POLISH-JAPANESE ACADEMY
OF INFORMATION TECHNOLOGY

Faculty of Information Technology

Department of Databases
Databases

Fatima Mehdizade, s28222
Khanin Yaroslav, s27237

PetMinder Exchange: A Points-Based Community Platform for Pet Sitting

Bachelor's degree thesis written
under the supervision of:

Filip Kwiatkowski, M.Eng.
Piotr Gago, M.Eng.

Warsaw, February 2026

POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Departament Baz Danych
Bazy Danych

Fatima Mehdizade, s28222
Khanin Yaroslav, s27237

PetMinder Exchange: Platforma Społecznościowa do Opieki nad Zwierzętami oparta na Punktach

Praca inżynierska napisana pod kierunkiem:

Mgr inż. Filip Kwiatkowski
Mgr inż. Piotr Gago

Warszawa, luty 2026

Abstract

This thesis addresses the challenge of finding reliable pet care within local communities. To bridge this gap, PetMinder was developed — a web-based platform built with ASP.NET Core, Blazor WebAssembly, and PostgreSQL that facilitates reciprocal pet-sitting exchanges using a points-based model inspired by time-banking[1]. By prioritizing community engagement over monetary transactions, the system provides a secure environment for pet owners and sitters to support one another through location-based discovery, real-time communication, and a transparent review system.

Keywords

Pet Sitting · Time Banking · ASP.NET Core · Blazor WebAssembly · PostgreSQL

Acknowledgements

We would like to thank our supervisor, Mr. Filip Kwiatkowski, for his guidance and feedback throughout the development of this thesis. His supervision was essential in helping us bring this project to completion.

We also extend our thanks to Mr. Piotr Gago for conducting the diploma seminar and providing the structural guidelines necessary for this documentation.

Additionally, we appreciate the support of Mr. Abdulla Mohamed, who voluntarily dedicated his time to assist us. His technical advice helped us resolve several key challenges during the implementation phase.

On a personal note, I, Fatima, would like to thank my friend Natalia for her support and for keeping my spirits high during the stressful moments of this journey. I must acknowledge my dog Maja, who provided not only emotional support but also the inspiration for building a platform dedicated to better pet care.

Finally, we thank each other for the effective collaboration and persistence required to deliver this project.

Streszczenie

Niniejsza praca dyplomowa dotyczy wyzwań związanych z dostępem do zaufanej opieki nad zwierzętami. W odpowiedzi na te potrzeby opracowano PetMinder — platformę internetową zbudowaną w technologiach ASP.NET Core, Blazor WebAssembly oraz PostgreSQL, która umożliwia wzajemną pomoc w opiece nad zwierzętami z wykorzystaniem modelu punktowego zainspirowanego ideą banków czasu[1]. Stawiając na zaangażowanie społeczności zamiast transakcji finansowych, system zapewnia bezpieczne środowisko dla właścicieli i opiekunów poprzez wyszukiwanie lokalizacyjne, komunikację w czasie rzeczywistym oraz przejrzysty system opinii.

Słowa kluczowe

Opieka nad zwierzętami · Bank czasu · ASP.NET Core · Blazor WebAssembly · PostgreSQL

Podziękowania

Chcielibyśmy serdecznie podziękować naszemu promotorowi, Panu Filipowi Kwiatkowskiemu, za jego przewodnictwo i cenne uwagi przekazane w trakcie powstawania tej pracy. Jego nadzór był kluczowy dla pomyślnego ukończenia tego projektu.

Wyrazy wdzięczności kierujemy również do Pana Piotra Gago za prowadzenie seminarium dyplomowego i dostarczenie wytycznych strukturalnych niezbędnych do przygotowania niniejszej dokumentacji.

Dodatkowo doceniamy wsparcie Pana Abdulli Mohameda, który dobrowolnie poświęcił swój czas, aby nas wesprzeć. Jego porady techniczne pomogły nam rozwiązać wiele kluczowych wyzwań w fazie implementacji.

Osobiste podziękowania chciałabym — ja, Fatima — złożyć mojej przyjaciółce Natalii za jej wsparcie i podtrzymywanie mnie na duchu w stresujących momentach tej drogi. Muszę również wspomnieć o mojej czworonożnej przyjaciółce, Mai, która była nie tylko źródłem wsparcia emocjonalnego, ale także codzienną inspiracją do stworzenia platformy dedykowanej lepszej opiece nad zwierzętami.

Na koniec dziękujemy sobie nawzajem za efektywną współpracę i wytrwałość, które były konieczne do zrealizowania tego projektu.

Table of Contents

1. Introduction and background

1.1 Problem statement

Pet owners frequently face difficulties when seeking trustworthy, flexible, and affordable pet-sitting options. Traditional platforms often rely on direct monetary transactions, which can limit accessibility for owners who prefer alternative or budget-friendly arrangements. Moreover, these monetary-based models do not fully leverage the sense of community and reciprocal support that can emerge when pet owners help each other. As a result, there is a gap for a service that fosters a peer-to-peer exchange system without centering on profit, allowing members to share resources more equitably.

At a technical level, managing such a points-based pet-sitting system introduces added complexity. The platform must maintain high standards of data integrity—tracking user profiles, handling real-time points balances, and coordinating scheduling in a way that is accurate and efficient. Failing to address these database requirements can lead to inconsistencies in points calculation, delays in matching sitters with owners, and overall user dissatisfaction. Robust data modeling, query optimization, and a reliable database architecture are therefore essential to ensure the platform can scale, handle concurrent requests, and uphold fair exchanges among its users.

In addition, trust and transparency are critical in any community-oriented platform. Without a clear method for verifying user identities, rating sitter performance, and preventing fraudulent activity, the quality and safety of pet care remain uncertain. Implementing a secure database structure that supports verifiable user interactions and feedback loops can help mitigate these risks. Consequently, there is a clear and pressing need for a well-architected, points-based pet-sitting platform that integrates effective database solutions to provide a reliable, community-focused environment for both pet owners and sitters.

1.2 Motivation

Modern pet owners often struggle with finding consistent and affordable ways to ensure their pets receive proper care. Traditional pet-sitting platforms tend to rely on monetary transactions, which can lead to a purely commercial relationship rather than a communal exchange of support. By introducing a points-based model, users are encouraged to help one another without incurring direct financial costs, thus fostering a sense of reciprocity and neighborhood-like trust. This approach not only lowers the barriers to accessing pet care but also strengthens communal ties, as members exchange services and build reputations over time[2].

Furthermore, a structured system for tracking user credibility, handling bookings, and managing points can enhance the overall experience for both pet owners and sit-

ters. Clear rules and transparent transactions reduce the risk of misunderstandings and ensure fair, consistent exchanges[3]. By incentivizing high-quality care and responsible participation, the platform can ultimately create a more reliable and supportive community—one that values mutual benefit and shared love for pets over mere profit.

1.3 Goals

Aim of the Project:
The primary aim of this project is to develop a points-based pet-sitting platform that facilitates secure, transparent, and community-driven exchanges of pet care services. By implementing a structured system for managing user profiles, scheduling, and points transactions, the platform aspires to foster reciprocal support among pet owners, reduce costs associated with traditional pet-sitting, and enhance trust and convenience in the overall pet care process.

Objectives:

1. Platform Analysis and Design:
    - Determine functional and non-functional specifications to guide system architecture.
    - Develop an optimal database schema to ensure data integrity, scalability, and robust querying.
    - Select a suitable technology stack (ASP.NET Core, Blazor, PostgreSQL) to support the platform's envisioned features.
    - Develop a Minimum Viable Product (MVP) and iterate on features based on technical requirements.
2. Development of Key Features:
    - Incorporate a mechanism for earning, spending, and tracking points that eliminates the need for direct monetary transactions.
    - Create secure registration, authentication, and role-based access controls for pet owners and sitters.
    - Enable users to request and confirm pet-sitting sessions, ensuring availability and streamlined scheduling.
    - Allow users to evaluate completed sittings via a double-blind review system, fostering accountability.
3. Improvement of Communication and Engagement:
    - Provide real-time communication channels for coordinating pet-sitting details and building trust.
    - Implement automated notifications about new requests, confirmations, and changes to scheduled sittings.

    - Encourage active participation through gamification elements such as badges, achievement levels, and referral incentives.

4. Integration and Technical Assessment:
    - Integrate the application with a serverless cloud database (Neon) to support scalable data persistence.
    - Conduct database benchmarking to assess query optimization and ensure sub-millisecond responsiveness for critical data feeds.
5. **Sustainability and Future Growth:**
    - Design mechanisms (such as point expiration and verification rewards) that promote ongoing user retention and reciprocal engagement.
    - Incorporate architectural decisions that support future horizontal scaling and feature expansion.
    - Develop administrative guidelines and automated policies to ensure the points-based model remains equitable.

1.4 Conventions

Throughout this thesis, specific formatting conventions are used to enhance readability and distinguish between different types of information:

- **Bold text** is used for key terms, user interface elements (e.g., buttons, menus), and architectural components.
- *Italicized text* is used for emphasis, book titles, and foreign words.
- `Monospace text` is used for code snippets, database tables, file names, and API endpoints.
- **Code Listings**: In code blocks, a line without a preceding line number indicates a continuation of the previous long line that has been wrapped to fit the document's width.

1.5 Results

The project resulted in the successful development and deployment of PetMinder, a fully functional Proof-of-Concept (PoC) platform. Key outcomes include:

- A robust circular economy model validated through functional testing and scenario simulations.
- A responsive Single Page Application (SPA) capable of handling real-time interactions via SignalR.
- A secure backend architecture implementing role-based access control and advanced data integrity measures.

- Optimization of critical database queries to achieve sub-millisecond execution times through advanced indexing strategies.
- Implementation of a moderation system and "Safety Center" to manage user reports and maintain community trust.
- Integration of multistep identity verification and device fingerprinting to prevent fraud and ensure platform safety.
- Deployment of a geolocation-based discovery module utilizing the Haversine formula for precise sitter matchmaking.

The system demonstrates that a non-monetary, trust-based exchange model is technically feasible and scalable.

1.6 Document structure

The remainder of this thesis is structured as follows:

- **Chapter 2: Existing Solutions** analyzes current market competitors and highlights the gap PetMinder aims to fill.
- **Chapter 3: System's Scope, Limitations, and Features** outlines the functional non-functional requirements and the technical constraints of the project.
- **Chapter 4: System Design and Architecture** details the high-level architecture, database schema, and security models.
- **Chapter 5: Implementation** discusses specific coding challenges, key algorithms, and the integration of third-party services.
- **Chapter 6: Impact and Sustainability** evaluates the social and economic models underpinning the platform.
- **Chapter 7: Conclusion** summarizes the project's achievements and suggests directions for future development.

2. Existing solutions

2.1 Overview of existing platforms

A variety of online platforms and community-driven services have emerged to connect pet owners with individuals who can provide care for their animals. These platforms range from fully commercial, profit-oriented models to informal, community-based networks. While they all facilitate the basic process of matching pet owners and sitters, each service offers unique approaches to user interaction, payment systems, communication, and overall user experience. This section provides a detailed overview of several well-known solutions, highlighting their primary operational models, technological approaches, and target user communities.

2.1.1 Petsy

Petsy is a pet-focused platform[4] that facilitates connections between pet owners and sitters or care providers. Typically accessible through both a website and a mobile application, Petsy allows users to create detailed profiles specifying their expertise, availability, and preferred types of pets. Pet owners, in turn, can search for suitable sitters based on location, schedule, and specific pet needs.

Operational Model. Petsy commonly employs a listing-based format, wherein sitters showcase their services with photos, certifications (if any), and a short biography. Pet owners can filter potential sitters by criteria such as geographic proximity, specialization in certain breeds, or experience with particular pet care tasks (e.g., administering medication, grooming). Although the service model may vary by region, the platform often integrates a secure messaging tool that allows users to discuss requirements and finalize arrangements without needing to leave the app.

Technological Features. Most Petsy implementations use standard web technologies (e.g., JavaScript-based frontend frameworks) alongside a cloud-based backend infrastructure. Basic features often include a user-friendly interface for searching and booking, user review mechanisms, and integration with third-party map services to identify local availability. Some implementations also support automated reminders, calendar management, and in-app notifications, contributing to a relatively streamlined user experience.

2.1.2 Rover

Rover is one of the most widely recognized platforms[5] for pet-sitting and dog-walking services, operating in multiple countries through a unified online presence. Although best known for dog-related services, Rover also accommodates various other pets, including cats, small mammals, and exotic animals in certain markets.

Operational Model. Users on Rover typically register as either pet owners or service providers (sitters and walkers). Owners can create a profile detailing their pet's breed, age, health conditions, and behavioral quirks. Sitters list their services—such as overnight boarding, dog walking, or drop-in visits—along with their rates and availability. Booking and payment transactions usually occur directly through Rover's platform, which also offers customer support and mediation in case of disputes or emergencies.

Technological Features. Rover's website and mobile application employ secure user account creation, integrated payment processing, and a proprietary scheduling system. Features such as GPS tracking of dog walks and photo updates are incorporated to provide transparency and peace of mind for owners. The platform's database typically stores user profiles, booking records, messages, and ratings in a structured format that supports large-scale queries, ensuring the system remains responsive even with millions of users. Rover's approach to technology focuses on reliability, security, and ease of use, aiming for a seamless experience for both owners and sitters.

2.1.3 TrustedHousesitters

TrustedHousesitters operations on a global scale[6], offering pet-sitting and housesitting arrangements for a membership fee rather than per-visit charges. This platform emphasizes an exchange of services wherein house sitters (who often care for pets as part of their stay) receive free accommodation in exchange for looking after the property and any resident animals.

Operational Model. Rather than charging a direct booking fee for each pet-sitting job, TrustedHousesitters generally employs a subscription-based model where both owners and sitters pay an annual fee to gain full access to listings. Once subscribed, users post or apply to sitting opportunities without additional charges. This structure is designed to encourage community-building and ongoing participation, as members can engage in multiple pet-sitting arrangements throughout the year.

Technological Features. The platform includes a comprehensive profile system, allowing sitters to showcase references, background checks, and personal testimonials. Owners similarly detail household responsibilities, pet care requirements, and any special instructions. Most communication occurs within the platform's messaging system, which stores conversations, sitter applications, and user feedback in a secure and centralized database. Global reach necessitates features for time-zone adjustments in availability listings, multi-currency support for membership fees, and built-in translation tools in some regions.

2.1.4 OLX / Facebook Groups

In addition to specialized pet care websites and apps, many pet owners and sitters turn to broader online marketplaces and social platforms. OLX[7], a classifieds marketplace operating in several countries, and Facebook Groups[8] are prime examples

of these general-purpose forums where users informally arrange pet-sitting opportunities.

Operational Model. Users typically create public or private listings on OLX or post requests in region-specific Facebook Groups dedicated to pets or community help. While these listings can be free to post, they usually involve a direct monetary arrangement between the owner and the sitter outside of the platform itself. In some cases, community members might offer reciprocal arrangements, but this is less structured than on dedicated pet-sitting services.

Technological Features. Since OLX and Facebook Groups are not exclusively designed for pet-sitting, they lack specialized features such as integrated scheduling tools, automated booking systems, or reputation management tailored to pet care. Instead, transactions and arrangements hinge on direct communication via private messages or comments, placing the onus on users to coordinate details, verify trust, and negotiate pricing. Although these platforms can be efficient in reaching a wide audience, they do not provide the same level of built-in support or oversight that dedicated pet-sitting services typically offer.

2.2 Strengths and weaknesses of current solutions

This section evaluates the strong points and limitations commonly observed across existing pet-sitting platforms and marketplaces. While each solution has its unique attributes and technological underpinnings, there are notable areas in which many of these services excel or fall short, particularly in terms of user experience, transaction methods, community engagement, and data management practices.

2.2.1 Strengths

- **Established User Base and Brand Recognition:** Platforms such as Rover and TrustedHousesitters have invested heavily in marketing and user acquisition, resulting in broad awareness and large user communities. This wide reach simplifies the process of finding sitters and builds initial trust among new users.
- **Streamlined Booking and Payment Systems:** Many platforms integrate secure online payments, real-time scheduling, and automated reminders. These features reduce the friction involved in coordinating availability and ensuring financial transactions are logged and confirmed. A centralized system for managing bookings and payments can significantly simplify the pet-sitting arrangement process.
- **Rating and Review Mechanisms:** In systems like Rover and Petsy, user feedback often takes the form of ratings and written reviews, enabling prospective clients to gauge a sitter's reliability, professionalism, and experience. Reviews serve as a form of reputation capital that can encourage consistent quality of service.

- **Verification and Identity Checks:** Some platforms, notably TrustedHousesitters, emphasize user verification, background checks, and references as a core part of their membership model. By vetting service providers and potential sitters, these platforms aim to bolster user confidence and minimize fraudulent activity.
- **Niche Community Engagement:** Certain platforms, particularly smaller or regional ones like Petsy, cater to specific niches (e.g., owners of certain breeds or exotic pets). This specialization can foster a tight-knit community where users share advice, resources, and support beyond the scope of basic pet-sitting arrangements.

2.2.2 Weaknesses

- **Reliance on Direct Monetary Transactions:** Many existing solutions, including large-scale platforms, rely almost exclusively on paid services, which can limit accessibility for owners with budget constraints. This monetization model can also overshadow the potential for community-building and reciprocal support, as financial considerations dominate user interactions.
- **Limited Community Reciprocity:** Although some platforms incorporate review systems or community forums, few truly emphasize reciprocal exchanges or incentivize users to help one another in a non-monetary capacity. This can result in a more transactional atmosphere rather than a collaborative one that values mutual support.
- **Varying Quality of Service Providers:** Despite verification processes or profile reviews, the overall quality of sitters can differ substantially. Profiles, ratings, and references may not always reflect real-time performance, and newly registered sitters or owners often lack robust histories to establish trust.
- **Non-Standardized Data and Record-Keeping:** In channels like Facebook Groups or OLX, communication, booking details, and reviews can be scattered across posts and personal messages. This fragmentation hinders efficient data retrieval, complicates dispute resolution, and can lead to incomplete records of previous pet-sitting arrangements.
- **Lack of Structured Incentives:** While commercial platforms incentivize sitters with direct payments, there is generally little to encourage ongoing user engagement for those willing to participate in broader community activities. Without gamification or additional community-driven rewards, user motivation can wane once immediate needs are met.

2.3 Gaps in existing solutions

The platforms and marketplaces discussed in the previous sections cover many facets of pet-sitting: from large-scale, globally active applications to local, community-driven channels. However, certain areas remain insufficiently addressed, undermining the overall effectiveness and inclusivity of existing solutions. These gaps are particularly

evident in the realms of accessibility, community reciprocity, and advanced data-driven features. The following points detail key deficiencies that can motivate the development of a more comprehensive, points-based pet-sitting platform.

- **Limited Incentives for Community Engagement:** Although many platforms offer user reviews and ratings, there is often no systematic way to reward or encourage reciprocal exchanges of care. Consequently, the user experience tends to be purely transactional, with minimal focus on fostering long-term relationships or supportive communities. A points-based model could address this gap by providing tangible rewards and recognition for participants who actively help others.
- **Financial Barriers and Inflexible Pricing:** The monetization strategies on most existing platforms can exclude users with budget constraints or those who prefer alternative compensation methods. While some platforms (e.g., TrustedHousesitters) partially mitigate this issue through membership fees, fully eliminating financial constraints remains a challenge. Approaches that do not rely solely on monetary transactions have yet to see widespread adoption.
- **Fragmented Data Management and Scalability:** General-purpose channels like Facebook Groups or OLX offer little in the way of structured data management, making it difficult to track user interactions, accumulate reputational feedback, or integrate scheduling and messaging in a seamless manner. Even specialized platforms can face scalability issues as user bases grow, risking slower response times or disorganized booking histories.
- **Trust and Transparency Mechanisms:** Although some solutions implement user verification and background checks, there remains a broader need for improved trust-building features, such as more detailed sitter/owner profiles, real-time validation of services rendered, or enhanced fraud prevention. Without these mechanisms, uncertainty persists regarding sitter qualifications and the reliability of prospective pet owners.
- **Geographic and Demographic Constraints:** Many pet-sitting platforms concentrate on urban or high-demand areas, leaving rural regions underserved. Certain demographic groups (e.g., seniors or those lacking digital fluency) may also find it difficult to navigate existing platforms. A more inclusive design could address the unique needs of users in underrepresented locations and communities.

Collectively, these gaps underscore the importance of an alternative pet-sitting model that does not revolve solely around monetary transactions. By integrating a well-defined points system, transparent data tracking, and consistent mechanisms for trust-building, a new platform has the potential to improve accessibility, foster a supportive community, and better accommodate the diverse needs of modern pet owners and caregivers.

3. System's scope, limitations, and features

3.1 Platform scope

The PetMinder Exchange platform is designed to address the needs of pet owners who seek flexible, trustworthy, and cost-effective care services for their animals while promoting a community-focused model. Unlike traditional pet-sitting services that center on monetary transactions, PetMinder Exchange leverages a points-based system to encourage reciprocal support among users. The platform spans several critical functional domains, each contributing to its overarching goal of fostering a cooperative, user-driven pet care ecosystem. Key aspects of its scope include:

- **Pet Owner and Sitter Facilitation:** The system enables both pet owners and potential sitters to interact through dedicated profiles, creating a transparent environment for matching pet care needs with available, community-verified sitters. Tools for scheduling, communication, and reviews are provided to streamline the user experience.
- **Points-Based Exchange Mechanism:** At the core of the platform's design is a points-based model where users can earn points by offering pet-sitting services and spend them when they need someone to look after their own pets. This feature aims to democratize access by reducing the reliance on direct monetary transactions, thus fostering a sense of mutual support within the community.
- **Data Management and Security:** Given the need to track points transactions, verify user identities, and store booking information securely, the platform's database and data structures play a pivotal role. The scope explicitly encompasses implementing robust relational models supplemented by any necessary non-relational data handling. Ensuring data integrity, consistency, and security is a top priority.
- **User Communication and Notifications:** PetMinder Exchange offers integrated messaging features to assist in coordinating pet-sitting arrangements. Real-time notifications on booking requests, confirmations, and changes are also within the platform's scope to ensure users remain informed and can respond promptly to time-sensitive updates.
- **Review and Rating Infrastructure:** As community trust is vital for successful peer-to-peer exchanges, the platform incorporates a review and rating system. This mechanism allows participants to evaluate pet-sitting experiences, thereby reinforcing accountability and establishing reputational benchmarks for future interactions.
- **Gamification and User Engagement:** Recognizing the importance of sustained user participation, the scope includes incorporating gamification elements such as badges or achievement levels. These features aim to keep com-

munity members motivated to offer and request pet-sitting services, thereby fostering ongoing engagement.

Within these defined boundaries, PetMinder Exchange aspires to offer a seamless and community-centric approach to pet-sitting. By catering to both practical concerns (e.g., scheduling, communication) and broader community objectives (e.g., trust building, reciprocal support), the platform seeks to fill a unique niche in the existing pet care landscape.

3.2 Technical and geographical limitations

Beyond the broad functional scope of this thesis project, it is crucial to identify potential constraints that may affect platform performance, accessibility, and overall usability. These constraints can be categorized into technical and geographical limitations, each posing unique challenges for design, deployment, and ongoing support.

3.2.1 Technical limitations

From a technical perspective, the platform is expected to operate within a resource-constrained hosting environment. This reality necessitates careful consideration of both the technology stack and the underlying application architecture, ensuring that the system remains responsive and stable under concurrent usage. As a result, the software solution will place an emphasis on open-source libraries and frameworks, which, while offering robust community support and no licensing fees, may not provide the advanced features found in some proprietary products. With finite computational resources, strategies for database design will need to prioritize efficient schema implementation, selective data retention, and optimized query performance. These measures aim to minimize infrastructure overhead while maintaining the capacity to handle routine user operations reliably. Furthermore, high availability and load-balancing mechanisms may be limited during initial deployments, meaning that any unexpected surge in user traffic could test the system's resilience. In addition, backup and recovery policies must be adapted to fit within the constraints of available storage, placing greater importance on well-planned data management to reduce the risk of information loss or extended downtime.

3.2.2 Geographical limitations

The initial rollout of the platform will focus on serving users in Poland, which reduces the need for extensive multi-language support in the short term. Nevertheless, local regulations, cultural expectations regarding pet ownership, and established norms for digital transactions will shape some aspects of the user experience. Over time, the platform may expand its reach to other countries, each with its own regulations governing liability, pet care standards, and data protection requirements. Such expansion may therefore necessitate localized adjustments to the platform interface, verification processes, and user-facing features. While language diversity might be minimal during the early stages, additional linguistic support and cultural customizations could become essential if the system is adopted in regions outside Poland. Lastly, infrastruc-

ture considerations—such as variations in internet connectivity and the availability of digital services—will remain a critical factor, as consistent online access is fundamental to providing real-time communications, scheduling features, and notifications related to pet-sitting arrangements.

3.3 Functional requirements

This section defines the core functionalities that the proposed points-based pet-sitting platform must provide. These functionalities ensure a centralized, user-friendly environment where pet owners and sitters can interact efficiently, build trust, and engage in mutually beneficial exchanges without relying on direct monetary transactions.

3.3.1 Points-based pet-sitting system

Functional Requirements:

1. **Points Allocation and Deduction:**
    - The system shall automatically credit points to a sitter's account upon the successful completion of a pet-sitting session.
    - The system shall deduct the corresponding number of points from the owner's account when a booking is confirmed, pending final completion of the service.
2. **Transaction Logging and History:**
    - The system shall maintain a comprehensive log of all points transactions for each user, including date, time, and the reason for the transaction.
    - The system shall provide users with access to their personal transaction history, enabling them to view and verify past exchanges.
3. **FIFO Point Consumption and Lot Tracking:**
    - The system shall manage points in distinct "lots" based on their source transaction and expiration date.
    - The system shall enforce a **First-In-First-Out (FIFO)** consumption model, ensuring that the points with the earliest expiration timestamps are utilized first during any transaction.
    - The system shall include an automated background scheduler to identify and invalidate expired point lots without requiring manual administrative intervention.
4. **Minimum Points Requirement for Sitters:**
    - The system shall allow sitters to set a minimum points requirement for accepting a booking request.

- If an owner offers fewer points than the sitter's minimum requirement, the system shall notify the owner about the minimum amount of points required for this particular sitter.

5. **Smart Recommendation System for Point Allocation:**
    - The system shall suggest how many points an owner should offer based on duration, pet type, pet behavior complexity, urgency, and sitter experience level.
6. **Points Expiry and Circulation:**
    - The system shall implement a controlled circulation model where points flow between users and are not infinitely generated.
    - The system may enforce an expiration policy where unused points disappear after a specific period unless spent or donated.

3.3.2 User profiles and verification

Functional Requirements:

1. **Profile Creation and Management:**
    - The system shall allow new users to register by providing essential details (e.g., name, contact information, password) and selecting a role (owner, sitter, or both).
    - The system shall allow new users to apply a referral code shared by another user. Upon successful completion of the first booking request by the referred user, the system shall automatically credit a bonus amount of points (e.g., 500 points) to the referrer's account balance.
    - The system shall enable users to update their profiles, including personal information, contact details, and role preferences.
    - The system shall allow sitters to set a minimum points requirement for bookings. Owners who do not meet this requirement shall not be able to send booking requests to that sitter.
2. **Identity and Profile Verification:**
    - The system shall enforce a mandatory verification process consisting of two steps: email confirmation and profile photo upload.
    - The system shall incentivize profile completion through the following one-time point rewards:
        - **Email Verification:** A user shall receive 50 points upon successful email confirmation during registration.
        - **Profile Photo Upload:** A user shall receive 50 points upon uploading a profile photo.

- The system shall restrict outgoing booking requests until the user has completed "Full Verification" (both email and photo). Unverified users may accumulate points (e.g., via referrals) but shall be prohibited from spending them until verification is complete.
- The system shall visually indicate the verification status on user profiles to signal trustworthiness to other community members.
- Users who complete both verification steps shall automatically receive a non-spendable "Verified" badge on their profile.
- Sitters who accumulate five or more positive reviews shall receive a non-spendable "Trusted Sitter" badge on their profile.

3. **Pet and Sitter Details:**
   - The system shall allow pet owners to create and manage one or more pet profiles, specifying breed, age, health requirements, and behavioral notes.
   - The system shall enable sitters to list relevant qualifications (e.g., veterinary experience) and specify restrictions (e.g., limited ability to care for certain animals).
4. **Fraud Prevention and Community Safety:**
   - The system shall enforce phone number and email uniqueness to prevent multiple fake accounts.
   - **Community Moderation & Reporting:** The system shall enable community-driven policing through the following mechanisms:
     - **User Reporting:** The system shall allow users to report other user profiles for suspicious activity, harassment, or policy violations.
     - To prevent abuse, a user can report a specific target user only once.
     - **Automated Flagging:** The system shall automatically flag a user account for administrative review if it accumulates 6 or more unique reports. Flagged accounts may be subject to temporary restrictions pending moderator investigation.

3.3.3 Booking and scheduling module

Functional Requirements:

1. **Availability Management:**
   - The system shall provide sitters with a calendar interface to set their available dates and times.
   - The system shall allow owners to view sitters' availability prior to initiating a booking request.
2. **Proximity-Based Sitter Discovery:**

- The system shall calculate the geographical distance between the pet owner's primary address and available sitters using the **Haversine formula**[9] to ensure accuracy on a spherical surface.
- The system shall default to sorting search results by proximity, displaying the nearest available confirmed sitters first.

3. **Booking Requests and Confirmation:**
   - The system shall enable pet owners to submit booking requests, specifying desired time slots, pet details, and any special requirements.
   - The system shall notify the sitter of a new request and require explicit acceptance or rejection for the booking to be confirmed.
   - The system shall prevent double bookings by disallowing overlapping reservations for a given sitter.
   - The system shall allow sitters to set a minimum points requirement on their profile. Owners who offer fewer points than this threshold shall have their requests automatically rejected.
   - The system shall integrate a Smart Recommendation System, which suggests the appropriate number of points an owner should offer based on factors such as duration, pet type, pet behavior complexity, urgency, sitter experience level, and any special medical or behavioral care needs.
4. **Scheduling Modifications and Cancellations:**
   - The system shall allow both owners and sitters to propose changes to confirmed bookings (e.g., updated dates, altered durations).
   - The system shall handle cancellations according to platform policy, including the possibility of partial point deductions or refunds in certain scenarios.
   - **Refund Policy Specification:** To ensure fairness for both parties, the system enforces a tiered automated refund policy based on the timing of the cancellation relative to the service start date:
     - **Standard Refund (100%):** A full refund of points is issued if the booking is cancelled more than 48 hours before the scheduled start time.
     - **Late Cancellation (50%):** If the cancellation occurs within 48 hours of the start time, the owner is refunded only 50% of the escrowed points. The remaining 50% is transferred to the sitter as compensation for the reserved time.
     - **In-Progress/No-Show (0%):** Cancellations made after the booking start time result in a 0% refund; the full point amount is transferred to the sitter.

3.3.4 Review and rating system

Functional Requirements:

1. **Post-Booking Reviews:**
    - The system shall allow both owners and sitters to submit a review after the completion of a booked service, capturing quantitative ratings (e.g., star ratings) and optional text commentary.
    - The system shall permit only users who participated in a completed booking to leave reviews for each other.
    - The system shall implement a **double rating mechanism**, where:
        - Owners rate the sitter after service completion.
        - Sitters rate the owner and pet after a pet-sitting service.
2. **Visibility and Integrity of Feedback:**
    - The system shall display a user's aggregate rating and recent reviews on their profile, assisting others in evaluating the user's reputation.
    - The system shall not permit the direct editing or removal of submitted reviews by the original author.
    - The system shall allow users to report unfair or fraudulent reviews, which can then be reviewed by platform moderators for potential removal.
3. **Aggregation of Ratings:**
    - The system shall calculate an overall rating metric based on all available feedback, providing a concise measure of user reliability.
    - The system shall update rating aggregates in real time to reflect each new review or rating.
    - The system shall use separate rating categories for sitters, owners[10], pets, and houses to provide a more detailed evaluation.

3.3.5 Gamification for user engagement

Functional Requirements:

1. **Tiered Badge System:**
    - The system shall implement a multi-tiered badge structure (**Bronze, Silver, Gold**) for all achievements, rewarding progressive user milestones (e.g., 5, 15, and 30 completed bookings).
    - The system shall automatically recalculate badge eligibility upon relevant actions (e.g., completing a booking, receiving a review) and upgrade users to higher tiers immediately.
2. **Specific Achievement Definitions:**

- **Reliable Sitter Streak:** Awarded for maintaining a sequence of completed bookings without sitter-initiated cancellations.
- **Peak-Season Helper:** Awarded for completing bookings during high-demand holiday periods (e.g., Christmas, New Year's, Thanksgiving).
- **Top Rated:** Awarded to sitters who maintain an average rating above 4.8 with a minimum of 10 verified reviews.
- **Reviewer Badge:** Awarded to users who consistently provide feedback for their peers, recognizing levels at 5, 15, and 30 submitted reviews.

3. **Referral Incentives:**
    - The system shall award bonus points to users who successfully refer new members to the platform, promoting organic community growth.
    - The system shall notify referrers when their bonus points are credited.
    - The system shall impose safeguards to prevent referral abuse, such as linking referrals to unique and verified accounts.

3.3.6 In-app messaging and notifications

Functional Requirements:

1. **Real-Time Messaging:**
    - The system shall offer an internal messaging feature, allowing owners and sitters to converse without relying on external applications.
    - The system shall store conversation histories securely to facilitate resolution of disputes or misunderstandings.
2. **Push Notifications and Alerts:**
    - The system shall send alerts to users whenever a new booking request is received, a booking status changes, or a relevant message arrives.
    - The system shall allow users to toggle individual notification types (e.g., enabling "Booking Alerts" while disabling "System Alerts") via a preferences dashboard.
    - The system shall provide "Bulk Actions" for notification management, allowing users to dismiss or delete all notifications simultaneously.

3.3.7 Platform administration module

Functional Requirements:

1. **User Oversight and Management:**
    - The system shall enable administrators to search and retrieve any user profile within the registry.

- The system shall allow administrators to view a comprehensive aggregation of a user's history, including recent transactions, bookings, submitted reviews, point balances, and chat logs.
- The system shall allow administrators to manually flag specific accounts for monitoring and modify account statuses (e.g., suspend, ban, or reactivate) based on investigation outcomes.

2. **Content and Community Moderation:**
   - The system shall aggregate all reported reviews and reported user profiles into a moderation queue for administrative review.
   - The system shall provide mechanisms to adjudicate reports by either enforcing penalties (e.g., deleting a review, banning a user) or dismissing the report as invalid.
   - The system shall ensure that the removal of a review or user does not corrupt the integrity of associated historical transaction records.
3. **Dispute Resolution and Data Access:**
   - The system shall grant administrators read access to private conversation histories and detailed transaction logs specifically for the purpose of resolving disputes or investigating safety violations.
   - The system shall allow administrators to intervene in disputes by analyzing the provided evidence (chat logs, booking details) to determine a fair outcome.
4. **Economic Governance:**
   - The system shall allow administrators to configure global economic parameters, such as the cancellation policy thresholds or point values for specific actions.
   - The system shall enable administrators to manually adjust (credit or debit) a user's point balance to rectify system errors or resolve disputes.
   - The system shall require a reason or annotation for any manual point adjustment to maintain an audit trail.

3.4 Non-functional requirements

Non-functional requirements define the quality attributes and constraints that the platform must satisfy to ensure optimal performance, security, reliability, and usability. These requirements do not describe specific functionalities but rather the standards by which the system's operations will be evaluated. Each requirement is formulated according to the SMART (Specific, Measurable, Achievable, Relevant, Time-bound) principle to facilitate clear assessment and verification.

3.4.1 Security requirements

1. **Accountability and Authenticity:**
    - The system shall log all user actions (e.g., point transactions, review submissions, and booking updates) with a unique user identifier to support auditing and traceability.
    - The system shall enforce strong password policies and multi-factor authentication (MFA) for administrative and high-privilege accounts.
2. **Confidentiality and Integrity:**
    - The system shall encrypt all sensitive user data (e.g., personal information, phone numbers) in transit (using TLS/HTTPS) and at rest (using AES-256 or an equivalent standard)[11].
    - The system shall validate and sanitize all user input to protect against unauthorized data manipulation and common attacks such as SQL injection and cross-site scripting (XSS).
3. **Threat Mitigation and Availability:**
    - The system shall implement IP throttling and rate-limiting mechanisms to restrict excessive API requests from a single source, preventing brute-force attacks and abuse.
    - The system shall utilize device fingerprinting techniques to identify and block malicious actors attempting to bypass account restrictions or create duplicate entities.

3.4.2 Maintainability requirements

1. **Modularity and Reusability:**
    - The system shall be developed following a modular design pattern, allowing components (e.g., points management, booking processes) to be independently upgraded or replaced.
    - The system shall incorporate a well-documented API structure, enabling seamless integration and future feature extensions.
2. **Analyzability and Changeability:**
    - The system shall provide clear error messages and detailed logs to facilitate quick diagnosis and resolution of issues, supporting efficient technical troubleshooting.
    - Minor system updates (e.g., user interface modifications, bug fixes) shall be deployable without requiring downtime for end-users.

4. System design and architecture

4.1 High-level architecture overview

The PetMinder Exchange platform is designed using a robust **Three-Tier Architecture**[12], which ensures a clear separation of concerns, maintainability, and scalability. This architectural pattern divides the system into three logical and physical computing tiers: the Presentation Tier (Client), the Application Tier (Server), and the Data Tier (Database).

The system operates as a **Single Page Application (SPA)** where the user interface interacts with the backend services via a stateless RESTful API and a real-time SignalR connection.

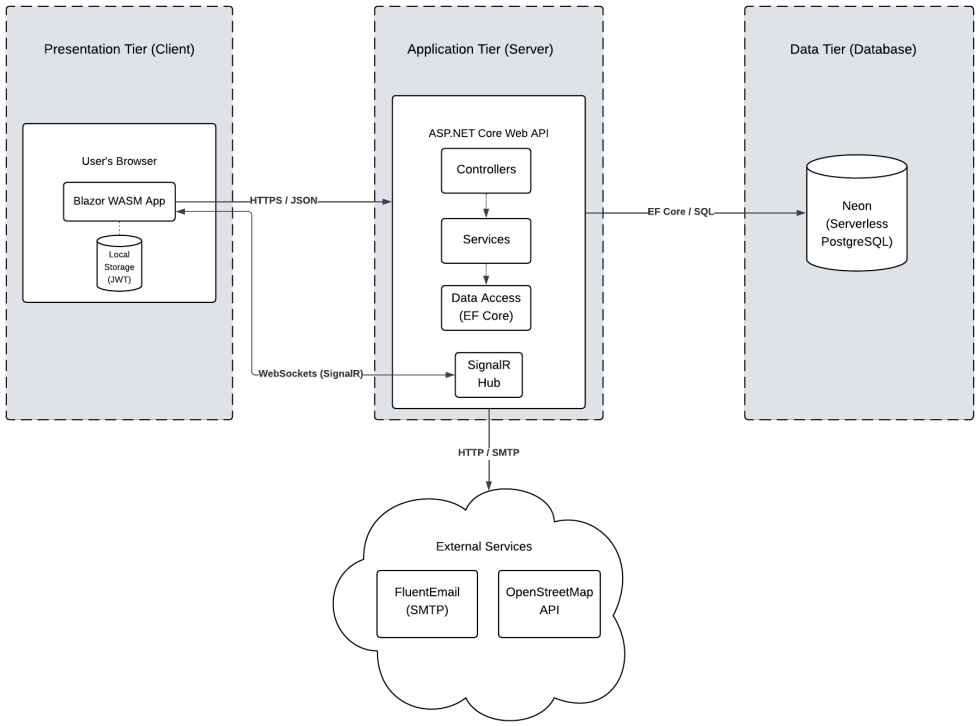

Figure 4.1: High-Level Architecture of the PetMinder Platform

4.1.1 Presentation tier (client)

The frontend is developed using **Blazor WebAssembly (WASM)**[13]. As illustrated in Figure 4.1, the application binary is downloaded to the **User's Browser** and executes client-side.

- **Blazor WASM App:** Responsible for rendering the UI and managing application state. It communicates with the server via **HTTPS/JSON** for standard data operations.
- **Local Storage (JWT):** To maintain security in a stateless environment, the application stores JSON Web Tokens (JWT) in the browser's local storage[14]. These tokens are attached to API requests to authenticate the user.

4.1.2 Application tier (server)

The backend is hosted on an **ASP.NET Core Web API**[15]. This tier handles business logic and acts as the bridge between the client and the data. The internal flow, as shown in the architecture diagram, follows a strict layered approach:

- **Controllers:** The entry point for HTTP requests. They validate Data Transfer Objects (DTOs) and route commands to the appropriate services.
- **SignalR Hub:** A dedicated component for handling real-time **WebSockets** connections, enabling instant chat messaging and live notifications.
- **Services:** This layer contains the core business logic (e.g., point calculations, booking validations). It orchestrates operations between the controllers and the data access layer.
- **Data Access (EF Core):** The lowest layer of the application tier. It utilizes **Entity Framework Core** to abstract database interactions, executing SQL commands securely.

4.1.3 Data tier (database)

The system utilizes **Neon**, a serverless provider[16] for **PostgreSQL**.

- **Serverless Architecture:** By using Neon, the database storage and compute resources scale automatically based on traffic demand.
- **Persistence:** This tier stores all relational data, including users, bookings, and transactions. It enforces data integrity through foreign keys and ACID compliant transactions.

4.1.4 External services

To fulfill specific functional requirements, the backend integrates with third-party providers via HTTP and SMTP:

- **FluentEmail (SMTP):**[17] Handles the delivery of transactional emails such as account verification links.
- **OpenStreetMap API:** Provides geospatial data for distance calculations during the sitter discovery process.

4.2 System actors and roles

The PetMinder Exchange platform serves a diverse set of human and non-human actors. The system is designed with a Role-Based Access Control (RBAC) model where specific actors interact with distinct modules of the application. The following subsections define these actors and their relationship to the system's architecture.

4.2.1 Identity and account actors

The entry point of the system involves actors responsible for account creation, authentication, and identity verification.

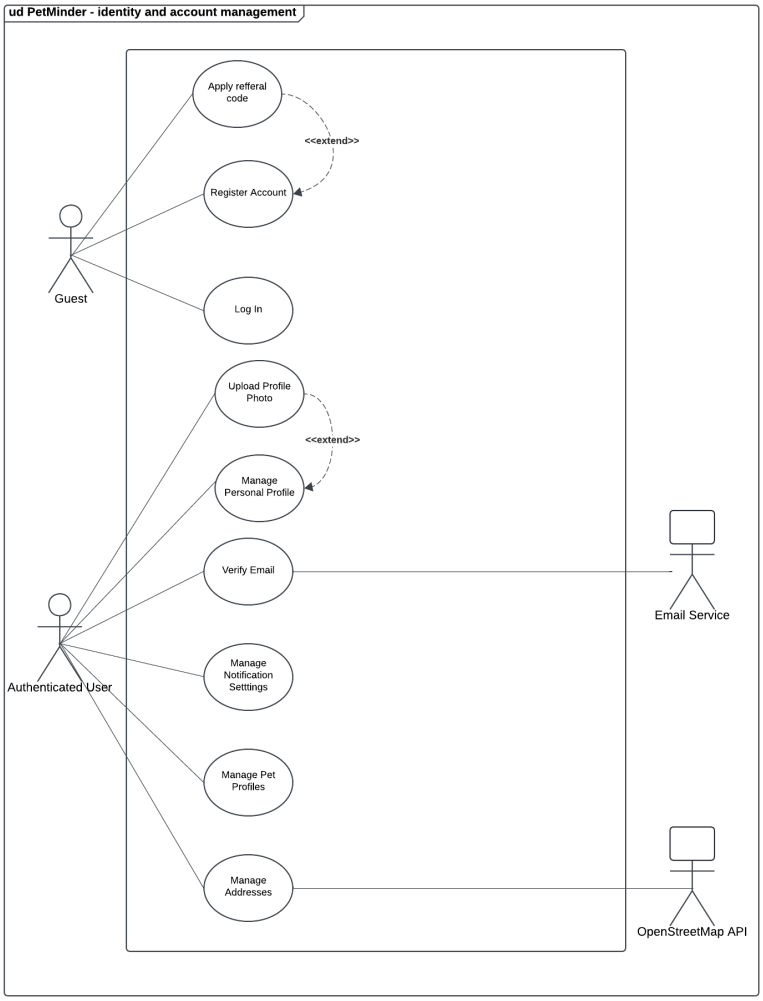

Figure 4.2: Use Case Diagram: Identity and Account Management

- **The Guest (Unauthenticated User):** Represents any visitor who has not yet logged in or registered. Their access is strictly limited to public landing pages, the registration form, and the login interface. They cannot view sitter profiles or initiate bookings.
- **Authenticated User:** Represents a user who has successfully logged in but

is performing actions common to all accounts, regardless of their specific role (Owner or Sitter). These actions include managing personal profile data, verifying email addresses, and configuring notification settings.

- **External Service Actors:** The system relies on external APIs to fulfill identity and verification requirements:
  - **Email Service:** An external system actor (via FluentEmail) responsible for dispatching emails. While the actual delivery is handled programmatically, the process is triggered by specific user actions, such as requesting a verification token.
  - **OpenStreetMap API:** An external geospatial actor used during profile management to validate user addresses and convert text locations into coordinates[18].

4.2.2 Booking participants

The core transactional functionality of the platform is driven by registered users. As illustrated in Figure 4.3, the system utilizes a generalization relationship where specific roles inherit capabilities from a common abstract actor.

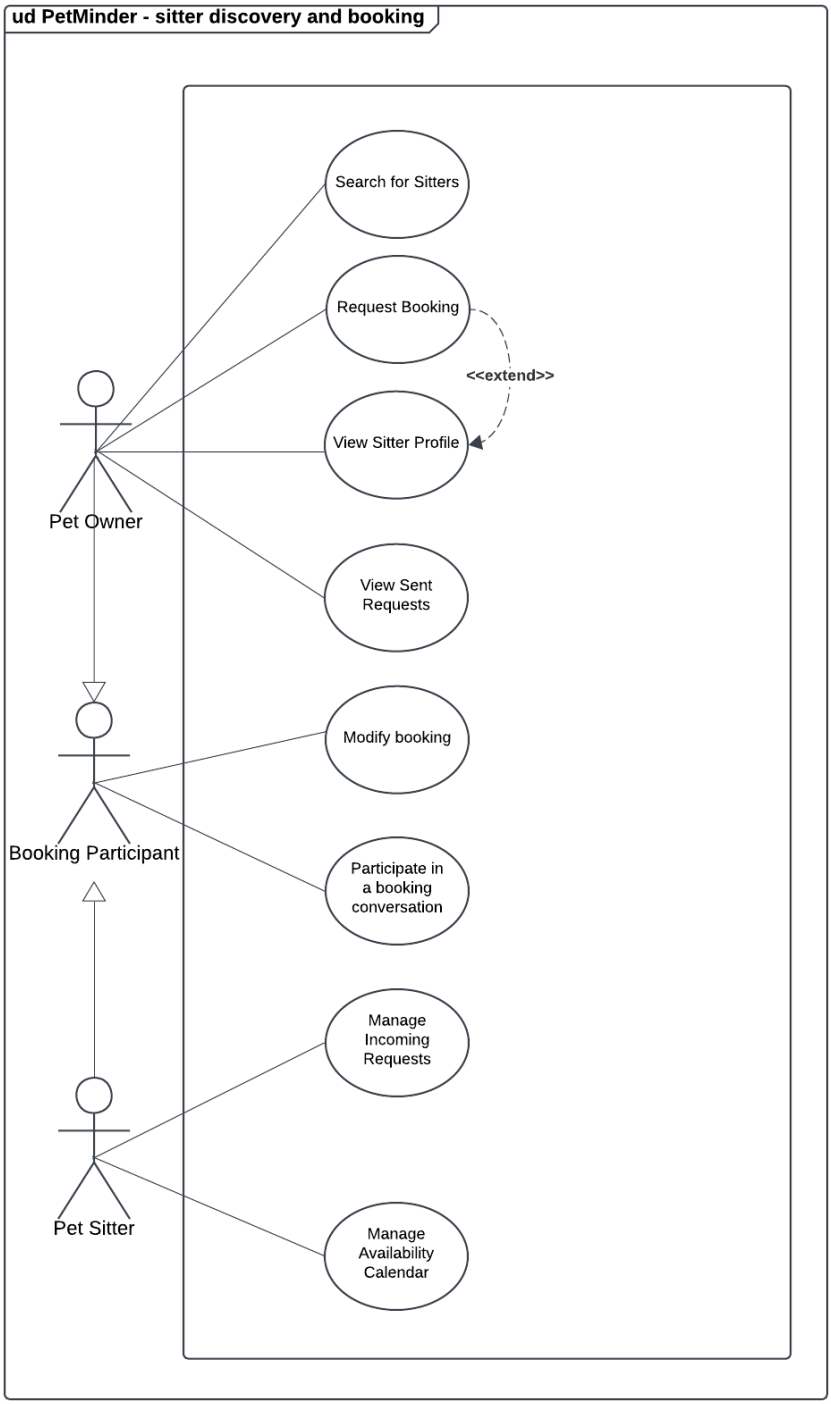

Figure 4.3: Use Case Diagram: Sitter Discovery and Booking

- **The Booking Participant (Abstract Actor):** This role represents the shared base functionality available to both parties in a transaction. As shown in the diagram, both Owners and Sitters inherit the ability to:
  - **Modify Booking:** Initiate changes to an existing reservation (e.g., date adjustments).

  - **Participate in a Booking Conversation:** Access the secure chat context linked to a specific booking.

- **Pet Owner:** A specialized role focused on the "Buyer" perspective. This actor is responsible for the discovery phase and initiates the following unique actions:
  - **Search for Sitters:** Queries the database for available service providers.
  - **View Sitter Profile:** Reviews detailed information, which can be extended into a booking request.
  - **Request Booking:** Sends a formal service proposal to a sitter.
  - **View Sent Requests:** Tracks the status of outgoing applications.
- **Pet Sitter:** A specialized role focused on the "Seller" perspective. This actor performs tasks related to service provision:
  - **Manage Incoming Requests:** Reviews pending offers to accept or decline them.
  - **Manage Availability Calendar:** Configures open slots and blocks off dates.

In addition to the booking flow, all users interact with the platform's internal economy and reputation systems. Figure 4.4 details these interactions, where the **Registered User** actor encompasses all authenticated accounts.

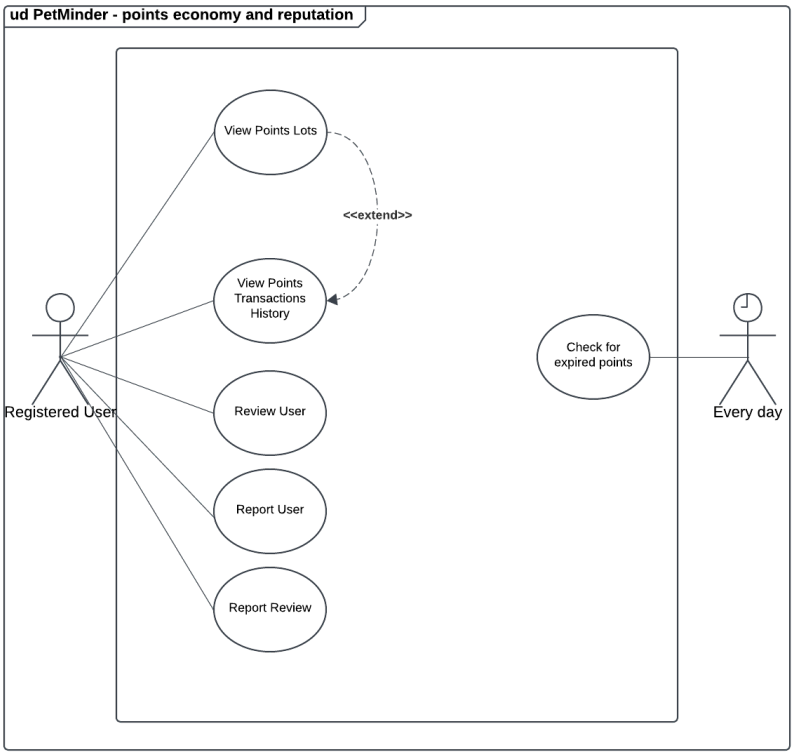

Figure 4.4: Use Case Diagram: Points Economy and Reputation

- **Registered User:** Regardless of their specific booking role, every verified user has access to the following economic and community features:
  - **View Points Lots:** Inspects the active balance and expiration dates of point bundles.
  - **View Points Transactions History:** Audits past earnings and spendings.
  - **Review User:** Submits feedback after a completed transaction.
  - **Report User / Review:** Flags suspicious accounts or content for administrative moderation.
- **Time Actor (Every Day):** A non-human actor representing the system's background scheduler. It triggers daily jobs to identify and expire unused points lots that have exceeded their 365-day validity period, ensuring the circular economy remains active.

4.2.3 Administrative actors

Governance and moderation are handled by a dedicated administrative role with elevated privileges. This actor is essential for maintaining the health of the community and the stability of the internal economy.

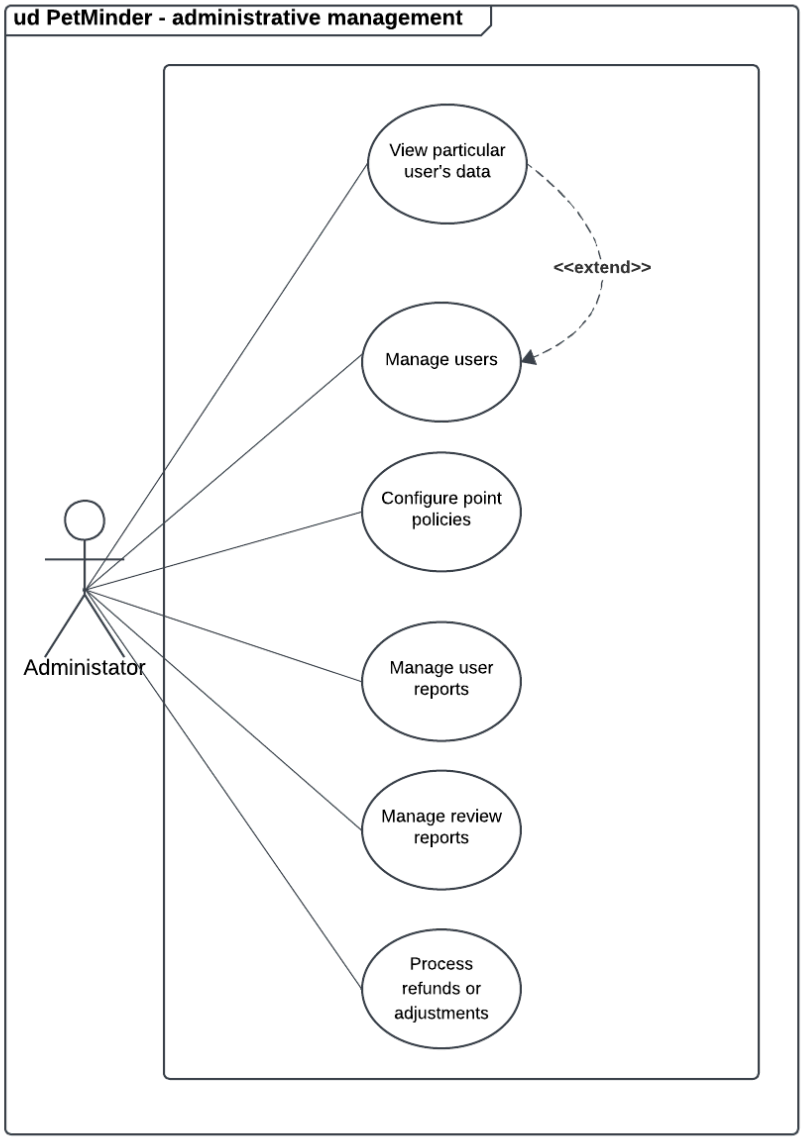

Figure 4.5: Use Case Diagram: Administrative Management

- **Administrator:** A super-user responsible for platform oversight. As illustrated in Figure 4.5, this actor does not participate in pet sitting transactions but manages the ecosystem through the following specialized actions:
  - **Manage Users:** Oversees the user base, with the ability to search for accounts and intervene if necessary.
  - **View Particular User's Data:** An extension of user management that allows the admin to inspect sensitive account details (such as transaction history or chat logs) for investigation purposes.
  - **Configure Point Policies:** Adjusts global economic parameters, such as the base exchange rate or point expiration periods, to control inflation and engagement.

- **Manage User Reports:** Moderates user reports related to other users, having the authority to dismiss false reports or ban accounts.
- **Manage Review Reports:** Moderates feedback disputes, having the authority to dismiss false reports or delete abusive reviews.
- **Process Refunds or Adjustments:** Manually overrides the points ledger to resolve financial disputes between owners and sitters.

4.3 Key use case scenarios

4.3.1 Booking initialization (pet owner perspective)

This scenario describes the primary workflow where a registered Pet Owner searches for a service provider and initiates a transaction. The process is designed to ensure that a booking is only created if strict availability and financial constraints are met.

uc PetMinder - booking initialization

Actor: Pet Owner

Purpose and context: A pet owner wants to book a sitter to take care of their pet for a specific time period.

Preconditions:
- The actor has a verified email address and a profile photo.
- The actor has added at least one pet to their profile.
- The actor has added at least a primary address to their profile.

Initiating Business Event (Trigger):
- The actor clicks on "Find Sitters" tab.

Basic flow of events:
1. The system displays "Find Sitters" page with input fields such as desired date ranges, location selection, and a pet selection.
2. The actor selects dates, location and a pet.
3. The actor clicks on "Find Sitters" button on the searching page.
4. The system displays available sorted list of sitters along with their distance from our selected location
5. The actor selects a desired sitter and clicks on their profile.
6. The system displays the sitter's detailed profile including contact details, badges, qualifications and restrictions, availability calendar, reviews, hourly minimum points requirement.
7. The actor clicks on "Book Now" button.
8. The system displays the booking form along with recommended points to spend for a given desired booking request.
9. The actor selects a desired time slot from the availability calendar, which pet needs sitting and enters the amount of points they want to offer.
10. The actor clicks the **"Send Request"** button.
11. The system validates the request:
    - Validates that the end time is after the start time.
    - Validates that the start time is not in the past.
    - Validates the offered points meet the minimum requirements.
    - Validates the offered points are not lower than system's minimum spendable point policy.
    - Validates that offered points are not suspiciously lower for a given amount of hours in case if sitter's required points settings are too low.
    - Verifies the sitter is available and has no overlapping bookings for the requested slot.
    - Verifies that actor has enough funds.
12. The system creates the booking request with status "Pending".
13. The system creates a conversation thread for the booking.
14. The system sends a notification to the sitter about the new booking request.
15. The system redirects the actor to "Sent Requests".

Alternative flow of events:
- **Booking start time is after the end time:**

11a1. The system displays an error: "Booking end time must be after the start time."
11a2. Back to step 9.
- **Booking start time is in the past:**

11b1. The system displays an error: "Booking start time cannot be in the past."
11b2. Back to step 9.
- **Offered points are lower than minimum spendable point policy:**

11c1. The system displays an error: "The offered points are too low. The system minimum per transaction is [X] points."
11c2. Back to step 9.
- **Offered points are lower than sitter's minimum points requirement:**

11d1. The system displays error: "Offered points are lower than sitter's minimum requirement of [X] points."
11d2. Back to step 9.
- **Offered points are suspiciously low:**

11e1. The system displays error: "The offered points are suspiciously low for [X] hours. Based on global points, it should be closer to [X]"
11e2. Back to step 9.
- **Sitter no longer is available:**

11f1. The system displays error: "The sitter is not available for the given time slot."
11f2. Back to step 9.
- **Sitter has another overlapping booking:**

11g1. System displays error: "The requested time slot overlaps with an existing booking."
11g2. Back to step 9.

Post-condition:
- The booking request is created with status "Pending".
- The points required for the booking are held by the system(but not yet transferred).
- A conversation thread exists for owner-sitter communication.
- The sitter is notified and can accept or reject the request.
- Both parties (owner/sitter) can propose changes to the request.

Figure 4.6: Use Case Scenario: Booking Initialization

As detailed in Figure 4.6, the flow begins with the discovery phase, where the system filters sitters based on the owner's location and required dates.

A key component of this scenario is the validation step (Step 11), which acts as a gatekeeper before any data is persisted. The backend enforces the following business rules:

- **Overlap Check:** The system verifies that the sitter has no confirmed bookings that conflict with the requested time slot.
- **Solvency Check:** The system ensures the owner has a sufficient point balance to cover the offer.
- **Policy Compliance:** The offered points must meet both the system's global minimum (to prevent spam) and the sitter's specific minimum rate.

Outcome: Upon passing validation, the system transitions the request to a `Pending` state. Crucially, the required points are deducted from the owner's balance and placed in an escrow-like "held" state. This prevents double-spending while the request awaits the sitter's approval.

4.3.2 Request management and negotiation (pet sitter perspective)

This scenario details how a Pet Sitter processes incoming work, enforcing the rule that points are only transferred when a booking is explicitly accepted. It also covers the negotiation loop, allowing dates to be adjusted even after the initial agreement.

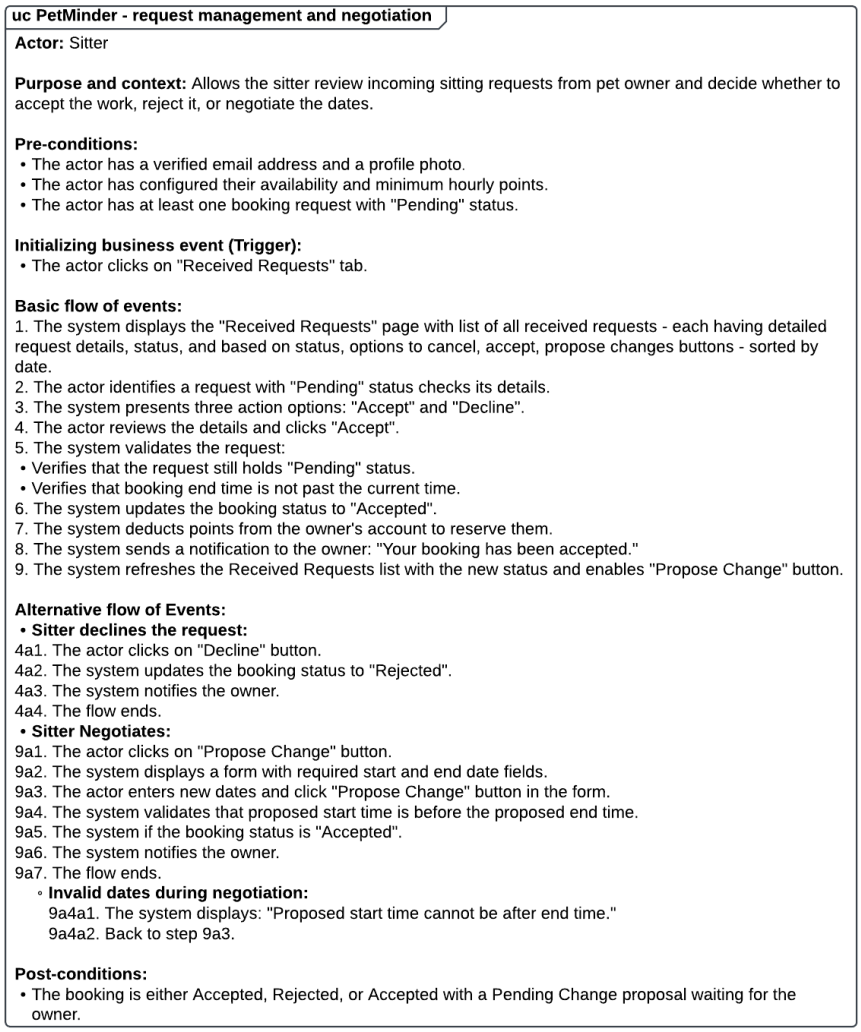

Figure 4.7: Use Case Scenario: Request Management and Negotiation

As shown in Figure 4.7, the workflow distinguishes between two primary phases:

1. Response Phase (Accept/Decline): Upon selecting a pending request, the sitter must make a binary decision.

- **Acceptance Logic:** The system first performs a "Time Check" (ensuring the booking date has not already passed) and a "Status Check" (ensuring the Owner has not cancelled). If valid, the status updates to `Accepted`, and points are reserved from the Owner's wallet.
- **Rejection:** If declined, the status moves to `Rejected`, and no points are exchanged.

2. Negotiation Phase (Post-Acceptance): The system recognizes that schedules are dynamic. Once a booking is `Accepted`, the "Propose Change" feature becomes active (Step 9).

- **Change Logic:** The sitter can suggest new dates. The system validates that

`Start Time < End Time` and that the new dates do not overlap with other confirmed jobs.

- **State Transition:** A successful proposal does not overwrite the booking immediately; instead, it flags the booking with a `Pending Change` proposal, triggering a notification for the Owner to approve or reject the modification.

4.4 Database schema design and normalization

The data persistence layer is implemented using **PostgreSQL**. The logical schema is designed to enforce strict referential integrity while optimizing for the high-concurrency requirements of a booking system. The database follows a relational model centered around the user identity and their associated transactional history.

4.4.1 Entity relationship diagram (ERD)

Figure 4.8 presents the complete physical data model. Due to the complexity of the relationships, the diagram illustrates the schema in detail, including data types, primary keys (PK), and foreign keys (FK).

Note: Enumerated types (integer status codes for bookings, address types, etc.) are annotated directly on the diagram for reference.

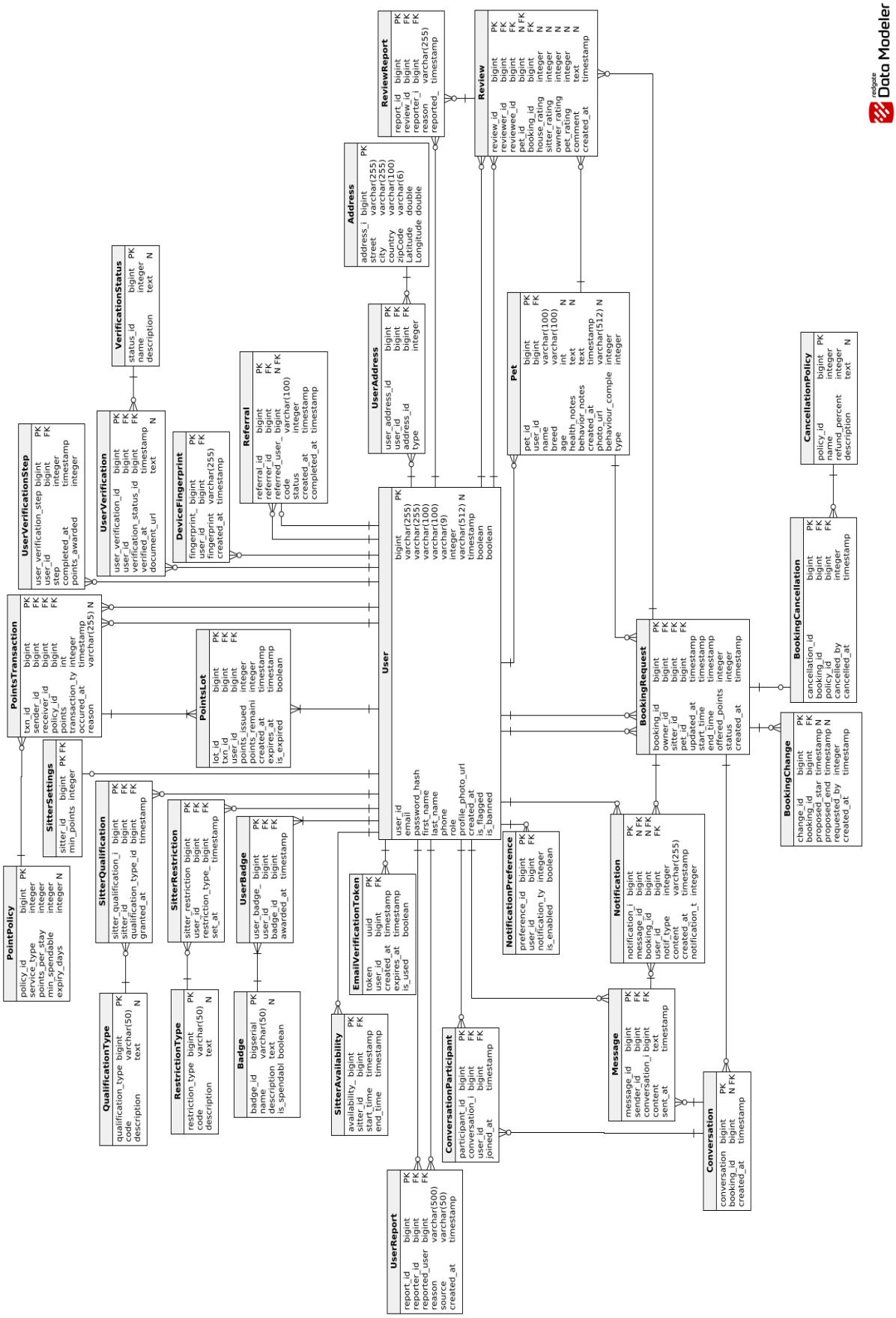

Figure 4.8: Entity Relationship Diagram (ERD) of PetMinder

4.4.2 Data dictionary and enumerations

To maintain a clean schema design, several entities utilize integer codes to represent enums. The specific allowed values for these fields are defined below.

Table 4.1: Data Dictionary: Enumerated Types and Status Codes

Entity	Attribute	Description
PointsTransaction	`txn_type`	{booking, donation, adjustment}
PointPolicy	`service_type`	{pet}
VerificationStatus	`name`	{unverified, basic, verified}
UserAddress	`type`	{primary, secondary, other}
BookingRequest	`status`	{pending, accepted, rejected, cancelled, completed}
BookingCancellation	`cancelled_by`	{owner, sitter}
CancellationPolicy	`name`	{standard, late}
BookingChange	`requested_by`	{owner, sitter}
Notification	`notif_type`	{bookingRequest, bookingAccepted, bookingRejected, bookingCancelled, bookingCompleted, changeRequested, changeAccepted, changeRejected, systemAlert}

4.4.3 Normalization and design decisions

The schema adheres to **Third Normal Form (3NF)**[19] to eliminate data redundancy and ensure that all non-key attributes are fully dependent on the primary key. Key design decisions include:

- **Normalization of Locations (UserAddress):** Rather than storing address fields (City, Street, Zip) directly on the `User` table, location data is normalized into a separate `Address` entity. A many-to-many join table (`UserAddress`) links them. This allows the system to support users with multiple locations (e.g., Primary Home vs. Vacation Home) without duplicating address strings or creating null columns.
- **Immutable Ledger (PointsTransaction):** The internal economy is modeled as a double-entry ledger system. The `PointsTransaction` table records every economic event (earnings, spendings) as an immutable row. This ensures a complete audit trail, preventing data inconsistencies that could arise from simply updating a mutable "Current Balance" column.
- **Handling Enumerations:** To optimize storage efficiency and query performance, status fields (such as `BookingRequest.status` or `Review.rating`) are stored as integers rather than strings. The application layer handles the mapping of these integers to business logic (e.g., 0 = Pending, 1 = Accepted), as detailed in Table 4.1.

- **Separation of Negotiation State:** The `BookingChange` entity is separated from the main `BookingRequest` table. This normalization ensures that temporary negotiation data (e.g., a proposed date change that hasn't been accepted yet) does not corrupt the state of the active booking.
- **Vertical Partitioning of Roles (SitterSettings):** The system distinguishes between generic user data (stored in `User`) and role-specific configuration. Attributes strictly related to service provision—such as `min_points` and `policy_id`—are isolated in the `SitterSettings` table. This ensures that the main `User` table remains lightweight and avoids storing redundant NULL values for users who act solely as Pet Owners.

5. Implementation

5.1 Technology stack selection

The PetMinder platform is built on the **.NET 8** ecosystem, enabling tight integration between frontend and backend components while preserving modularity. The frontend is implemented using **Blazor WebAssembly**, while backend services are exposed through an **ASP.NET Core Web API**. Data management is handled by **PostgreSQL**, with **Entity Framework Core (EF Core)** acting as an Object-Relational Mapping (ORM) middleware component. This modular approach ensures that the system remains scalable and maintainable.

Logic elements such as **Data Transfer Objects (DTOs)** and **enumerations** are shared between the client and server, eliminating redundant code and ensuring strict type safety across application layers.

5.1.1 Backend technologies

- **ASP.NET Core 8 Web API:**[15] Chosen for its high performance, cross-platform compatibility, and long-term support, the API exposes endpoints responsible for request routing, authorization, model validation, and data negotiation.
- **Entity Framework Core 9:** Serving as the ORM layer, EF Core allows database interaction through C# objects, reducing query complexity and simplifying schema maintenance.

Additional backend components include:

- **FluentEmail:** This library is used for email delivery, like account verification and system notifications.
- **SignalR Hubs:** Server-side SignalR hubs manage WebSocket connections, broadcast messages to connected clients, and track online user presence within the chat module.
- **Local File System (Server-Side):** User-uploaded content, such as profile pictures and pet photos, is stored securely on the server's local file system within the wwwroot directory. This approach keeps data control within the hosting environment without relying on external cloud storage services.
- **OpenStreetMap (Nominatim API):** Used for geocoding user addresses into latitude and longitude coordinates. This enables the "Proximity-Based Discovery" feature by allowing the backend to calculate precise distances between owners and sitters using the Haversine formula.

5.1.2 Frontend frameworks

- **C# and Blazor WebAssembly:** By selecting Blazor WASM, the project maintains a unified C# codebase across frontend and backend layers. This enables reuse of common models and validation logic, reducing development effort and improving long-term maintainability.
- **Tailwind CSS:**[20] Instead of traditional custom CSS, Tailwind CSS is used to define UI behavior directly within utility classes. This approach removes the need for separate stylesheets and promotes consistency and rapid UI development.

Additional frontend components include:

- **Blazor.Heroicons:**[21] This library provides scalable SVG icons used throughout the user interface to enhance visual clarity and user experience.
- **SignalR Client:** To support real-time interaction, the SignalR client enables instant updates for chat messages, eliminating the need for manual page refreshes.
- **Blazored.LocalStorage:**[14] This library enables access to the browser's local storage API, which is primarily used to securely persist JWT authentication tokens, allowing users to remain logged in across sessions.

5.2 Key development phases

The development of PetMinder was an evolutionary journey, moving from a skeletal proof-of-concept to a feature-rich platform. This process was guided by an iterative methodology, enabling a pivot based on technical challenges and evolving requirements.

5.2.1 MVP development

The initial phase, or the Minimum Viable Product (MVP), focused on establishing the core "circular economy" of pet sitting services. The primary goal was to build a stable foundation where a user could register, list their pets, and successfully request a booking from another user.

During this stage, priorities included:

- **Identity and Onboarding**: Implementing the initial authentication flow and basic profile management.
- **Discovery and Booking**: Developing the first version of the "Find Sitters" search and the state machine for booking requests (Pending $\rightarrow$ Accepted/Rejected).
- **Data Persistence**: Designing the base relational schema to handle the most critical entities (Users, Pets, Bookings).

This phase was about proving that the technical stack (Blazor and ASP.NET Core) could handle the fundamental logic of the marketplace.

5.2.2 Feature iteration

Once the foundation was solid, the project moved into a series of "sprints" to add depth and engagement to the platform. This wasn't just about adding more buttons; it was about solving friction points identified during early testing.

Key iterations included:

- **Gamification and Trust**: A *Badging System* was introduced to reward reliable sitters and the *Referral System* to encourage community growth[22].
- **Intelligent Assistance**: To help users decide on fair pricing, a *Smart Recommendation Service* was developed, which suggests point allocations based on pet complexity and sitter experience.

Listing 1: Smart Recommendation Logic

```
public int CalculateRecommendedPoints(TimeSpan duration, PetType petType,
    PetBehaviorComplexity complexity, bool isUrgent, double sitterRating, int
    sitterMinPoints, int policyBaseHourlyRate)
       {
           double baseHourlyRate = (double)policyBaseHourlyRate;
           double durationHours = duration.TotalHours;
           double basePoints = durationHours * sitterMinPoints;
           if (basePoints < 10) basePoints = 10;

           double complexityMultiplier = complexity switch
           {
               PetBehaviorComplexity.Low => 1.0,
               PetBehaviorComplexity.Moderate => 1.2,
               PetBehaviorComplexity.High => 1.5,
               PetBehaviorComplexity.Extreme => 2.0,
               _ => 1.0
           };

           double typeMultiplier = petType switch
           {
               PetType.Dog => 1.0,
               PetType.Cat => 0.9,
               PetType.SmallAnimal => 0.7,
               PetType.Bird => 0.8,
               PetType.Reptile => 1.1,
               _ => 1.0
           };

           double urgencyMultiplier = isUrgent ? 1.25 : 1.0;

           double calculated = basePoints * complexityMultiplier *
                typeMultiplier * urgencyMultiplier;

           if (sitterRating >= 4.8) calculated += 10;

           else if (sitterRating >= 4.5) calculated += 5;

           int recommended = (int)Math.Round(calculated);

           int sitterMinimumTotal = (int)Math.Ceiling(durationHours *
                sitterMinPoints);

           return Math.Max(Math.Max(recommended, sitterMinimumTotal), 0);
       }
```

- **Real-Time Connectivity**: Realizing that pet sitting requires constant communication, *SignalR* was integrated for instant chat messaging[23] and live notifications.

Listing 2: SignalR Hub Implementation

```
public async Task SendMessageToConversation(string conversationId, MessageDTO
    message)
{
    // Broadcast message to all clients in the conversation group
    await Clients.Group(conversationId).SendAsync("ReceiveMessage", message);
}
public async Task JoinConversation(string conversationId)
{
    await Groups.AddToGroupAsync(Context.ConnectionId, conversationId);
}
```

- **Frontend Overhaul**: Following the successful validation of the proof-of-concept, a complete redesign of the user interface was executed. This iteration moved beyond basic functionality to focus on a premium aesthetic and an intuitive user experience, ensuring the visual identity of the platform matched its technical sophistication.

  An advanced database indexing strategy was also implemented to ensure the application remains fast even as its user count scales (detailed in Section 5.3).

5.2.3 Deployment strategy

For a modern web application like PetMinder, the deployment strategy needed to be as agile as the development process. A hybrid cloud approach that emphasizes low latency and high availability was chosen.

The production infrastructure consists of:

- **Database Layer**: **Neon**, a serverless PostgreSQL provider[16] hosted on Azure, was utilized. This choice allows the database to scale automatically based on traffic while providing the advanced performance features (like B-Tree indexing[24]) relied upon.
- **Application Hosting**: The ASP.NET Core API and the Blazor WebAssembly frontend are designed for containerized deployment (e.g., Azure App Service or Docker). This ensures that the environment is identical between development and production.
- **Security in Transit**: All communication is strictly handled over **HTTPS**, with JWT tokens serving as the secure key for every exchange.

Environment-specific configurations are managed via `appsettings.json` files, allowing for a seamless switch between local development settings and production-grade cloud services without modifying the core logic.

5.3 Database implementation and query optimization

To ensure both data integrity and high-performance retrieval, a comprehensive indexing strategy was implemented using Entity Framework Core Fluent API. The indexes are categorized into three primary functional groups: data integrity enforcement, temporal query optimization, and composite range filtering.

5.3.1 Schema implementation

The database schema was defined using EF Core's `OnModelCreating` method within the `DbContext` class. This approach allows for a code-first design, where the C# entity models directly map to database tables. Each entity's properties are configured with appropriate data types, constraints, and relationships. Beyond simple primary keys, **Unique Composite Indexes** were utilized to enforce complex business logic directly at the database level. This prevents the "double-entry" problem and ensures the logical consistency of the marketplace.

- **Identity Uniqueness:** Single-column unique indexes on `Email` and `Phone` in the `User` table prevent duplicate registrations[25].
- **Relationship Constraints:** Composite unique indexes ensure that a sitter cannot possess duplicate `Qualifications` or `Restrictions`, and a user cannot be added to the same `Conversation` or `Address` record multiple times.
- **Action Limits:** To maintain the integrity of the feedback and reporting system, unique indexes on the `Review` and `ReviewReport` tables prevent users from submitting multiple reviews for the same booking or reporting the same review twice.

Listing 3: Implementation of Integrity and Unique Indexes

```
// Enforcing business rule: One review per booking
modelBuilder.Entity<Review>()
.HasIndex(r => new { r.ReviewerId, r.BookingId })
.IsUnique();
// Preventing duplicate verification steps
modelBuilder.Entity<UserVerificationStep>()
.HasIndex(uvs => new { uvs.UserId, uvs.Step })
.IsUnique();
```

5.3.2 Indexing for performance

For entities that serve as "feeds" (Messages, Notifications, and Transactions), the database must frequently retrieve the most recent records. By default, B-Tree indexes sort data in ascending order. To optimize "Newest First" queries, **Directional/Descending Indexes** were implemented.

By defining indexes with a `Descending` order on timestamp columns (e.g., `CreatedAt`, `SentAt`), the database engine can perform a backward scan or simply read the index from the end, eliminating the need for an in-memory `SORT` operation.

Listing 4: Descending Index for Message History

```
modelBuilder.Entity<Message>()
.HasIndex(m => new { m.ConversationId, m.SentAt })
.IsDescending(false, true); // Ascending ID, Descending Timestamp
```

Complex search scenarios, such as sitter availability and point balance calculations, require filtering across multiple dimensions simultaneously.

- **Availability Filtering:** To optimize sitter lookup, a composite index was implemented on `SitterAvailability(SitterId, StartTime, EndTime)`, allowing the system to instantly verify free slots within a specific time range.
- **Covering Index for Points:** The `PointsLot` index (`UserId, IsExpired, PointsRemaining, ExpiresAt`) contains all the data required to calculate a user's spendable balance. Since all filtered and selected columns are present in the index, the database can return the result without accessing the main table heap (a "Covering Index" scan).

5.3.3 Query testing and benchmarking results

To validate the indexing strategy, the database was populated with high-volume datasets using the **Bogus** library[26] for .NET. By generating up to 70,000 records per table with randomized but semantically valid data, the benchmarking environment was able to trigger the PostgreSQL cost-based optimizer into selecting advanced execution plans. The results demonstrated a consistent shift from linear complexity ($O(N)$) to logarithmic complexity ($O(\log N)$).

The most dramatic performance improvements were observed in entities requiring "Newest First" sorting, such as the `Messages` and `Notifications` tables.

In the `Messages` table, an unoptimized query for a specific conversation history required a full scan of 70,000 rows, followed by an in-memory *quicksort* operation. This resulted in an execution time of **10.031 ms**. After implementing a composite index on `(ConversationId, SentAt DESC)`, the database engine transitioned to an **Index Scan Backward**. Because the index is physically stored in the required sort order, the engine completely bypassed the sorting phase, retrieving the data in just **0.038 ms**—a performance gain of over 260x.

Similarly, for the `Notifications` table, the engine moved from a sequential scan of 30,000 rows to a targeted **Index Scan**. By isolating a user's alerts directly within the B-Tree structure, execution time was reduced from **2.290 ms** to **0.054 ms**, ensuring that user "feeds" remain responsive even as the system scales.

The `PointsLot` table benchmarking provided empirical evidence for the efficiency of **Covering Indexes**. A standard query to calculate a user's spendable balance previously required a sequential scan, taking **2.407 ms**.

By implementing a composite index that includes all columns used in the filter and selection (`UserId, IsExpired, PointsRemaining, ExpiresAt`), the database was able to perform an **Index Only Scan**[27]. In this plan, the engine retrieves all

necessary data directly from the index B-Tree without ever accessing the main table "heap." This eliminated nearly all disk I/O for the operation, reducing the execution time to **0.050 ms**.

Complex marketplace scenarios, such as finding sitter availability within specific time windows or filtering booking requests by status, were optimized through composite range indexing.

For `SitterAvailabilities`, which often involves overlapping timestamp checks, the engine moved from a costly sequential scan of 110,000 rows (**5.059 ms**) to a **Bitmap Index Scan**. This plan allows the database to create a temporary memory map of rows matching the time-range criteria before fetching them, resulting in a more efficient retrieval time of **0.632 ms**.

A similar transition occurred in the `BookingRequests` table. By indexing the combination of `SitterId`, `Status`, and `StartTime`, the engine reduced the time required to populate a sitter's dashboard from **1.183 ms** to **0.069 ms**, even with 20,000 active requests in the system.

While PetMinder does not feature a public "search by email" function, the lookup of user credentials by email is the most critical step in the **Authentication flow**. Benchmarking with a dataset of 51,000 users showed that a sequential scan during the login process took **2.471 ms**.

By utilizing a unique B-Tree index on the `Email` column, the database can locate the exact user record in just **0.048 ms**. Beyond the speed increase, this index serves as the final layer of data integrity, preventing the accidental creation of duplicate accounts.

Summary of Technical Impact The removal of sequential scans in favor of index-driven lookups ensures that the platform's latency is decoupled from its data growth. By utilizing advanced patterns—specifically directional sorting to eliminate heapsorts and covering indexes to avoid heap fetches—the system maintains sub-millisecond response times for all core operations. Furthermore, the conversion of Enums (such as `UserRole`) to integers ensures that these index structures remain compact, maximizing the hit ratio of the database buffer cache.

5.4 Backend features

The backend of PetMinder acts as the main operational component of the platform, responsible for orchestrating complex business processes while ensuring that all user interactions are secure, fast, and reliable. The backend has a modern, service-oriented architecture and is based on the ASP.NET Core framework. This approach combines the convenience of a project-wide monolith with the modular structure required for future expansion and modification.

5.4.1 API endpoints and microservices

PetMinder was created with a microservice-first approach, even though it is now built as a modular monolith. Every significant functional component of the program, in-

cluding real-time notifications, booking orchestration, and user management, exists within its own logical service layer. This separation of concerns improves maintainability and allows each module to be refactored in the future without affecting the rest of the system.

Communication between the frontend and backend is handled by a set of RESTful API endpoints. Controllers such as the `BookingsController` and `PetsController` act as entry points that validate incoming requests before passing execution to the desired services. To support development and testing, Swagger (OpenAPI) has been integrated into the system, providing an interactive documentation tool that exposes all available endpoints, required parameters, and expected response formats.

For features that require immediate feedback, such as chat messaging, SignalR is used to enable real-time communication. This allows the backend to push updates directly to connected clients, avoiding the standard request–response process and creating a more responsive and interactive user experience.

5.4.2 Authentication and authorization

Security was treated very seriously for PetMinder. It is a vital part that affects every request. To handle user identity, a secure system based on JSON Web Tokens (JWT) was created.

The backend issues a cryptographically signed token after confirming the user's login information. This token travels with every new request from the client, allowing the server to identify the user without repeatedly calling the database. This approach is critical for the application's general performance.

The foundation of authorization logic is a role-based, granular system:

- **Identity Verification:** Before gaining access to essential features, new users must go through an email verification process, that was made easy by FluentEmail. This guarantees the integrity of the user base.
- **Role Enforcement:** The system differentiates between Owners, Sitters, and Admins. For example, a sitter can update their availability, but only an admin can flag or ban users. ASP.NET Core policy-based authorization is used to enforce these permissions at the API level.
- **Rate Limiting:** To protect against brute-force attacks or accidental loops in the frontend, a rate-limiting middleware was implemented. This system tracks requests per IP address and specific endpoints, such as registration, temporarily throttling clients that exceed safe limits.

By integrating these layers, the backend actively protects users and the data entrusted to the system and functions efficiently.

5.4.3 Specialized business logic: administration and safety

Beyond standard CRUD operations, the backend implements sophisticated business logic to handle financial transactions and community moderation. To adhere to the

Single Responsibility Principle, these high-privilege operations are decoupled into dedicated service layers: `AdminService` and `ReportService`.

Transactional Integrity in Refunds One of the most critical backend functions is the dispute resolution system. Processing a refund is not a simple database update; it requires moving points from one user's ledger to another's while ensuring that points are not lost or duplicated if a system error occurs.

The `AdminService` utilizes **ACID-compliant database transactions**. As shown in Listing 5.5, the refund operation is atomic: the deduction from the sitter and the credit to the owner must both succeed, or the entire operation is rolled back.

Listing 5: Atomic Transaction Logic for Point Refunds

```
public async Task<bool> ProcessRefundAsync(AdminTransactionAdjustmentDTO dto)
{
    // Use EF Core Transaction to ensure atomicity
    using var transaction = await _context.Database.BeginTransactionAsync();
    try
    {
        // 1. Deduct points from Sitter (Receiver)
        await _pointsService.DeductPointsAsync(
            senderId: originalTransaction.ReceiverId,
            receiverId: originalTransaction.SenderId,
            points: dto.RefundAmount,
            type: TransactionType.Adjustment,
            reason: $"REFUND DEBIT: {dto.Reason}"
        );

        // 2. Credit points back to Owner (Sender)
        await _pointsService.CreditPointsAsync(
            receiverId: originalTransaction.SenderId,
            senderId: originalTransaction.ReceiverId,
            points: dto.RefundAmount,
            type: TransactionType.Adjustment,
            reason: $"REFUND CREDIT: {dto.Reason}"
        );

        // 3. Commit only if both succeed
        await transaction.CommitAsync();
        return true;
    }
    catch
    {
        // Revert all changes on error
        await transaction.RollbackAsync();
        throw;
    }
}
```

Automated Moderation Logic To manage community safety at scale, the `ReportService` implements an automated flagging system. Instead of relying solely on manual review, the system aggregates reports against users in real-time.

Listing 5.6 demonstrates the logic where a user account is automatically flagged for restriction once a specific threshold of unique reports is reached. This separates the concern of "Reporting" from "Authentication," keeping the architecture modular.

Listing 6: Automated User Flagging Logic

```
public async Task ReportUserAsync(long userId, ReportUserDTO reportUserDto)
{
```

```
    // ... Validation (Prevent self-reporting, duplicates) ...

    var report = new UserReport
    {
        ReporterId = userId,
        Reason = reportUserDto.Reason,
        ReportedUserId = reportUserDto.ReportedUserId,
        Source = reportUserDto.Source
    };

    _context.UserReports.Add(report);

    // Count existing reports for the target user
    int existingReportsCount = await _context.UserReports
        .CountAsync(ur => ur.ReportedUserId == reportUserDto.ReportedUserId);

    // Business Rule: Auto-flag user if report threshold is met
    if (existingReportsCount + 1 >= 5)
    {
        var userToFlag = await _context.Users.FindAsync(reportUserDto.ReportedUserId);
        if (userToFlag != null && !userToFlag.IsFlagged)
        {
            userToFlag.IsFlagged = true;
        }
    }
    await _context.SaveChangesAsync();
}
```

5.5 User experience and frontend implementation

The frontend of PetMinder was built with **Blazor WebAssembly** and **Tailwind CSS** to deliver a responsive, single-page application (SPA) experience. The interface is designed to reduce friction for pet owners while providing sitters with the tools they need to manage their businesses effectively.

5.5.1 Global design and navigation

The application features a responsive layout that adapts to the user's role and device:

- **Role-Based Navigation**: The side navigation menu (`NavMenu.razor`) dynamically adjusts its content based on the user's authenticated role (Owner, Sitter, or Admin), ensuring that users only see relevant tools.

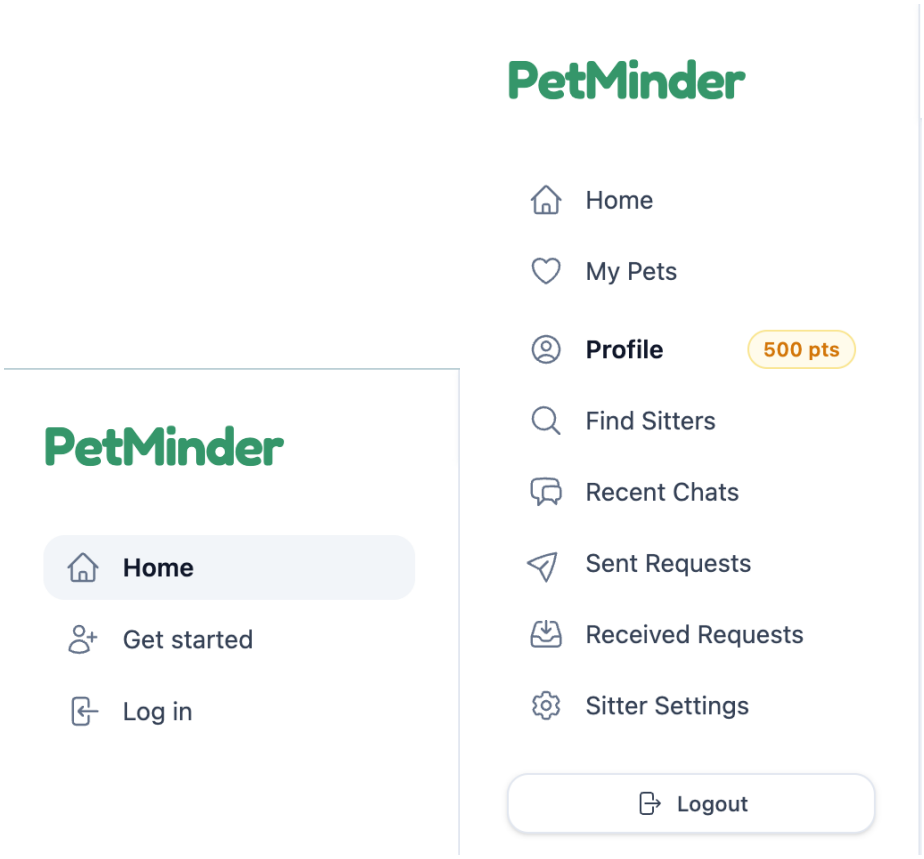

Figure 5.1: Role-based navigation: Guest view (left) vs. Authenticated user view (right).

- **Live Status Indicators**: A sticky header provides access to global states, such as a real-time notification bell and a "Points Balance" badge that updates instantly as transactions occur.

Figure 5.2: Real-time notification bell indicator.

- **Mobile-First Responsiveness**: The layout (`MainLayout.razor`) transitions from a persistent sidebar on desktop displays to a collapsible drawback menu on mobile devices, maintaining usability across all form factors.

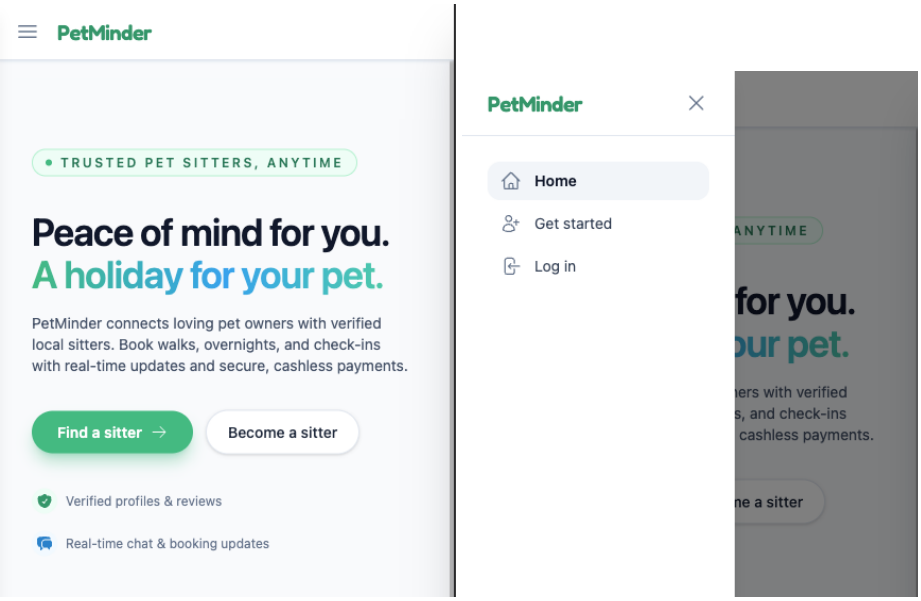

Figure 5.3: Mobile-responsive layout with collapsible sidebar.

5.5.2 Streamlined sitter discovery

The core discovery loop is implemented in `FindSitters.razor`, where the focus is on speed and transparency:

- **Context-Aware Search**: Users can filter sitters by specific dates, pet types, and location. The system utilizes the **Nominatim API** to geocode addresses upon registration. The integration logic ensures requests comply with OpenStreetMap's usage policies[18] by identifying the application via User-Agent headers, as shown in Listing.

Listing 7: Geocoding Address with Nominatim API

```
1  public async Task<(double Lat, double Lon)?> GetCoordinatesAsync(CreateAddressDTO
       address)
2  {
3      var query = $"{address.Street}, {address.City}, Poland";
4      // Nominatim requires a valid User-Agent header
5      var request = new HttpRequestMessage(HttpMethod.Get,
6          $"https://nominatim.openstreetmap.org/search?q={Uri.EscapeDataString(query)}&
               format=json&limit=1");
7
8      request.Headers.UserAgent.ParseAdd("PetMinderThesisApp/1.0");
9
10     var response = await _httpClient.SendAsync(request);
11     if (!response.IsSuccessStatusCode) return null;
12
13     using var doc = JsonDocument.Parse(await response.Content.ReadAsStringAsync());
14     var result = doc.RootElement.EnumerateArray().FirstOrDefault();
15
16     // Simplified JSON parsing logic
17     if (result.ValueKind != JsonValueKind.Undefined &&
18         result.TryGetProperty("lat", out var lat) &&
19         result.TryGetProperty("lon", out var lon))
20     {
21         return (
22             double.Parse(lat.GetString(), CultureInfo.InvariantCulture),
23             double.Parse(lon.GetString(), CultureInfo.InvariantCulture)
24         );
```

```
25      }
26      return null;
27  }
```

Once obtained, these coordinates are stored as double-precision floating-point numbers. This allows the `SitterSearchService` to perform in-memory distance filtering efficiently using the Haversine formula:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \quad (5.1)$$

Where:

- r is the Earth's radius (approx. 6,371 km),
- ϕ_1, ϕ_2 are the latitudes of the two points in radians,
- λ_1, λ_2 are the longitudes of the two points in radians.

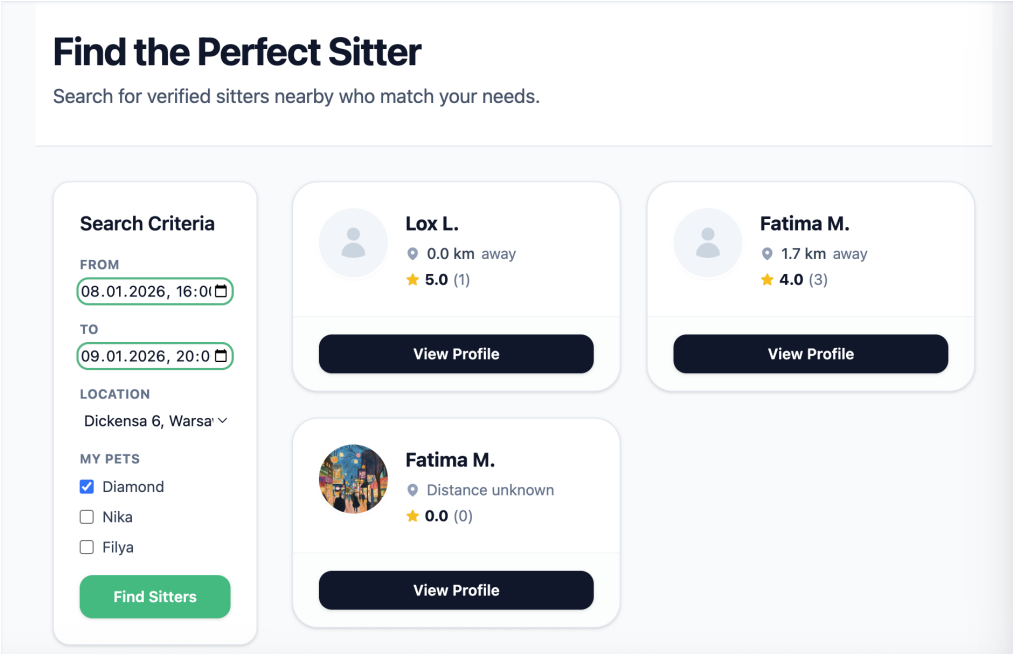

Figure 5.4: Context-aware sitter search interface.

- **Smart Indicators**: Results cards highlight key signals such as "Repeat User" badges or "Super Sitter" status to help owners make faster decisions.
- **Rich Profiles**: The detailed profile view (`SitterDetails.razor`) presents a "Dual Rating" system, showing a user's reputation both as a sitter and as a pet owner. This bilateral trust signal is unique to the PetMinder eco-system.
- **Availability Transparency**: A visual calendar prevents booking conflicts by only allowing owners to request times that match the sitter's configured availability slots.

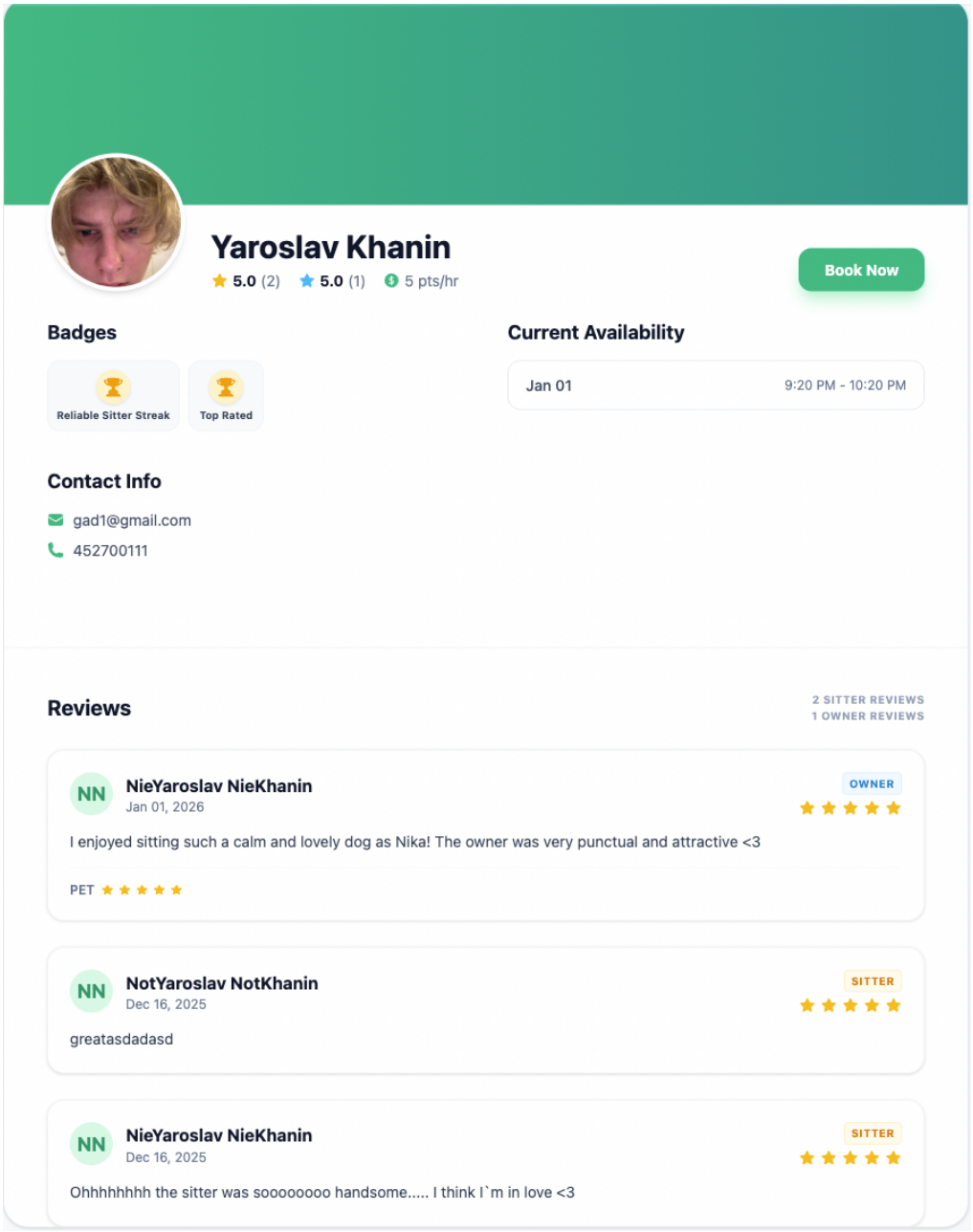

Figure 5.5: Detailed sitter profile with dual rating system.

5.5.3 Trust and verification systems

To mitigate risk in the marketplace, several verification layers are integrated directly into the user flow:

- **Identity Verification**: Before using the basic functionalities of the app, each user has to verify their account by confirming email and uploading a profile photo.

Listing 8: Verification Service Logic

```
1  public async Task CompleteVerificationStep(long userId, VerificationStep step)
2  {
```

```
    if (await IsVerificationStepComplete(userId, step)) return;
    int points = step switch
    {
        VerificationStep.EmailVerification => 50,
        VerificationStep.ProfilePhotoUpload => 50,
        _ => 0
    };
    // Award points and record step completion
    _context.UserVerificationSteps.Add(new UserVerificationStep
    {
        UserId = userId,
        Step = step,
        PointsAwarded = points
    });
}
```

- **Service Gating:** The implementation enforces a strict "Verification Gate" within the `BookingService`. Users who have not completed the required verification steps (Email + Photo) are programmatically blocked from creating booking requests, preventing anonymous or low-trust interactions.

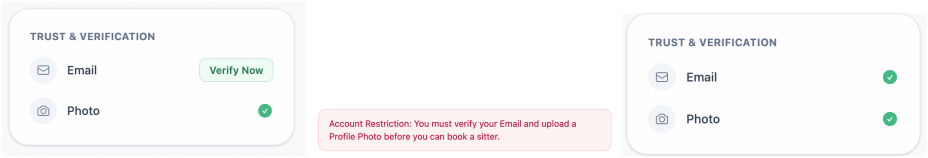

Figure 5.6: User identity verification flow: Initial prompt, restriction notice, and verified status.

- **Device Fingerprinting**: During registration, a JavaScript interop service captures a unique device fingerprint. This allows the system to detect and block potential ban-evasion attempts from the same hardware.

Listing 9: Device Fingerprinting Check

```
if (!string.IsNullOrEmpty(registerDTO.Fingerprint))
{
    // Prevent registration if device fingerprint already exists
    if (await _context.DeviceFingerprints.AnyAsync(df => df.Fingerprint ==
        registerDTO.Fingerprint))
    {
        throw new InvalidOperationException("Registration blocked by security
            policy.");
    }
}
```

- **Accessible Reporting Tools:** To ensure safety is accessible at all critical interaction points, the "Report User" action was integrated into three specific UI contexts:

  1. **Public Profiles:** On both Sitter and Owner profiles, allowing users to report suspicious bios or photos.
  2. **Active Conversations:** Inside the chat window (`Chat.razor`), enabling users to report harassment immediately during communication.
  3. **Booking Details:** Reports can be initiated within the booking management screens (`SentRequests.razor` and `ReceivedRequests.razor`) to address service failures or no-shows.

- **Safety Management**: A dedicated Admin Safety Center (`AdminReportedUsers.razor`) provides moderators with tools to investigate user reports, review chat logs, and issue bans if necessary.

5.5.4 The "points" economy

The marketplace runs on a custom virtual currency system designed to encourage velocity:

- **Transaction Ledger**: The `TransactionHistory.razor` page provides a transparent ledger of every earning, spending, and refund event.

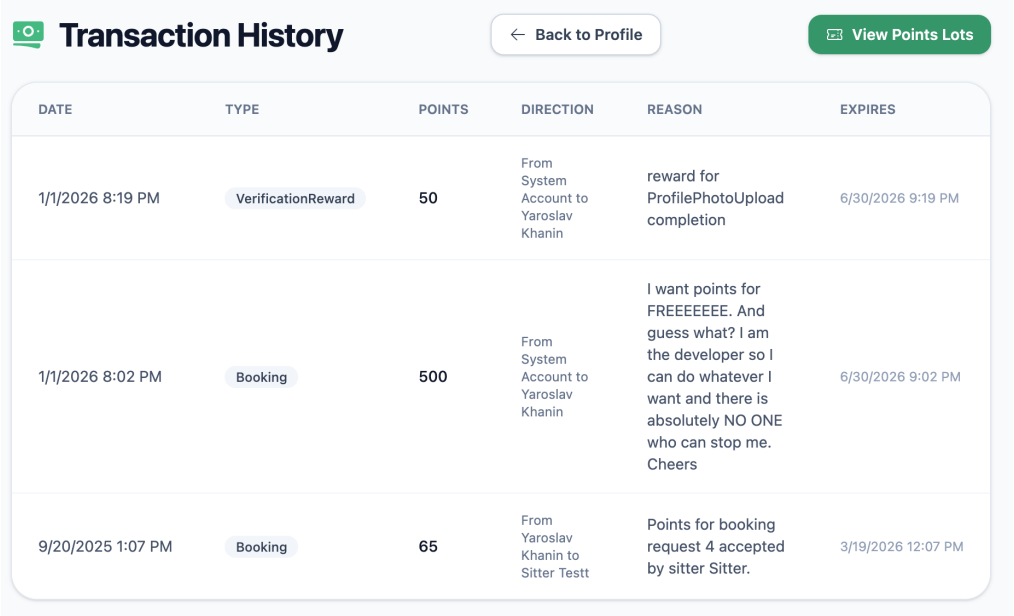

Figure 5.7: Complete transaction history ledger.

- **Point Lots**: Instead of a flat balance, points are managed as distinct "lots" with expiration dates (`PointsLots.razor`). This "use it or lose it" mechanic prevents hoarding and stimulates marketplace activity.

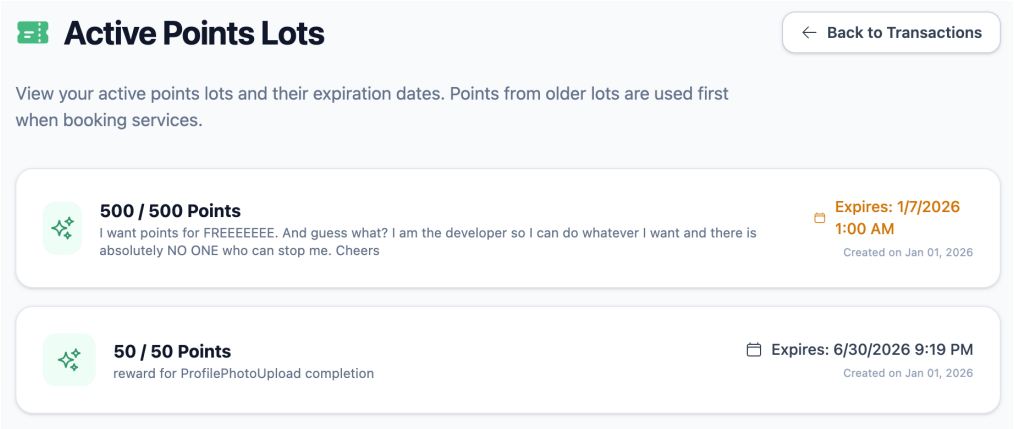

Figure 5.8: Points lots management with expiration dates.

- **Automated Expiration**: A **Hangfire** background job (`expire-points`), created using `Hangfire`, runs daily at 02:00 to strictly enforce validity periods[28]. This service invalidates expired lots and creates a transparent audit trail for the user.

Listing 10: Points Expiration Job

```
RecurringJob.AddOrUpdate<IPointsService>(
    "expire-points",
    service => service.ExpirePointsAsync(),
    Cron.Daily(2)
);
```

- **Referral System**: To drive organic growth, a referral engine allows users to generate unique invitation codes. Points are not awarded immediately upon registration; instead, the system waits for value generation. The `ReferralService` releases the 500 bonus points to the referrer only **after the new user successfully completes their first booking**, ensuring the platform rewards active engagement rather than just sign-ups.

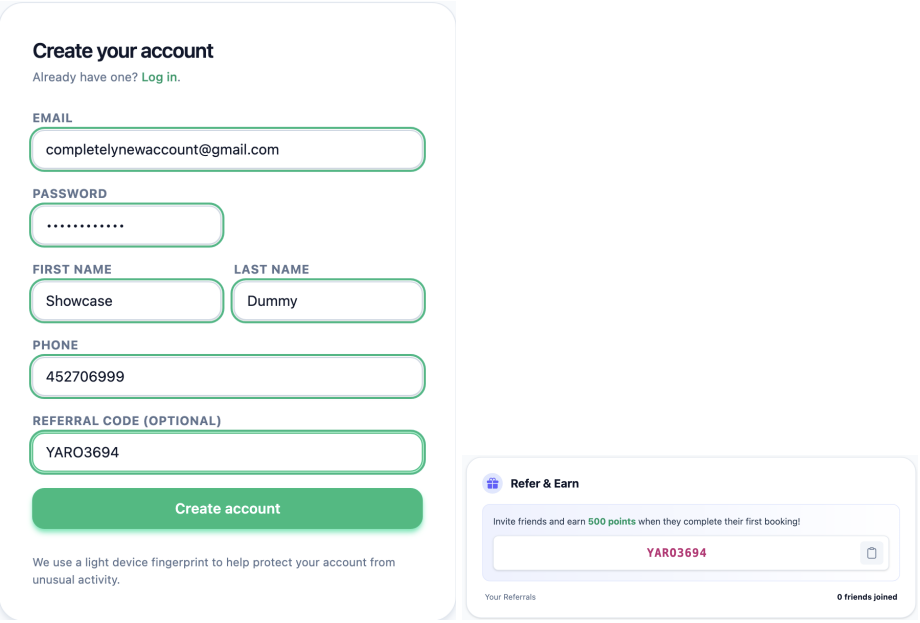

Figure 5.9: Referral system with invitation code generation.

5.5.5 Real-time negotiation and booking

The booking flow handles the complex negotiation often required for pet care:

- **Status Tracking**: Visual status indicators (Pending, Accepted, In-Progress, Completed) keep both parties informed of the booking's lifecycle stage.

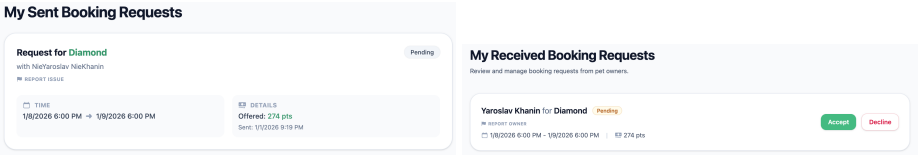

Figure 5.10: Booking requests: Sent (left) and Received (right) views.

- **Negotiation State Machine**: The booking request system allows for a "Propose Change" loop. Sitters and owners can negotiate dates or prices back-and-forth until both parties accept the final terms.

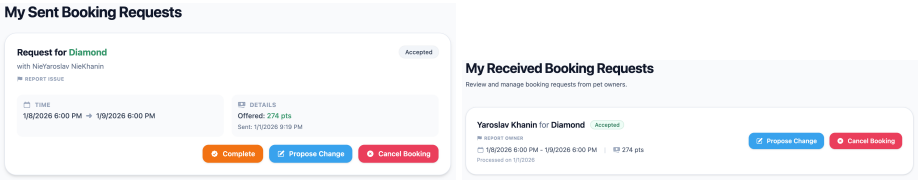

Figure 5.11: Accepted booking views: Owner perspective (left) and Sitter perspective (right).

- **Integrated Chat**: A SignalR-powered chat interface is embedded directly within booking requests, allowing for context-rich communication.

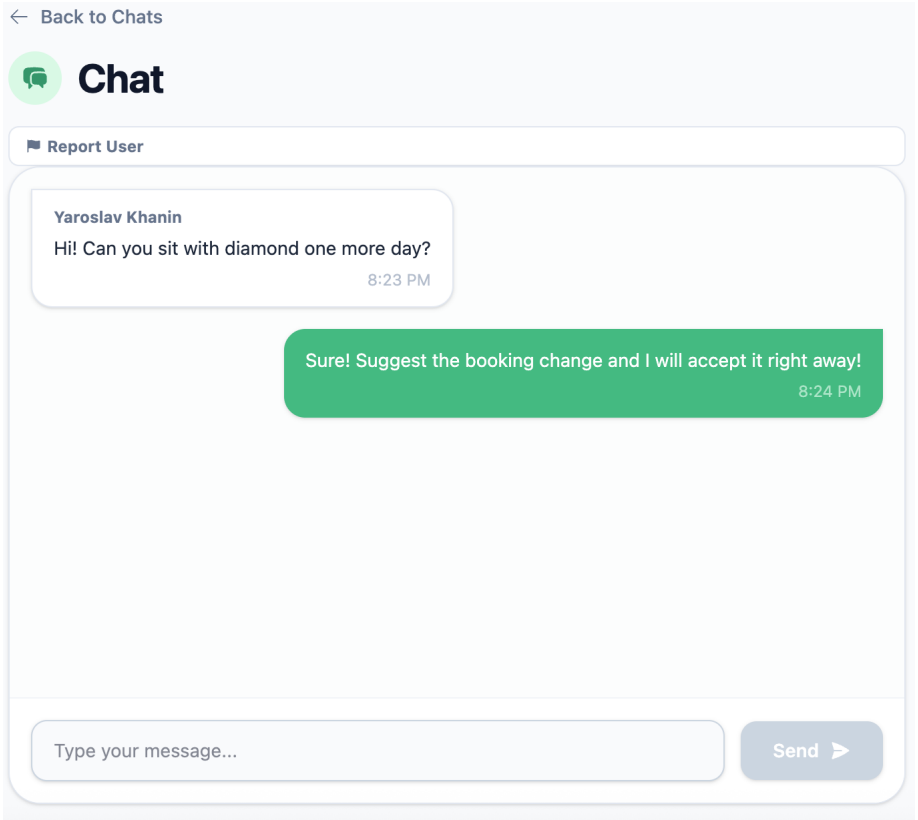

Figure 5.12: Real-time SignalR-powered chat interface.

5.5.6 Management and personalization

To ensure high-quality matches, the system captures detailed metadata:

- **Pet Complexity**: Owners can tag pets with their types and specific behavioral traits. These tags feed into the matching logic to find the most capable sitters.

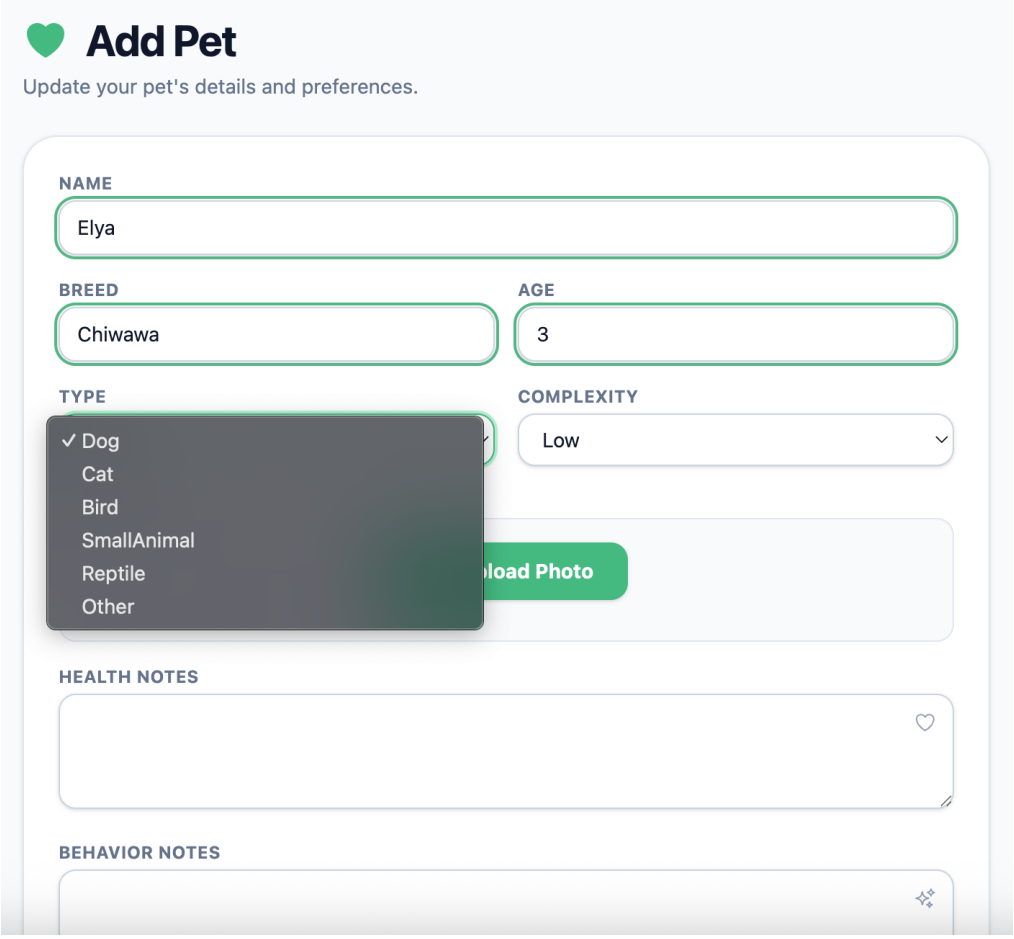

Figure 5.13: Pet creation form.

- **Granular Notifications**: Users have fine-grained control over which alerts they receive (Booking, Marketing, or System), preventing notification spam.

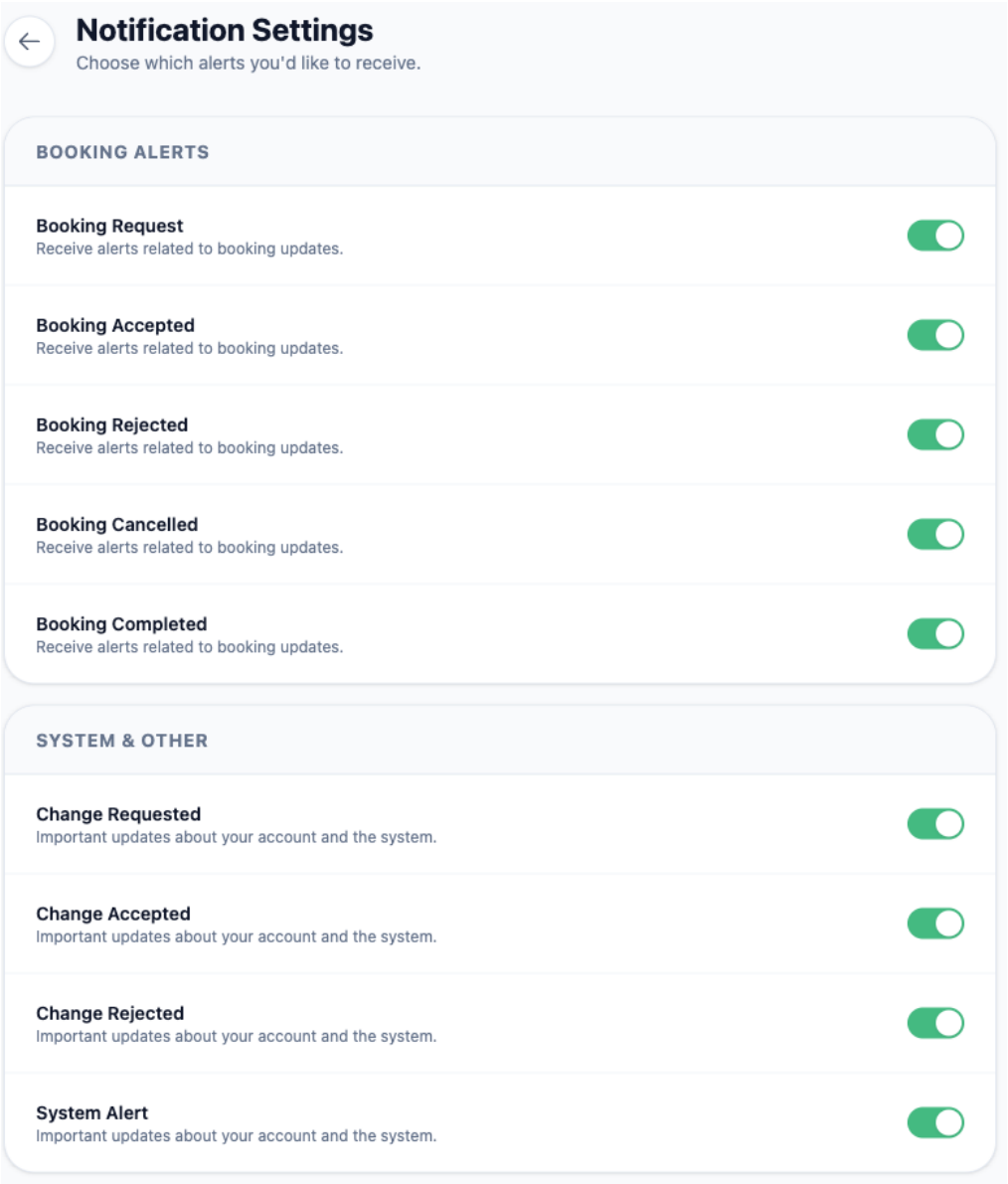

Figure 5.14: Granular notification settings interface.

5.5.7 Administrative governance

To ensure the safety and sustainability of the platform, a comprehensive Admin Panel was implemented. This module acts as a standalone application layer, utilizing a distinct layout structure (`AdminLayout.razor`) with a dedicated dark-themed sidebar to visually distinguish it from the standard user interface. Access is strictly governed by the `AuthorizeView` component, which renders a "Restricted Access" lock screen for any user lacking the `Admin` role.

The governance module consists of three primary pillars:

User management and investigation

The `UserManagement` view provides a high-level overview of the platform's population. It implements a **debounced search mechanism** that prevents database overload by waiting for 500ms of user inactivity before querying the API.

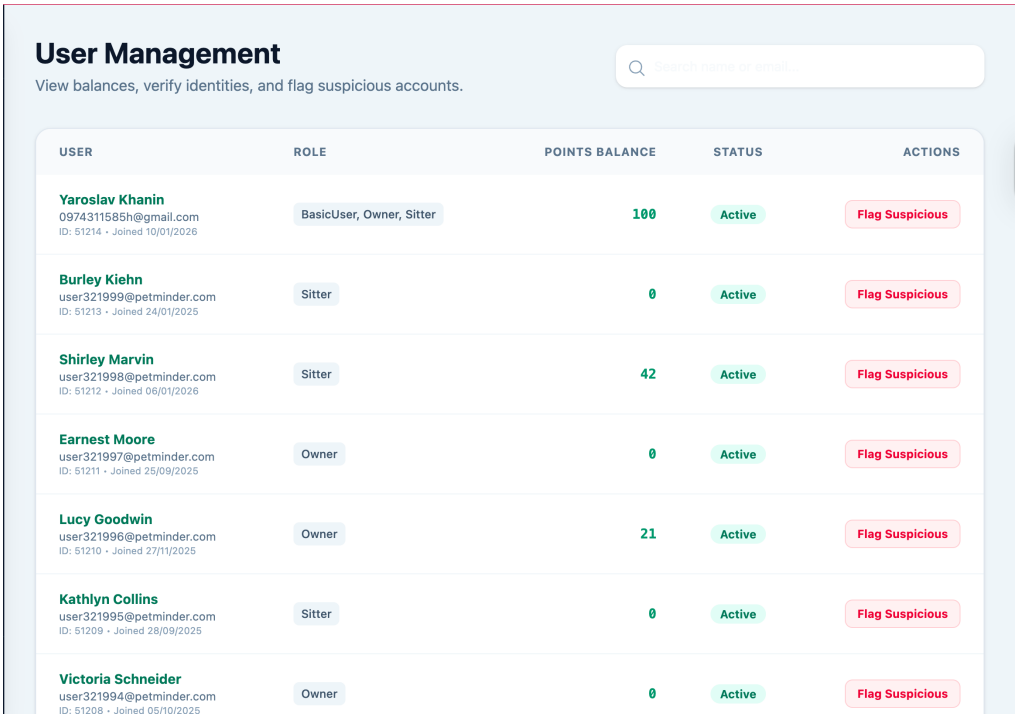

Figure 5.15: User Management interface with role tracking, live balance, and debounced search.

Administrators can drill down into specific accounts via the `AdminUserDetails` view. This interface serves as an investigative tool for dispute resolution, granting privileged access to data otherwise hidden for privacy reasons:

- **Financial Audit:** A complete log of the user's recent point transactions (credits and debits).
- **Booking Requests Logs:** A detailed log of the user's recent booking requests with their ID, Date, Details, and Status.
- **Communication Logs:** A read-only view of the user's SignalR chat transcripts, allowing admins to verify claims of harassment or misconduct.
- **Status Control:** Buttons to toggle "Flagged" status or issue permanent bans.

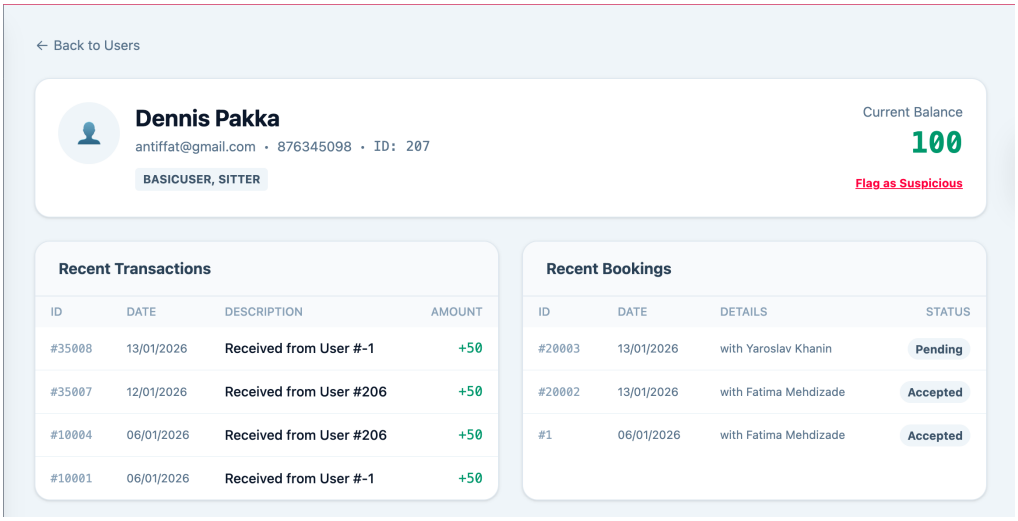

Figure 5.16: User Details interface within User Management tab with basic user details, current balance, flagging option, recent transactions and recent booking requests.

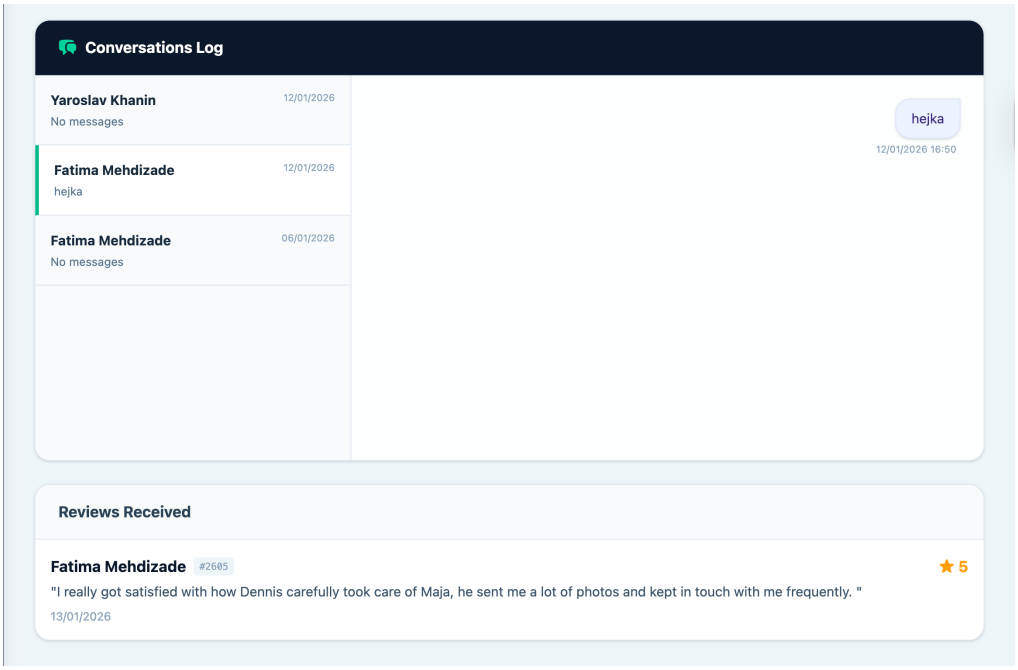

Figure 5.17: User Details interface within User Management tab with recent conversation logs, and recently received reviews.

Safety center and moderation

Community safety is managed through two dedicated dashboards: `ReportedUsers` and `ReportedReviews`.

- **Report Aggregation:** To prevent "brigading" (spam reporting), the system aggregates multiple reports against a single user into a unified case file. As seen

in Figure 5.18, moderators can view all the accusations and the specific reasons provided by reporters.

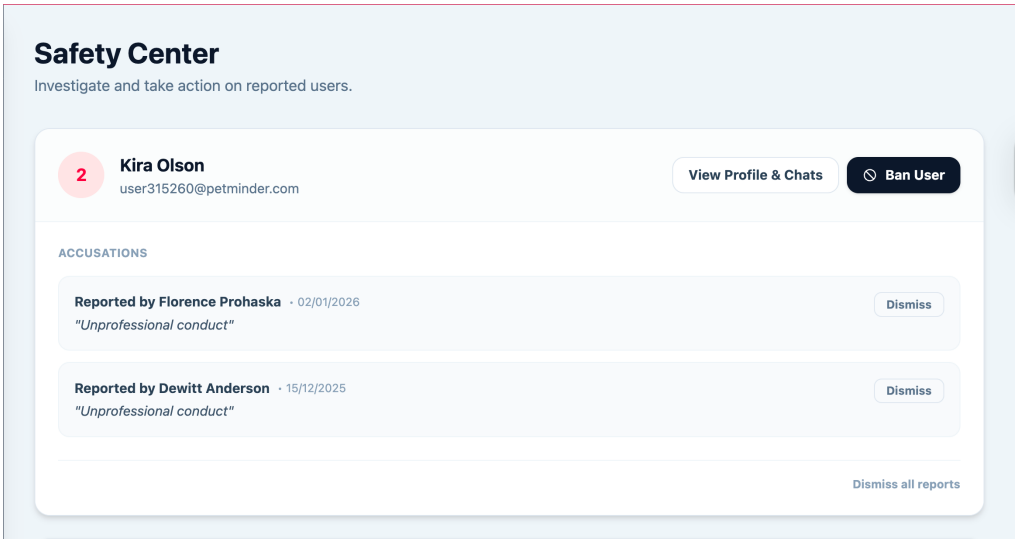

Figure 5.18: Admin Safety Center: Aggregated user reports with options to Ban or Dismiss.

- **Content Moderation:** The Review Moderation tool allows admins to analyze reported feedback. It provides dual actions: *Dismiss* (keeping the review public if the report is false) or *Delete* (permanently removing the content from the database).

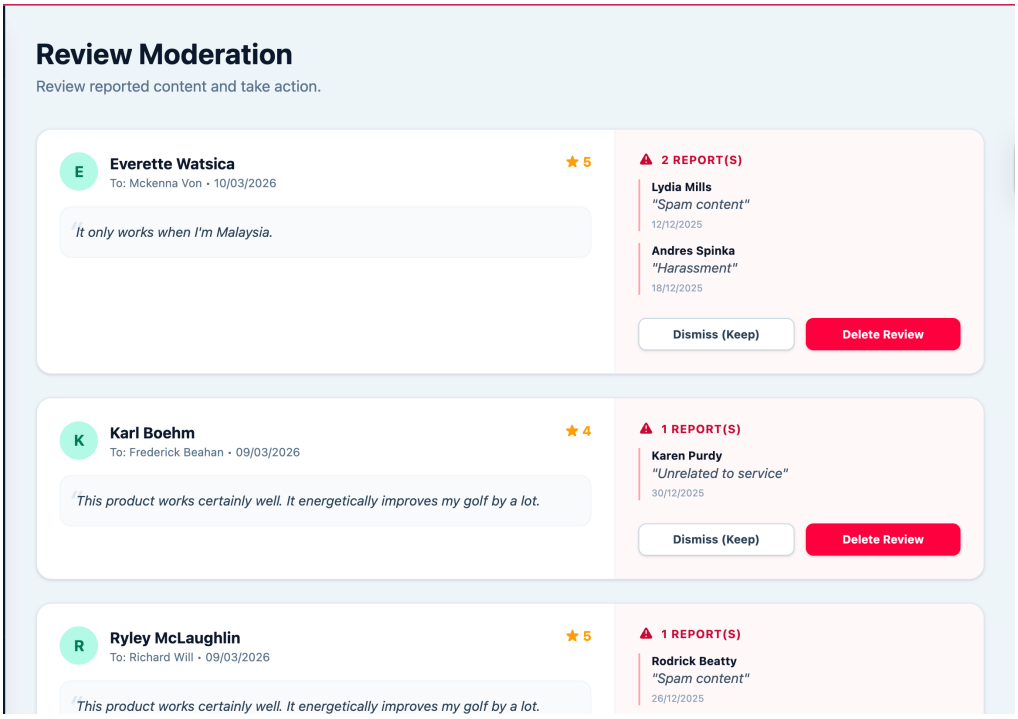

Figure 5.19: Review Moderation Interface: Analyzing reported feedback with options to dismiss or delete.

Economic configuration and disputes

The `PointsDashboard` allows administrators to manage the platform's economy without requiring code deployments or server restarts. It features a tabbed interface splitting functionality into two distinct areas:

1. **Policies:** A dynamic configuration form where administrators can adjust global economic parameters, such as the "Base Hourly Rate", "Minimum Spendable" threshold, and point "Expiry Days". This allows the platform to adapt to inflation or changing community needs in real-time.
2. **Dispute Resolution:** A transactional refund mechanism designed for conflict resolution. Admins input an `OriginalTransactionId` and a refund amount. The system validates that the refund does not exceed the original value before atomically reversing the points transfer between the Sitter and Owner.

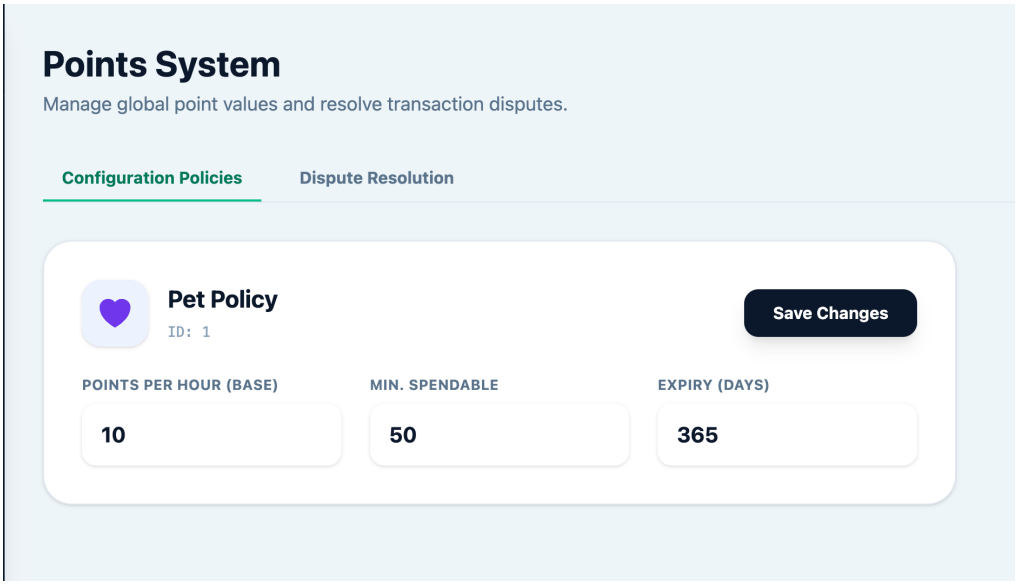

Figure 5.20: Points & Disputes Dashboard: Policy configuration interface.

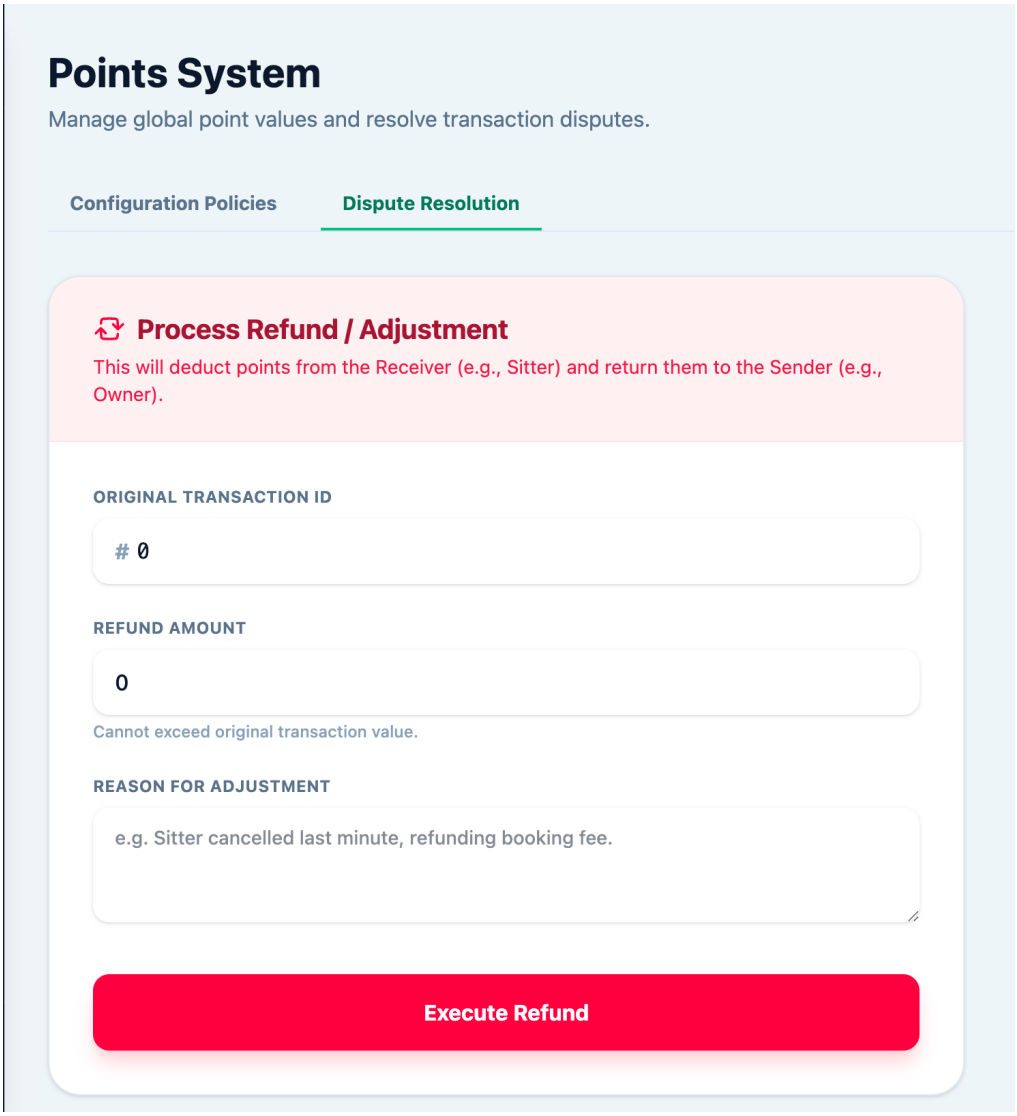

Figure 5.21: Points & Disputes Dashboard: Dispute Resolution interface.

6. Impact and sustainability

6.0.1 What pet owners and sitters gain from the platform

PetMinder fills a gap in the market by directly connecting pet owners with individuals who can provide care. Many professional pet services are costly and operate under strict, rigid rules. In contrast, PetMinder functions as a peer-to-peer system, where the same person who watches a dog one weekend may need help with their own cat the next. For pet owners, a major benefit is the ability to find nearby sitters who genuinely enjoy spending time with animals. Trusting a stranger with a pet can be difficult, but PetMinder reduces this uncertainty by displaying local sitters alongside reviews from other users. Since many sitters are pet owners themselves, they often show greater patience and empathy than large commercial services. For sitters, the platform offers a practical way to earn points, which can later be exchanged for pet care when needed. This creates a cycle: users contribute to the community when they can and benefit from it in return. The system also provides an opportunity for people who cannot own pets to interact with animals while building a positive reputation within their neighborhood. The platform also makes efficient use of otherwise unused time. Individuals who are already at home can care for a neighbor's pet without it feeling like a full-time job. As a result, the marketplace remains more flexible and less pressured than traditional pet care businesses.

6.0.2 Building a community of responsible participants

The primary objective of PetMinder is not only to develop an application, but to foster a community built on trust and responsibility. Because user reputations are public, participants are strongly encouraged to behave reliably and professionally. The rating system operates in both directions. Sitters are evaluated based on how well they care for pets, while owners are also reviewed. If an owner leaves their home in poor condition, provides unclear instructions, or repeatedly arrives late, their rating will decrease. This mutual accountability helps maintain high standards across the platform. Safety is further supported through basic verification measures. Users are required to confirm their email address and upload a profile photo before joining. Although this is not a full background check, it significantly reduces the likelihood of fake accounts. Additionally, the application uses device fingerprinting[29], allowing the system to restrict access from specific devices if a user repeatedly violates platform rules. Over time, these mechanisms contribute to the development of genuine trust. When owners find sitters they are comfortable with, they are likely to work with them again. This can naturally lead to stronger neighborhood connections, such as shared dog walks or exchanges of local advice. The referral system supports this organic growth, as people are more likely to join when recommended by someone they trust.

6.0.3 Ensuring long-term sustainability

For PetMinder to remain viable in the long term, it must be both technically robust and scalable. From a technical perspective, the database was designed to remain efficient even as the number of users grows into the thousands. The platform's internal economy is primarily based on points rather than direct monetary transactions. Points are earned by providing help and spent when assistance is needed. This closed-loop system reduces reliance on immediate cash payments and lowers barriers to accessing pet care. To encourage continued participation, points expire if they remain unused for an extended period, discouraging hoarding and promoting ongoing engagement. In the future, optional paid features may be introduced. For instance, sitters could pay a small fee to increase the visibility of their profile, or owners might purchase additional points in urgent situations. Importantly, the core functionality of the platform would remain free and centered on community participation. As the platform expands, the number of reports and disputes is expected to increase. To address this, an Admin Safety Center has been established to streamline moderation and identify rule violations more efficiently. There is also potential to involve highly rated, long-term users in basic moderation tasks as the community gets bigger. Finally, it is acknowledged that global scaling must occur gradually. A platform of this nature is only effective when there is a sufficient concentration of owners and sitters within the same area. For this reason, growth is planned on a town-by-town basis, ensuring that new users can reliably find the support they need.

7. Conclusion

7.1 Summary of achievements

The primary objective of this thesis was to design and implement a comprehensive pet care platform, PetMinder, which provides a secure and efficient environment for connecting pet owners with reliable sitters. Over the course of the project, several significant milestones were achieved:

- **Technological Foundation**: A robust, scalable architecture was developed using the ASP.NET Core ecosystem for the backend and Blazor WebAssembly for the frontend. This decoupled approach ensures high performance and a modern user experience.
- **Secure Communication**: Real-time chat functionality was integrated using SignalR, allowing users to communicate instantly within the platform without relying on third-party messaging services, thereby enhancing privacy and safety.
- **Advanced Scheduling and Booking**: A dynamic booking system was implemented, supporting sitter availability management and complex state-based workflows (Request, Accept, Complete, Cancel).
- **Location-Based Discovery**: Integrated geocoding services allow owners to find sitters based on geographical proximity, significantly improving the relevance of search results.
- **Gamification and Trust**: A points-based gamification system was introduced to incentivize high-quality service and community engagement. Furthermore, a verification workflow for sitters and a transparent review system were built to establish a foundation of trust.

The project successfully demonstrates how modern web technologies can be leveraged to solve real-world logistical problems in the pet care industry.

7.2 Limitations of current implementation

While the PetMinder platform addresses its core objectives, several limitations were identified during the development process that provide opportunities for future enhancement:

- **Mobile Experience**: Although the frontend is responsive and mobile-friendly, the lack of a native mobile application limits the use of certain hardware features, such as push notifications. Developing dedicated iOS and Android apps could significantly enhance user engagement and the application accessibility.

- **Advanced Recommendation Engine**: While proximity search is functional, the system could benefit from a more sophisticated recommendation algorithm that considers user preferences, historical pet matching, and seasonal demand.
- **Global Scalability**: The current geocoding and search implementation is optimized for local contexts. Scaling the application to support international markets with varying addressing standards and multi-language support remains a future task.

These limitations represent natural next steps for evolving PetMinder from a functional prototype into a production-grade commercial platform.

List of Figures

List of Listings

Bibliography

[1] E. S. Cahn, "Time banking," https://timebanking.org/wp-content/uploads/2020/05/Cahn-Gray-Stanford-Social-Innovation-Review.pdf, accessed: 26 January 2026.

[2] L. K. Ozanne, "The evolution of giving: An exploration of time banking as a community development instrument," https://scholarcommons.sc.edu/cgi/viewcontent.cgi?article=1211&context=senior_theses, accessed: 26 January 2026.

[3] M. Fowler, "Accounting transaction," https://martinfowler.com/eaaDev/AccountingTransaction.html, accessed: 26 January 2026.

[4] The Recursive, "Polish Petsy Secures €1.1M to Expand Verified Pet Care Services," https://therecursive.com/polish-petsy-secures-e1-1m-to-expand-verified-pet-care-services/, accessed: 26 January 2026.

[5] Rover, "Rover," https://www.rover.com/pl/, accessed: 26 January 2026.

[6] TrustedHousesitters, "Trustedhousesitters," https://www.trustedhousesitters.com, accessed: 26 January 2026.

[7] OLX, "Olx," https://www.olx.pl, accessed: 26 January 2026.

[8] Facebook, "Facebook," https://www.facebook.com, accessed: 26 January 2026.

[9] Distancia, "Haversine distance documentation," https://distancia.readthedocs.io/en/latest/Haversine.html, accessed: 26 January 2026.

[10] M. Möhlmann, "Trust in the sharing economy," https://www.researchgate.net/publication/326346569_Trust_in_the_Sharing_Economy, accessed: 26 January 2026.

[11] NIST, "Advanced Encryption Standard (AES) - FIPS 197," https://csrc.nist.gov/pubs/fips/197/final, accessed: 26 January 2026.

[12] Microsoft, "N-tier architecture style," https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier, accessed: 26 January 2026.

[13] James Amattey, "What is blazor webassembly (wasm)?" https://blazorise.com/blog/what-is-blazor-wasm, accessed: 26 January 2026.

[14] C. Sainty, "Blazored.localstorage," https://github.com/Blazored/LocalStorage, accessed: 26 January 2026.

[15] Microsoft, “Introduction to asp.net core,” https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core, accessed: 26 January 2026.

[16] Neon, “Serverless postgres architecture,” https://neon.tech/docs/introduction/architecture-overview, accessed: 26 January 2026.

[17] IronPDF, “Fluentemail c# guide,” https://ironpdf.com/blog/net-help/fluentemail-csharp/, accessed: 26 January 2026.

[18] OpenStreetMap Foundation, “Nominatim usage policy,” https://operations.osmfoundation.org/policies/nominatim/, accessed: 26 January 2026.

[19] DbFront, “Third normal form (3nf) explained,” https://dbfront.com/3nf, accessed: 26 January 2026.

[20] Tailwind CSS, “Utility-first fundamentals,” https://tailwindcss.com/docs/utility-first, accessed: 26 January 2026.

[21] NuGet, “Blazor.heroicons package,” https://www.nuget.org/packages/Blazor.Heroicons/, accessed: 26 January 2026.

[22] IEEE, “Gamification strategies for mobile device applications: A systematic review,” http://ieeexplore.ieee.org/document/7975943/, accessed: 26 January 2026.

[23] Microsoft, “Overview of asp.net core signalr,” https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction, accessed: 26 January 2026.

[24] PostgreSQL, “B-Tree Index Implementation,” https://www.postgresql.org/docs/current/btree-implementation.html, accessed: 26 January 2026.

[25] PostgreSQL , “Unique Indexes,” https://www.postgresql.org/docs/current/indexes-unique.html, accessed: 26 January 2026.

[26] B. Chavez, “Bogus: A simple fake data generator for .net,” https://github.com/bchavez/Bogus, accessed: 26 January 2026.

[27] PostgreSQL Tutorial, “PostgreSQL Covering Index,” https://www.pgtutorial.com/postgresql-tutorial/postgresql-covering-index/, accessed: 26 January 2026.

[28] Hangfire, “Documentation: Background jobs in .net,” https://docs.hangfire.io/, accessed: 26 January 2026.

[29] ResearchGate, “Impact of device fingerprinting on linking disparate synthetic identities,” https://www.researchgate.net/publication/396482867, accessed: 26 January 2026.