Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

TRiM - A Software Solution for Barber Shops

By
Ariel C. Nunes

February 16, 2026

Minor Dissertation

Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.

B.Sc. (Hons) in Software Development

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Small businesses such as barber shops can oftentimes be faced with less than ideal situations if their operation is walk-ins only. Things like lack of business insights, customer friction due to inability to be serviced when they wish, manual bookkeeping, etc. In today's day and age, digitalizing your business is a must, not only because it's what everyone is already doing, but also because it builds a bridge between the business and its customers, leading to a more established presence in the market.

By leveraging technologies such as *TRiM*, we are able to scale the business and bring real value to the table, and not just financial value, but also operational value as this tool aims to improve every corner the business. But what exactly is *TRiM*? This platform aims to support all operations of service-based businesses such as barber shops, from customer bookings, to payment integration, and employee management. The goal is to allow multiple businesses to use the platform independently so that each one of them can have access to all the features offered by *TRiM*.

1.2 Project Objectives

The objective of this project is to deliver a real and robust software solution to local service-based businesses. The following goals have been defined:

- **Customer Booking System**: Enable customers to book services online, select their preferred barber, choose from categorized services, and pick available time slots while the system prevents double bookings and scheduling conflicts.

- **Payment Integration**: Integrate Stripe for secure online payment processing, supporting both full online payments and deposit-based bookings where customers pay a percentage upfront and settle the outstanding balance in-shop (client requirement).
- **Business Administration**: Provide business owners with a comprehensive admin dashboard to manage service categories, individual services with pricing and duration, and employee (barber) profiles. Include analytics such as total revenue, booking statistics, and popular services.
- **Barber Management**: Allow barbers to manage their own availability by setting weekly working hours, defining break periods, and viewing/managing their upcoming bookings with the ability to confirm, complete, or handle cancellations and no-shows.
- **Customer Management**: Enable administrators to view customer history, track no-show counts, and blacklist problematic customers to protect business operations.
- **Multi-Tenant Architecture**: Design the platform to support multiple independent businesses, each with their own subdomain (business slug), isolated data, and separate admin/barber/customer user bases.
- **Role-Based Access Control**: Implement distinct user roles (Customer, Barber, Admin) with appropriate permissions and tailored interfaces for each role's specific needs.

1.3 Dissertation Focus

The focus of this dissertation is mostly architectural, with a focus on the database design pattern and data isolation techniques applied. With *TRiM*, we're exploring how to create a multi-tenant software application that's scalable and robust across the board, but also leverages efficient design patterns that allows us to grow without massive costs or performance overhead. We also investigate how to make sure multiple business can have access to this system whilst maintaining performance and security at the top of our priorities.

1.4 Success Metrics

From a functional standpoint, the system must handle all core operations reliably: bookings should complete without conflicts, payments should process securely, and the admin dashboard should provide accurate insights into business operations.

Beyond functionality, performance represents a critical area of evaluation. The system will be benchmarked under various conditions to assess its resilience. This includes measuring response times for key API endpoints, evaluating database performance as the number of tenants and bookings increases, and verifying that heavy usage by one business does not degrade the experience for others. Tenant isolation extends beyond security considerations, it also ensures fair distribution of system resources across all users.

Security constitutes another essential metric. With multiple businesses and their customer data at stake, the system must demonstrate robust protection mechanisms. Cross-tenant data leakage would represent a critical failure, necessitating thorough testing of all isolation mechanisms to prevent such occurrences.

Finally, usability remains a key consideration. The primary objective of *TRiM* is to streamline operations for business owners and their customers. An unintuitive interface or cumbersome booking flow would undermine the value of the underlying architecture.

1.5 Chapters Overview

Give an overview of all the chapters.

Chapter 2

Methodology

2.1 Development Approach

2.1.1 Agile Development

2.1.2 Project Management with GitHub Projects

2.1.3 Version Control (GitHub)

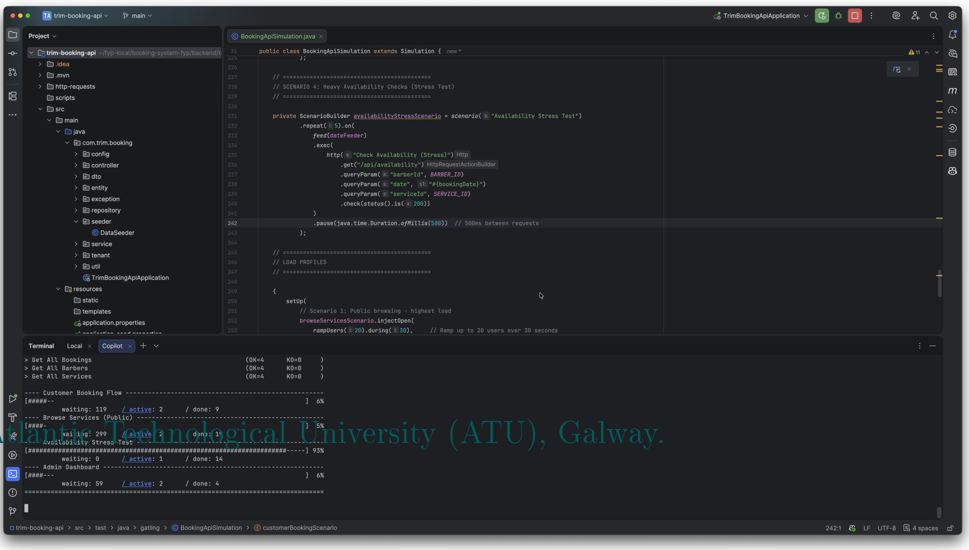
2.2 Research Approach

2.2.1 Literature Search

2.2.2 Choosing the Technology Stack

2.3 Evaluation Methodology

2.3.1 Benchmarking



```
public class BookingApiSimulation extends Simulation {
            );

    // ==========================================
    // SCENARIO 4: Heavy Availability Checks (Stress Test)
    // ==========================================

    private ScenarioBuilder availabilityStressScenario = scenario("Availability Stress Test")
            .repeat(5).on(
                feed(dateFeeder)
                .exec(
                    http("Check Availability (Stress)")
                        .get("/api/availability")
                        .queryParam("barberId", BARBER_ID)
                        .queryParam("date", "#{bookingDate}")
                        .queryParam("serviceId", SERVICE_ID)
                        .check(status().is(200))
                )
                .pause(java.time.Duration.ofMillis(500))  // 500ms between requests
            );

    // ==========================================
    // LOAD PROFILES
    // ==========================================

    {
        setUp(
            // Scenario 1: Public browsing - highest load
            browseServicesScenario.injectOpen(
                rampUsers(20).during(30),     // Ramp up to 20 users over 30 seconds
```

```
> Get All Bookings                                  (OK=4      KO=0     )
> Get All Barbers                                   (OK=4      KO=0     )
> Get All Services                                  (OK=4      KO=0     )

---- Customer Booking Flow ------------------------------------------------------
[#####--                                                                    ]  6%
          waiting: 119    / active: 2      / done: 9
---- Browse Services (Public) ---------------------------------------------------
[####-                                                                      ]  5%
          waiting: 299    / active: 2      / done: 19
---- Availability Stress Test ---------------------------------------------------
[######################################################################-----] 93%
          waiting: 0      / active: 1      / done: 14
---- Admin Dashboard ------------------------------------------------------------
[####---                                                                    ]  6%
          waiting: 59     / active: 2      / done: 4
=================================================================================
```

trim-booking-api
main
TrimBookingApiApplication
Project
BookingApiSimulation.java
Terminal
Local
Copilot
trim-booking-api > src > test > java > gatling > BookingApiSimulation > customerBookingScenario

Figure 2.1: Benchmarking API in IntelliJ IDE.

2.3.2 Test Data Generation

2.3.3 Performance Metrics

Chapter 3

Technology Review

3.1 Multi-Tenant Database Architectures

3.1.1 Different Types of Multi-Tenant Database Designs

3.1.2 Trade-offs: Isolation vs. Efficiency vs. Cost

3.1.3 Data Isolation Techniques

3.2 Related Works

3.2.1 Multi-Tenant SaaS Database Design Patterns

3.2.2 Literature Review

Chapter 4

System Design

4.1 System Overview

4.1.1 Architecture Diagram

4.1.2 RESTful API Design

4.1.3 Technology Stack

4.2 Multi-Tenant Database Design

4.2.1 Entity-Relationship Model

4.2.2 Tenant-Aware Schema

4.2.3 Data Isolation Strategy

4.2.4 Migration From Single to Multi-Tenant

4.3 Multi-Tenant Implementation

4.3.1 TenantFilter & TenantContext

4.3.2 Subdomain-based Routing

4.4 Business Operations

4.4.1 Race Condition Prevention

4.4.2 Booking & Payment Flow

4.4.3 Cross-Subdomain Token Exchange

4.4.4 Authentication & Security

4.4.5 JWT & Role-Based Access Control

4.5 Frontend Architecture

4.5.1 Component Structure

and algorithms to illustrate your design.

Chapter 5

System Evaluation

5.1 Performance Evaluation

5.1.1 Benchmark Results Before and After Optimization

5.2 Security Evaluation

5.2.1 Tenant Isolation Verification

5.3 Critical Analysis

5.3.1 Strengths

5.3.2 Limitations

Chapter 6

Conclusion

6.1 Future Work

Briefly summarise your context and objectives. Remind the reader about the overall rationale and goals of the project. Highlight your findings from the System Evaluation chapter.

Bibliography

[1] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.