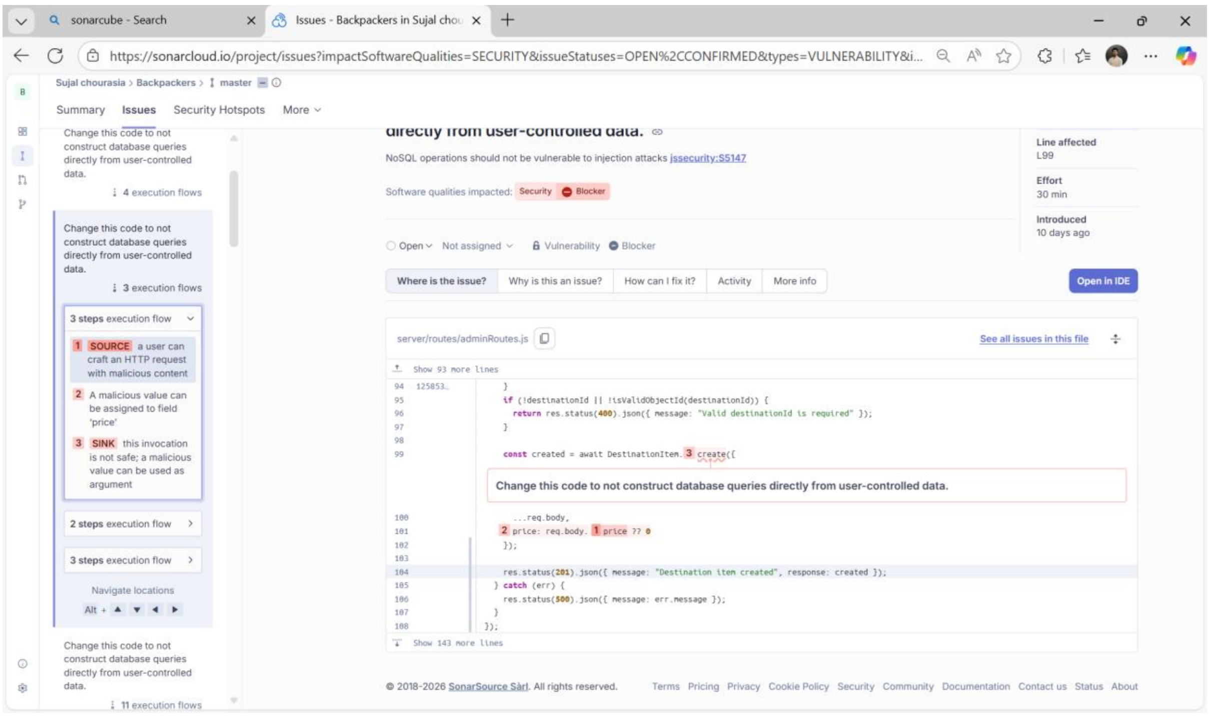
Change this code to not construct database queries directly from user-controlled data. Intentionality
Security Blocker cwe
Open Not assigned L182 · 30min effort · 10 days ago · Vulnerability · Blocker

server/routes/authRoutes.js

Change this code to not construct database queries directly from user-controlled data. Intentionality
Security Blocker cwe
Open Not assigned L24 · 30min effort · 10 days ago · Vulnerability · Blocker

Change this code to not construct database queries directly from user-controlled data. Intentionality
Security Blocker cwe
Open Not assigned L66 · 30min effort · 10 days ago · Vulnerability · Blocker

Change this code to not construct database queries directly from user-controlled data. Intentionality
Security Blocker cwe



Change this code to not construct database queries directly from user-controlled data.
4 execution flows

Change this code to not construct database queries directly from user-controlled data.
3 execution flows

3 steps execution flow
1 SOURCE a user can craft an HTTP request with malicious content
2 A malicious value can be assigned to field 'price'
3 SINK this invocation is not safe; a malicious value can be used as argument

2 steps execution flow

3 steps execution flow

Change this code to not construct database queries directly from user-controlled data.
11 execution flows

directly from user-controlled data.

NoSQL operations should not be vulnerable to injection attacks jssecurity:S5147

Software qualities impacted: Security Blocker

Line affected
L99

Effort
30 min

Introduced
10 days ago

server/routes/adminRoutes.js

```
Show 93 more lines
125853…  }
      if (!destinationId || !isValidObjectId(destinationId)) {
        return res.status(400).json({ message: "Valid destinationId is required" });
      }

      const created = await DestinationItem.3 create({
```

Change this code to not construct database queries directly from user-controlled data.

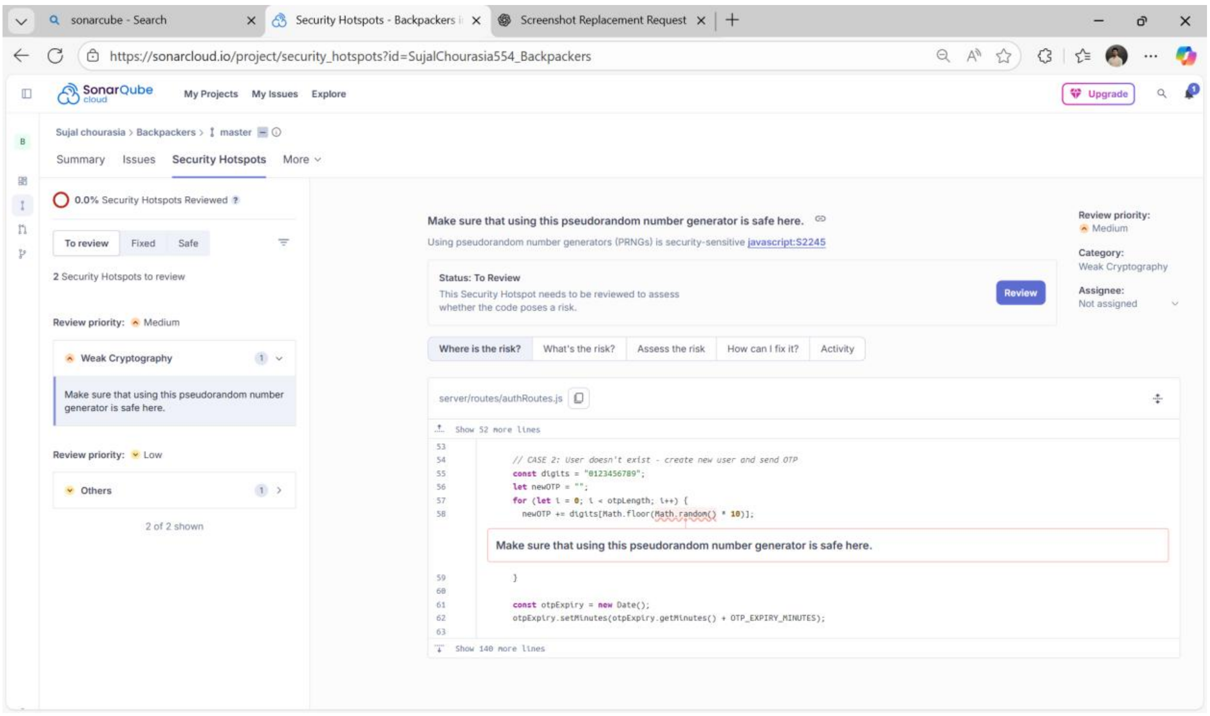
```
        ...req.body,
      2 price: req.body.1 price ?? 0
      });

      res.status(201).json({ message: "Destination item created", response: created });
    } catch (err) {
      res.status(500).json({ message: err.message });
    }
  });
Show 143 more lines
```

## 7. Identified Vulnerabilities

Security hotspots represent sensitive areas in the source code that may introduce security risks and therefore require **manual review** to confirm whether they are exploitable.

Example Hotspot:
Category: Injection

Issue: Use of user-controlled input in database operations without explicit validation.


sonarcube - Search
Security Hotspots - Backpackers
Screenshot Replacement Request
https://sonarcloud.io/project/security_hotspots?id=SujalChourasia554_Backpackers
SonarQube cloud
My Projects
My Issues
Explore
Upgrade
Sujal chourasia > Backpackers > master
Summary
Issues
Security Hotspots
More
0.0% Security Hotspots Reviewed
To review
Fixed
Safe
2 Security Hotspots to review
Review priority: Medium
Weak Cryptography
Make sure that using this pseudorandom number generator is safe here.
Review priority: Low
Others
2 of 2 shown

Make sure that using this pseudorandom number generator is safe here.

Using pseudorandom number generators (PRNGs) is security-sensitive javascript:S2245

Status: To Review

This Security Hotspot needs to be reviewed to assess whether the code poses a risk.

Review priority: Medium

Category: Weak Cryptography

Assignee: Not assigned

Review
Where is the risk?
What's the risk?
Assess the risk
How can I fix it?
Activity
server/routes/authRoutes.js
Show 52 more lines

```

    // CASE 2: User doesn't exist - create new user and send OTP
    const digits = "0123456789";
    let newOTP = "";
    for (let i = 0; i < otpLength; i++) {
      newOTP += digits[Math.floor(Math.random() * 10)];
```

Make sure that using this pseudorandom number generator is safe here.

```
    }

    const otpExpiry = new Date();
    otpExpiry.setMinutes(otpExpiry.getMinutes() + OTP_EXPIRY_MINUTES);

```

## 8. Conclusion

The Static Application Security Testing (SAST) analysis conducted using **SonarCloud** on the **Backpackers** application identified multiple **Blocker-level security vulnerabilities**, primarily related to **NoSQL Injection** caused by the direct use of user-controlled input in database operations.

These findings highlight the importance of secure input handling and adherence to secure coding practices. Addressing the reported vulnerabilities by implementing proper input validation, sanitization, and safe database query construction will significantly enhance the **security, reliability, and maintainability** of the application.

## 9. Appendix

Tool: SonarCloud

Scan Type: Static Analysis

Report Prepared By: Sujal Chourasia