TQS: Product specification report

Henrique Lopes [119954]
v2025-04-25

1 Introduction

1.1 Overview of the project

This project is developed within the scope of the TQS (Software Testing and Quality) course, where the main objective is to design, implement, and validate a software system while applying systematic Software Quality Assurance and DevOps practices. Throughout the assignment, the team must incorporate continuous integration, automated testing, code quality monitoring, and agile project management.

The product developed for this project is *ToolShed*, a digital platform that enables users to share and rent tools and equipment securely and efficiently.

ToolShed connects tool owners, who can monetize underused items, with renters who need temporary access for personal or professional projects. The platform supports searching, booking, managing tool listings, and maintaining trust between participants through a reputation and review system. ToolShed is designed for individuals, hobbyists and small businesses seeking affordable, on-demand access to tools without the need to purchase them.

1.2 Project limitations and know issues

The main limitations are:

- **Payment processing is simulated**, and no real payment gateway is integrated (e.g., MB Way or PayPal)

1.3 References and resources

During the development of the ToolShed platform, several technologies, tools, and learning resources were essential to ensure a robust, maintainable, and well-tested solution.

Core Technologies:

- **React** (https://react.dev/): Used for building the frontend interface. Its component-based structure and extensive documentation made it easier to create reusable UI elements.
- **Spring Boot** (https://spring.io/projects/spring-boot): The foundation of our backend services. Spring Boot's opinionated, "convention over configuration" approach allowed rapid development of REST APIs.
- **PostgreSQL** (https://www.postgresql.org/): The relational database chosen for storing users, tools, bookings, and platform analytics.
- **Stripe** (https://stripe.com/): Used in test mode to simulate secure payments, deposits, and future refund or damage-fee workflows.
- **GeoAPI** (https://geoapi.pt/): Used to support location-based search and geographic data resolution for tools and users.

API Development & Documentation:

- **Swagger / OpenAPI** (https://swagger.io/): Used to document and validate our REST API endpoints. This tool made it easier to maintain API consistency, guide frontend–backend integration.

- **NGINX** (https://nginx.org/en/): Used as a reverse proxy and entry point for the platform's backend API. NGINX allowed us to manage request routing, static asset serving, security headers, and load balancing.

DevOps, Deployment, and CI/CD:

- **Docker** (https://docs.docker.com/): Essential for containerizing the backend, database, and supporting services. Docker ensured consistency between development and deployment environments.
- **GitHub Actions** (https://docs.github.com/en/actions): Used to implement Continuous Integration and Continuous Deployment (CI/CD), running automated tests and build workflows on every commit.
- **SonarQube** (https://www.sonarsource.com/products/sonarqube/): Applied for static code analysis, code smells detection, and monitoring overall code quality and coverage.

2 Product concept and requirements

2.1 Vision statement

ToolShed aims to solve the accessibility and affordability problem associated with owning specialized tools and equipment. Many individuals and small businesses need occasional access to tools but cannot justify the cost, storage, or maintenance of owning them. ToolShed provides a simple, reliable, and secure digital platform that enables users to rent, share, and manage tools within their community.

As a functional, black-box system, ToolShed enables renters to search for tools, view item details, book rentals, filter listings by location, and handle payments. Tool owners can create listings, update availability, manage booking requests, and track their inventory. The platform also supports core user management, authentication, platform analytics, and a reputation system with ratings and evaluations.

ToolShed is conceptually similar to other rental marketplaces (e.g., Fat Llama), but it differentiates itself through its emphasis on simplicity, a structured owner/renter workflow, and a more focused domain (tool sharing). The system is designed to be extendable to additional categories in the future without significant restructuring.

The requirements were gathered by analyzing typical rental use cases, mapping business needs to user stories, and prioritizing features that support the essential epics: item discovery, booking & scheduling, item management, platform admin operations, and basic user management. The backlog reflects these priorities, ensuring a clear path for delivering a minimal yet functional and testable MVP.

2.2 Personas and scenarios

Urban Renter: Maria Lontra

"I just want to quickly find a tool near me, rent it without complications, and finish my small home projects without spending too much."

Profile
Name: Maria Lontra
Age: 27
Gender: Female
Occupation: Junior Architect
Location: Coimbra, Portugal
Income: Middle income
Tech familiarity: Comfortable with apps, frequent mobile user

Description

Maria lives in a small apartment and doesn't own many tools. Whenever she wants to hang a shelf, fix furniture, or do a small renovation, she needs tools temporarily. She values affordability, proximity, and reliability. She dislikes the high cost of buying tools for one-time use and wants a fast, trustworthy rental process.

Goals

- Find nearby tools quickly
- Compare prices and availability
- Rent tools for very short periods
- See clear details about the item condition
- Track ongoing and past rentals

Frustrations

- Tools being too far away
- Lack of clear pictures or condition details
- Overpriced rentals
- Not knowing if a tool will actually be available
- Confusing or slow booking flows

Behaviours

- Browses from her phone on the way home
- Uses keyword search and location filters often
- Reads reviews before renting

- Prefers chat or messaging over calls

Needs

- Fast and intuitive search
- Accurate availability information
- Clear pricing
- A smooth reservation/confirmation flow
- Secure payments and receipts
- Ability to review owners and tools after use

Scenario: Finding and Renting a Tool

It's Saturday morning, and Maria wants to assemble a new shelf. She opens ToolShed and searches for a power drill. Using the location filter, she finds one 500 meters from her apartment. After reviewing price, availability, and photos, she selects a two-hour rental slot. She receives a confirmation from the owner, picks up the drill, completes her project, and later submits a review about the experience.

Casual Tool Owner: Simão Total

"I have tools sitting around, so why not make some extra money? I just need a simple way to manage bookings and avoid scheduling conflicts."

Profile
Name: Simão Total
Age: 42
Gender: Male
Occupation: Logistics Supervisor
Location: Porto, Portugal
Income: Middle–high income
Tech familiarity: Average, uses apps when needed

Description

Simão owns several tools he uses occasionally, drills, sanders, saws, and gardening tools. Most of the time, they sit unused in storage. He wants to earn extra income by renting them out but insists on maintaining control over whom he lends to and avoiding overlapping reservations.

Goals

- Monetize unused tools
- Easily approve/decline booking requests
- Control availability and pricing
- Keep tools in good condition
- Avoid double bookings

Frustrations

- Having to message renters repeatedly
- Users not returning tools on time
- Lack of visibility into upcoming reservations
- Managing availability manually

Behaviours

- Checks the app mainly in the evening
- Updates availability once per week
- Reads renter reviews before accepting requests
- Uses a desktop more often than mobile

Needs

- A clear overview of requests
- A simple dashboard for upcoming rentals
- Controls to activate/deactivate tools
- Messaging or clear renter profiles
- Notifications for new requests and returns

Scenario: Managing Tools and Requests

Simão logs in after work and sees two new booking requests for his drill. He reviews the renter profiles, checks the dates, and approves one while declining the other due to a conflict. He updates the tool's availability for the following week and uploads a new photo to improve his listing.

System Supervisor: Martim Jonas

"I need to keep the platform healthy, if something goes wrong, I must detect it before users feel it."

Profile

Name: Martim Jonas
Age: 35
Gender: Male
Occupation: Platform Administrator
Location: Aveiro, Portugal
Income: Stable, mid-level
Tech familiarity: Very high; uses monitoring tools daily

Description

Martim oversees the platform's operations and stability. He needs real-time metrics to detect issues early and ensure smooth user experience. His job depends on visibility: usage charts, error rates, active listings, and booking trends.

Goals

- Monitor technical and business KPIs
- Detect issues before they escalate
- Ensure safety and platform reliability
- Support users who report problems
- Maintain data accuracy and system health

Frustrations

- Missing or delayed metrics
- Not knowing when failures occur
- User complaints arriving before system alerts
- Unclear logs or lack of traceability

Behaviours

- Checks dashboards multiple times a day
- Uses monitoring tools and logs frequently
- Responds quickly to anomalies
- Collaborates with DevOps to diagnose issues

Needs

- A centralized admin dashboard
- KPIs on listings, bookings, revenue, activity
- Error logs and performance graphs
- Alerts for unusual patterns
- Tools to validate users and listings

Scenario: Platform Monitoring

Martim opens the admin dashboard on a Monday morning. He notices a sudden drop in successful bookings. Investigating further, he sees error spikes in the booking API. He alerts the DevOps team, checks logs, and identifies a recent deployment issue. After the fix, the KPIs return to normal, and Martim closes the incident.

2.3 Project epics and priorities

Epics

EPIC 1: Booking, Scheduling and Item Discovery

Allow renters to quickly discover tools, compare availability, filter by location, and book items for specific dates. The epic includes search, filtering, tool detail exploration, booking flows, and payment processing. The system must show accurate availability, prevent overlapping bookings, and ensure a smooth checkout process.

User Stories:
US1: Search Tools
As a renter
I want to search tools by keywords
So that I can quickly find the tool I need.

Acceptance criteria:

- Search bar is visible in the homepage
- Results must match title or description
- No results → show “No tools found”

US2: Filter tools by location

As a renter
I want to filter tools by location
So that I only see tools near my area.

Acceptance criteria:

- Filter by district/city
- List updates immediately

US3: See tool details

As a renter
I want to view the full details of a tool
So that I can decide whether to rent it.

Acceptance criteria:

- Shows price, description
- Shows owner name and rating
- Availability calendar is displayed

US4: Book a tool

As a renter
I want to book a tool for specific dates
So that I can ensure it is available for me.

Acceptance criteria:

- User selects start and end date
- System validates overlapping bookings
- Confirmation page must appear
- Booking stored in the system

US5: Pay the Rent for a Tool

As a renter
I want to pay the rent for a tool
So that I can complete the rental process and secure my reservation.

Acceptance Criteria

- User must be able to select a tool to pay for
- System must display the rental price and total amount due
- User must choose a payment method
- Payment must be validated and processed successfully
- A payment confirmation page must appear
- Payment details and transaction must be stored in the system

US6: Manage booking requests (owner)

As a tool owner
I want to approve or decline booking requests
So that I can control who rents my tools.

Acceptance criteria:

- Owner receives a notification
- Buttons: “Approve” and “Reject”
- After approval → booking becomes final
- Tool availability automatically updates

EPIC 2: Item Management and User Management

This epic focuses on enabling tool owners to manage their listings and availability, while also providing users with account registration and authentication. These features ensure a secure and organized environment for renters and owners.

User Stories:

US7: Manage tool (CRUD)

As an owner
I want to create, view, update and delete my tools on the platform
So that I can keep my listings accurate and available for renters.

Acceptance Criteria:

- **Create**
  - Form with required fields: title, daily price, description, location
  - When I save, the new tool appears in my owner dashboard
  - If required fields are missing, the system shows validation errors
- **Read / View**
  - I can open a detail page for any of my tools
  - The detail page shows current data (title, price/day, description, status, etc.)
- **Update**
  - I can edit the fields of an existing tool
  - After saving, changes are visible in the dashboard and in the public catalog (if active)
- **Delete**
  - I can delete a tool from my account
  - The system asks for confirmation before deleting
  - Deleted tools no longer appear in search or in my dashboard

US8: Update tool availability

As a owner
I want to mark a tool as active/inactive
So that I can control when it is rentable.

Acceptance criteria:

- Toggle “active/inactive”
- Inactive tools disappear from search
- Owner still sees them in the dashboard

US9: Register as a user

As a user
I want to log in
So that I can access my dashboard.

Acceptance criteria:

- Email + password authentication
- Incorrect login shows error message

EPIC 3: Platform Administration and Reviews

This epic includes administrative functionalities such as viewing platform metrics, and user-generated reviews. These features increase transparency, trust, and system oversight.

User Stories:
US10: See the platform metrics and details

As a admin
I want to see global metrics
So that I can monitor the health of the platform.

Acceptance criteria:

- Number of active users
- Number of active listings

- Number of bookings per month

US11: Evaluate the owner or tool

As a renter
I want to leave a review after a rental
So that I can help future renters.

Acceptance criteria:

- Rating from 1 to 5
- Optional comment
- Review appears in the tool page
- Cannot review without a completed booking

3 Domain model

The ToolShed system is organized into core entities that structure all data and business logic for sharing and renting tools online. Here's a summary of each, and their key relationships:

Entities

User:
Represents anyone registered in the system, either as a renter, owner, or administrator.

- Key Data: Name, email, password, user role (RENTER, OWNER, ADMIN), reputation score, status (AVAILABLE/UNAVAILABLE), registration date.
- Responsibilities: Owns tools, makes bookings, leaves reviews.

Tool:
Represents a single item that can be listed for rent.

- Key Data: Title, description, price per day, location (city/district), owner, active/inactive status, calendar of availability, overall rating, damage status and description.
- Responsibilities: Can be rented, has availability schedule, receives reviews, may be marked as damaged.

Booking:
Records each tool rental period and process.

- Key Data: Associated tool, renter, owner, dates, status (pending, approved, completed, etc.), payment status, total price, tool damage reporting, resolution status.
- Responsibilities: Manages the rental lifecycle; supports reporting of tool damage by either renter or owner.

Payment:
Tracks financial transactions for rentals.

- Key Data: Associated booking, amount, payment method, status (paid, refunded, failed), payment date.
- Responsibilities: Ensures rent is paid and tracked for each booking.

Review:
Allows renters to rate tools or owners after using them.

- Key Data: Associated booking, reviewer, owner, tool, rating (1–5), optional comment, review date.
- Responsibilities: Helps build trust through feedback and public ratings.

AdminMetrics
Aggregates data for platform monitoring and administration.

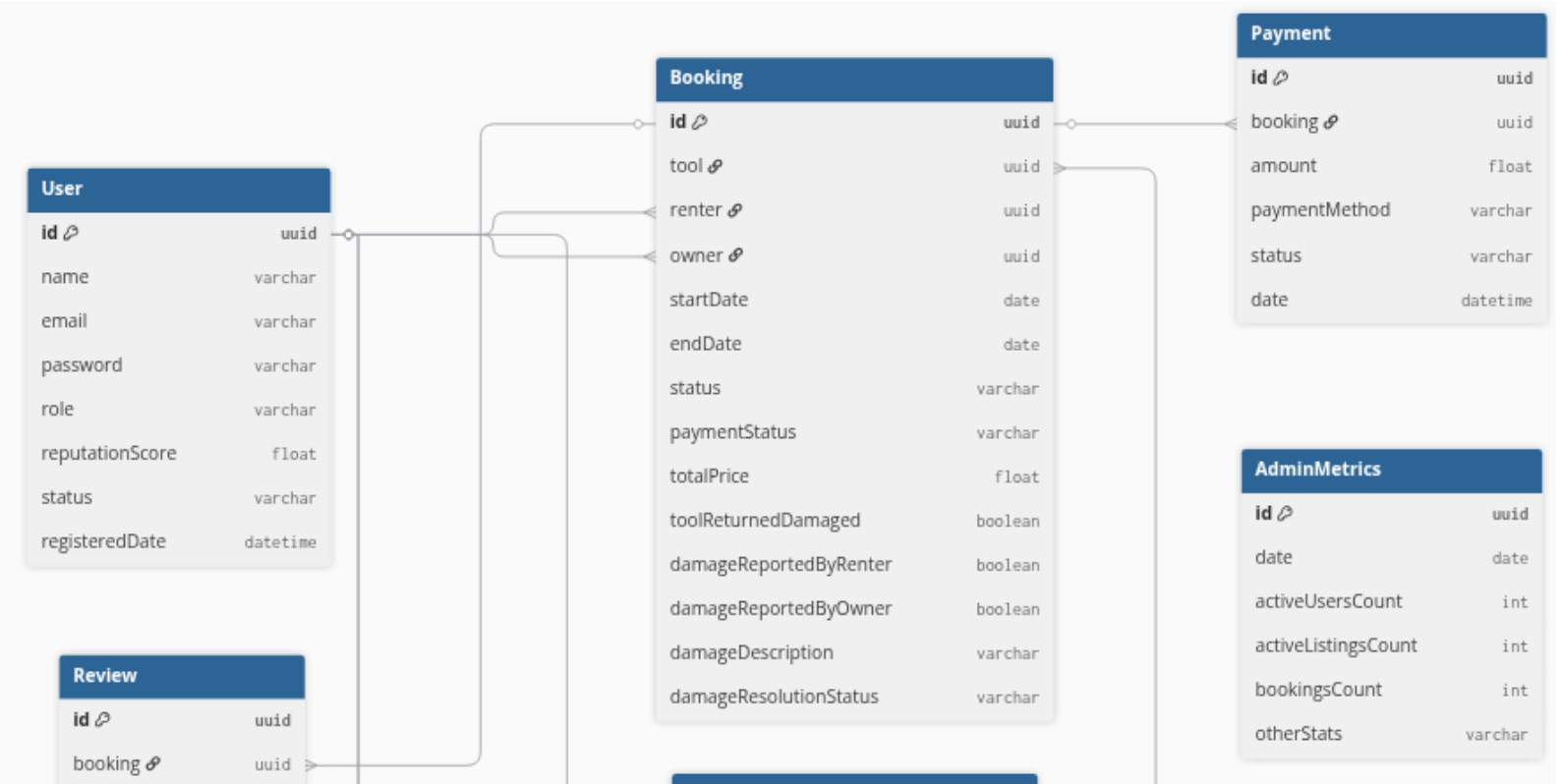
- Key Data: Date, number of active users, active listings, bookings, other summary stats.
- Responsibilities: Used by administrators in platform health and usage monitoring.

Relationships

- A User may own multiple Tools.
- A User (as renter) may create multiple Bookings.
- Each Tool may be booked multiple times (Booking).
- Each Booking is associated with a single Payment.
- Each Booking may result in a single Review from the renter.
- Each Tool may have many Reviews.
- Each User may leave multiple Reviews and own many Tools.

Special Features

- User status helps restrict activity for problematic renters/owners.
- Tool damage logic enables both renters and owners to report problems, triggering further platform actions and resolution.
- The review system ensures reputation and quality control.
- AdminMetrics provides actionable metrics for system administration.



4 Architecture notebook

4.1 Key requirements and constrains

Hardware Depende:
The ToolShed platform is designed as a containerized microservices architecture using Docker, ensuring hardware abstraction and portability. The system isolates infrastructure dependencies through:

- PostgreSQL database container for persistent storage
- NGINX reverse proxy for load balancing and routing
- Stripe API integration for payment processing (external dependency)

No specialized hardware is required; the system runs on commodity servers and is cloud-agnostic.

External Integrations:

Integration	Purpose	Isolation Strategy
Stripe Payment Gateway	Payment processing checkout sessions	Abstracted via *PaymentServiceImpl* with fallback for dev mode
GeoAPI	Location caching for districts	Cached responses in *GeoApiService* with warm-up on startup

User-Interfacing Platforms:
The system is offered as a responsive web application (React + TypeScript) accessible via desktop and mobile browsers. The frontend employs a modular architecture with distinct modules for each user role:

- **Renter Module** – Browse tools, book, pay, leave reviews
- **Owner Module** – Manage tools, approve bookings, track earnings
- **Admin Module** – Platform metrics and user management

Architectural Characteristics:

Characteristic	Priority	Rationale
Scalability	High	Must handle concurrent users and payment requests; validated with k6 load tests
Security	Critical	Payment data handling via Stripe PCI compliance; role-based access control
Testability	High	Comprehensive testing pyramid (unit, integration, BDD) to ensure code quality
Availability	Medium	Containerized deployment enables horizontal scaling and quick recovery
Extensibility	Medium	Modular service layer allows adding new features (e.g., Pro subscriptions) without major refactoring
Usability	High	Intuitive UI for both technical and non-technical users; multi-role support

Unusual Conditions:
The system handles edge cases including:

- Payment failures - Graceful fallback with status tracking (PENDING, FAILED, COMPLETED)
- Booking conflicts - Overlap detection prevents double-booking
- Deposit lifecycle - Security deposits are automatically returned when rentals complete

4.2 Architecture view

The architecture of the ToolShed system follows a layered and modular design, combining elements of microservices and service-oriented architecture (SOA). The system is logically divided into four main functional layers: the Presentation Layer, the Application Layer, the Data Layer, and the External Integration Layer.

1. **Presentation Layer:**

    - **Frontend Module**

- A web-based user interface built with **React** and **TypeScript**, styled using **TailwindCSS** and **shadcn/ui** component library.
- Delivered via a frontend Docker container.
- Communicates with backend services via RESTful APIs through an **NGINX reverse proxy.**
- Organized into role-based modules:
  - Renter Module - Tool browsing, booking, payments, and reviews
  - Supplier Module - Tool management, booking approvals, earnings tracking
  - Admin Module - Platform metrics and user oversight
  - Auth Module - Authentication and registration flows

2. Application Layer (Backend Services)

Hosted in a Spring Boot backend container, the core business logic is distributed across several independent services:

Service	Responsibility
AuthService	Handles user registration, authentication, and role management
ToolServiceImpl	Manages tool CRUD operations, search, and availability
BookingServiceImpl	Manages booking lifecycle, status transitions, cancellations, and refunds
PaymentServiceImpl	Processes Stripe payments, checkout sessions, wallets, and payouts
ReviewServiceImpl	Handles user-to-user and tool reviews with rating aggregation
SubscriptionServiceImpl	Manages Pro Member subscriptions and discount logic
AdminService	Provides platform-wide metrics and analytics
GeoApiService	Caches geolocation data for district-based filtering

3. Data Persistence Layer

- **PostgreSQL Database**
  - Hosted in its own Docker container.
  - Accessed via **Spring Data JPA Repositories**, which provide abstraction over SQL operations.
  - Key entities include: *User, Tool, Booking, Review, Payout, Payment.*

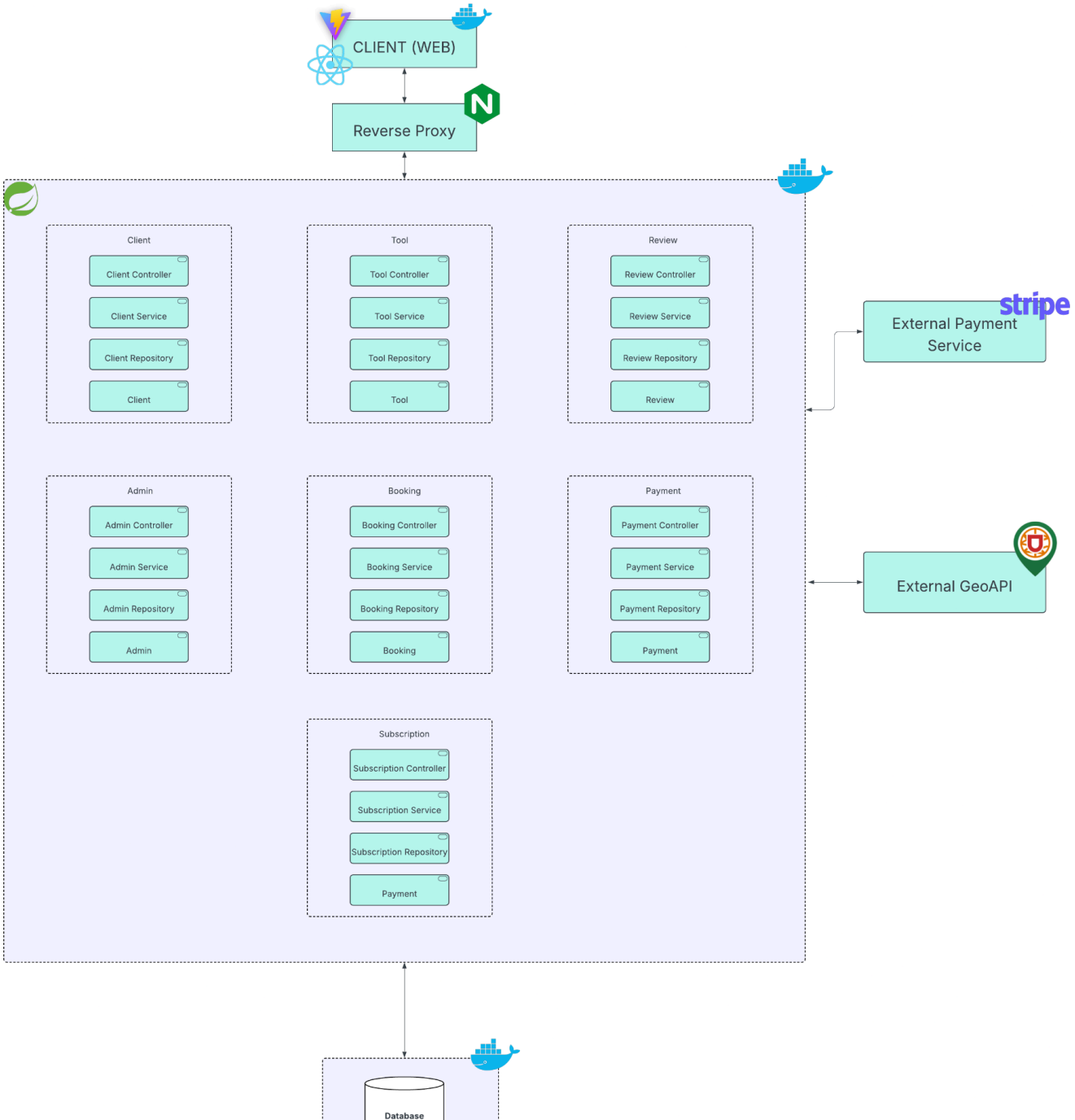
4. Infrastructure Layer

- **Docker Compose Orchestration**
  - All services are containerized and orchestrated via ***docker-compose.yml***

- Components:

  - frontend - React application container
  - backend - Spring Boot API container
  - db - PostgreSQL database container
  - nginx - Reverse proxy for routing and load balancing

5. Architecture Layout



4.3 Deployment view (production configuration)

To maintain consistency across development, testing, and production environments, the ToolShed system is deployed using four main Docker containers, each encapsulating a distinct component of the application. These containers are orchestrated via Docker Compose and interact with one another through an internal Docker network.

User Interface Container (Frontend)

Attribute	Value
Purpose	Serves the web application built with React, TypeScript, and Vite
Functionality	Provides the user interface to Renters, Owners, and Admins. Handles user-initiated actions by making HTTP requests to the backend API
Technology	React 18 + Vite + TailwindCSS + shadcn/ui
Port	Exposed on port 5173
Environment	VITE_STRIPE_PUBLISHABLE_KEY for Stripe integration

Backend Container (API Server)

Attribute	Value
Purpose	Hosts the Spring Boot application containing all business logic
Functionality	Processes booking workflows, payment transactions, review management, and integrates with external APIs (Stripe, GeoAPI)
Technology	Spring Boot 3.x + Spring Data JPA + Hibernate
Port	Exposed on port 8080
Environment Variables	Database connection, Stripe secret key, GeoAPI cache configuration
Volumes	Persistent geo-cache storage at /data/geo-cache.json

Database Container (PostgreSQL)

Attribute	Value
Purpose	Runs the PostgreSQL 15 database instance
Functionality	Stores all persistent application data including Users, Tools, Bookings, Reviews, Payments, and Payouts
Technology	PostgreSQL 15
Port	Internal port 5432 (exposed for development; secured in production)
Volumes	Persistent data storage via Docker volume db_data
Security	In production, access is restricted to the backend container only

Reverse Proxy Container (NGINX)

Attribute	Value
Purpose	Acts as a reverse proxy and load balancer
Functionality	Routes incoming HTTP requests to the appropriate container, provides a single entry point for the application
Technology	NGINX 1.27
Port	Exposed on port 8081 (maps to internal port 80)
Configuration	Proxies requests to frontend container

Together, these containers provide a clean separation of concerns, ensure environment parity across development and production, and simplify deployment, scaling, and maintenance of the ToolShed application.

5 API for developers

[Explain the API organization and main collections in general terms. Details/documentation of methods should be included in a hosted API documentation solution, such as Swagger, Postman documentation, or included in the development itself (e.g., Maven site).
 Be sure to use best practices for REST Api design. Keep minda REST API applies a resource-oriented design (APIs should be designed around resources, which are the key entities your application exposes, not actions)

The backend API of the ToolShed system follows a RESTful architecture with a resource-oriented design. Each controller manages a specific type of resource (e.g., tools, bookings, payments, reviews), and exposes endpoints that allow standard CRUD operations using HTTP methods such as GET, POST, PUT, and DELETE.

The API is organized in a modular way, with separate controllers for each resource domain. This ensures a clean separation of concerns and improves maintainability and scalability. The main resource domains include tool management, booking lifecycle, payment processing via Stripe, user reviews, Pro Member subscriptions, and issue reporting.

All endpoints are documented using Swagger (OpenAPI 3.0 Specification), which provides a user-friendly interface for exploring and testing the API. The documentation is hosted within the application itself and is automatically generated based on annotations in the source code (using springdoc-openapi). This guarantees that the documentation remains synchronized with the implementation at all times.

The design of the API respects best practices for REST APIs, including:

- **Use of meaningful and plural nouns** for resource paths (e.g., /api/tools, /api/bookings, /api/payments)
- **HTTP methods that clearly reflect the action** being performed (GET for retrieval, POST for creation, PUT for updates, DELETE for removal)
- **Path parameters for specific resource access** (e.g., /api/tools/{toolId}, /api/bookings/{bookingId})
- **Query parameters for filtering** (e.g., /api/bookings?ownerId={id}, /api/tools/search?keyword={term})
- **Clear status codes and consistent response structures** across all endpoints
- **UUID identifiers** for all resources to ensure global uniqueness

This structure ensures the API is intuitive for developers, supports easy client integration, and is prepared for future extensions.

Swagger UI Available at: http://localhost:8080/swagger-ui/index.html

Bookings Operations related to tool bookings

- PUT `/api/bookings/{bookingId}/status` Update booking status
- GET `/api/bookings` Get bookings for owner
- POST `/api/bookings` Create a booking
- POST `/api/bookings/{bookingId}/pay-deposit` Pay deposit
- POST `/api/bookings/{bookingId}/condition-report` Submit condition report
- POST `/api/bookings/{bookingId}/cancel` Cancel booking

Tools Operations related to tool listings and search

- GET `/api/tools/{toolId}`
- PUT `/api/tools/{toolId}`
- DELETE `/api/tools/{toolId}`
- GET `/api/tools` Search active tools by keyword, district, and/or price range
- POST `/api/tools`
- POST `/api/tools/{toolId}/maintenance`
- GET `/api/tools/supplier/{supplierId}`
- GET `/api/tools/search`
- GET `/api/tools/active`

Reviews Operations related to reviews

- PUT `/api/reviews/{id}` Update a review
- POST `/api/reviews` Create a review

report-controller

- PUT `/api/reports/{id}/status`
- GET `/api/reports`
- POST `/api/reports`
- DELETE `/api/reports/{id}`

subscription-controller

- POST `/api/subscriptions/webhook`
- POST `/api/subscriptions/pro/{userId}`
- POST `/api/subscriptions/activate/{userId}`
- GET `/api/subscriptions/status/{userId}`
- DELETE `/api/subscriptions/{userId}`

payment-controller

- POST /api/payments/payout/{ownerId}
- POST /api/payments/mark-paid/{bookingId}
- POST /api/payments/mark-deposit-paid/{bookingId}
- POST /api/payments/create-deposit-checkout/{bookingId}
- POST /api/payments/create-checkout-session
- GET /api/payments/wallet/{ownerId}
- GET /api/payments/wallet/{ownerId}/earnings
- GET /api/payments/status/{bookingId}
- GET /api/payments/payouts/{ownerId}

auth-controller

- POST /api/auth/register
- POST /api/auth/login
- GET /api/auth/check-email

admin-controller

- POST /api/admin/users/{id}/deactivate
- POST /api/admin/users/{id}/activate
- GET /api/admin/users
- GET /api/admin/stats