Architectural Document

A project by Team SEGFAULT
For EPI-USE LABS

COS301 Capstone Project
University of Pretoria

Mihir Arjun	- 21458759
Troy Clark	- 21436933
Deetlef Koen	- 20577304
Aliyah Limbada	- 22621522
Luke Nobrega	- 22517244

Table of Contents

Introduction 2

Design Patterns 3

1. Strategy 3

Class Diagram 4

Quality Attributes 5

1. Security 5
2. Availability 5
3. Usability 6
4. Scalability/Performance 6
5. Reliability 7

Architectural Styles 8

Architectural Patterns 9

1. Services-Oriented Architecture (SOA) 9
2. Repository Architecture pattern 10
3. Model-View-Controller (MVC) 11

Architecture Diagram 12

Constraints 13

Architectural Constraints 13

Performance Constraints 13

Availability Constraints 13

Technology Choices 15

Front-end Technologies 15

Backend Technologies 16

Database Technologies 18

Deployment Technologies 19

Final Technology Stack 21

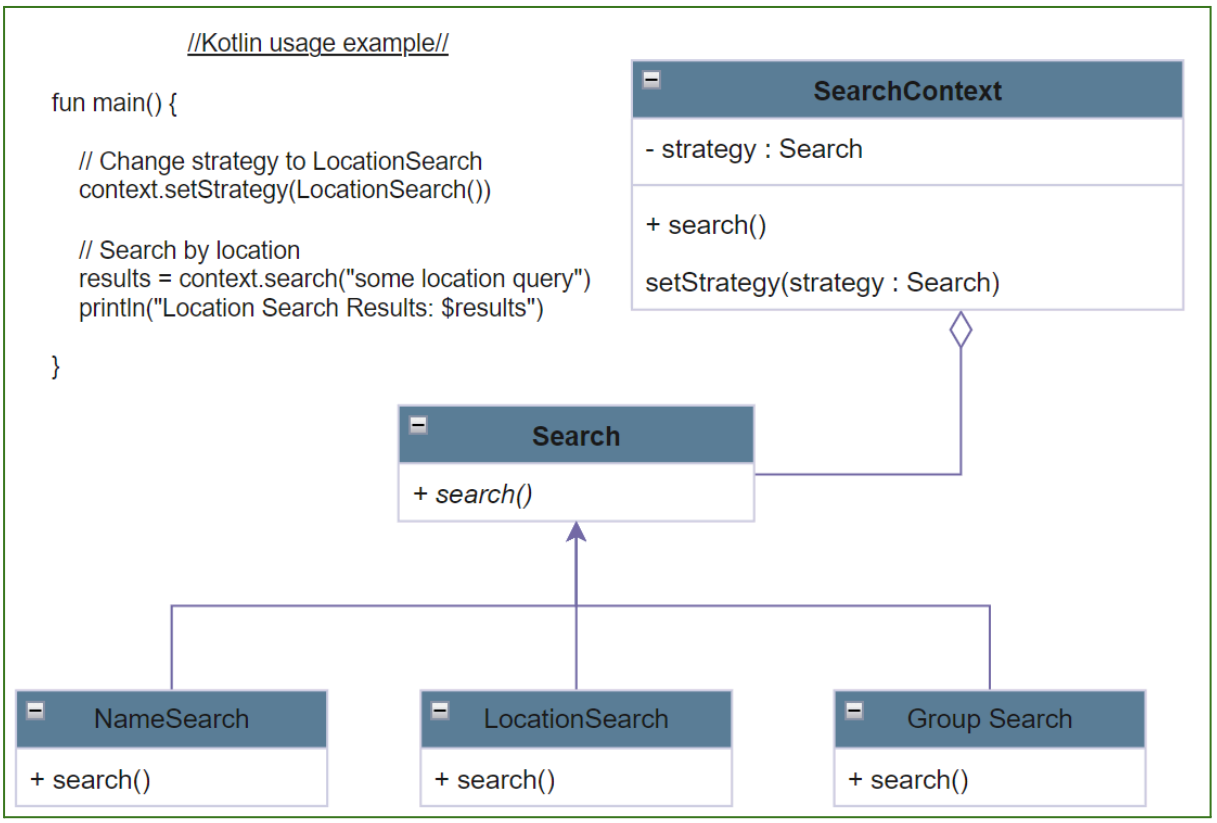
Introduction

Lookout is a modern Progressive Web Application that envisions a user-friendly platform revolutionising how nature enthusiasts, conservationists and hikers interact with and share information about their experiences in the outdoors. Lookout focuses on two main functionalities: Proof of sightings and Social interaction. It will allow users to download the application as a PWA to their native mobile devices, enabling easy access and usage on the go. The application is not just a tool limited to recording animal sightings; it's a community-driven platform that brings together nature enthusiasts from around the world. By combining modern technologies with a user-friendly interface, Lookout aims to enhance the animal spotting experience, making it more social, interactive, and rewarding for users.

Design Patterns

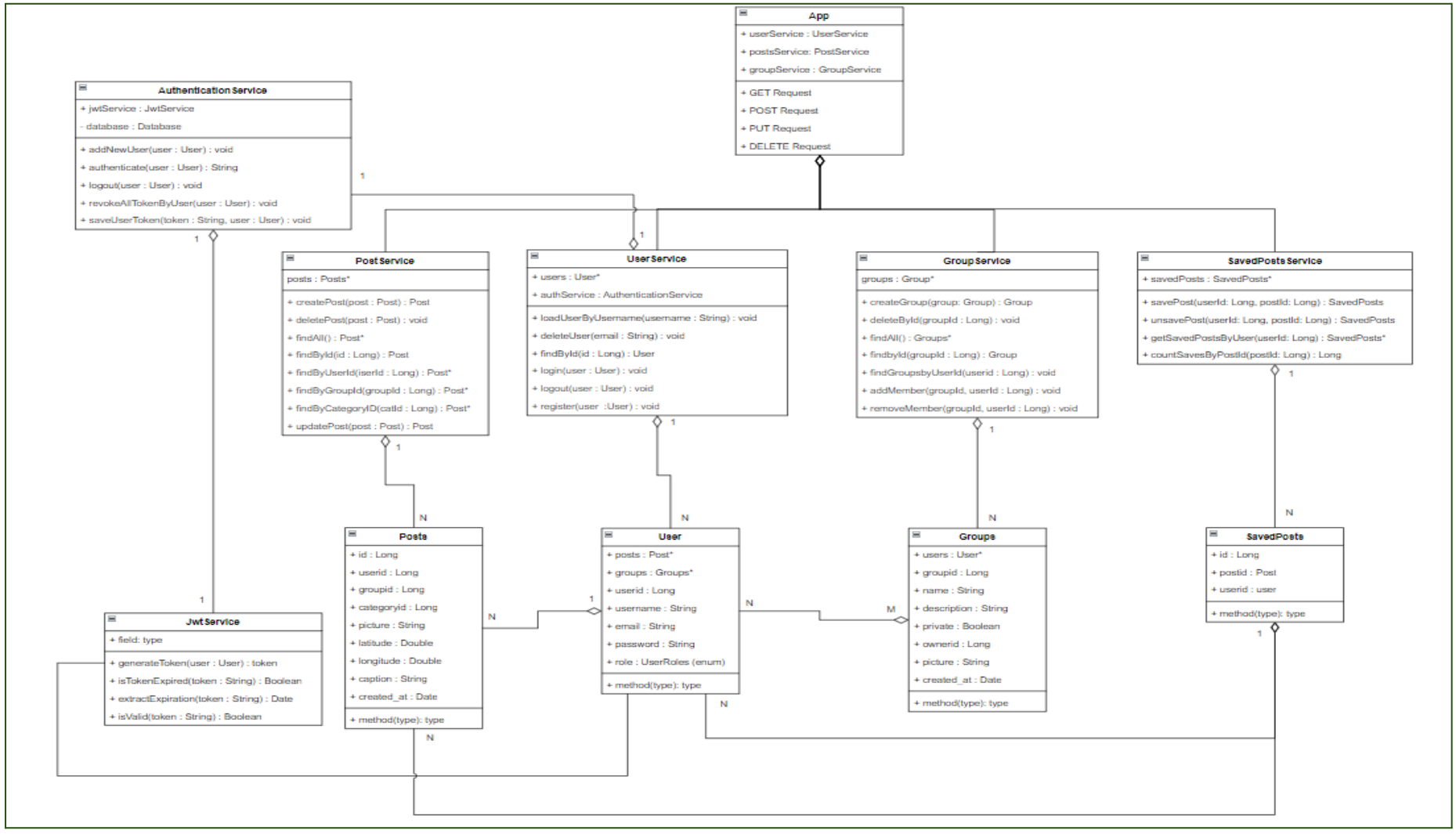
1. Strategy

Defining a family of algorithms, encapsulating each one, and making them interchangeable. This is useful for implementing different search or filter strategies for points of interest.



team.segfault.capstone@gmail.com

Class Diagram



team.segfault.capstone@gmail.com

Quality Attributes

1. Security

Security is a critical requirement for Lookout as it deals with sensitive user data, including location information and photos. Ensuring that this data is securely stored and transmitted is vital to protect users from privacy breaches and potential misuse of their personal information. The use of Google OAuth for secure login adds an extra layer of security by leveraging Google's robust authentication mechanisms. This protects user accounts from unauthorised access and ensures that data remains secure during the login process. Additionally, the system will include regular security audits and vulnerability assessments to identify and mitigate potential risks proactively, maintaining user trust and compliance with relevant data protection regulations.

Quantifications:

- All sensitive data must be encrypted using AES-256 during storage and TLS 1.3 during transmission.
- Implement multi-factor authentication (MFA) for additional security layers.
- Conduct quarterly security audits and vulnerability assessments.
- Ensure compliance with GDPR and other relevant data protection regulations.

2. Availability

Lookout must be available offline to accommodate users in remote areas with poor or intermittent internet connectivity. Offline functionality is essential for users who are often in areas with limited network coverage, such as hiking trails or remote camping sites. Ensuring the app works offline means users can still capture and save data locally, which will be synced once a connection is re-established. This functionality is crucial for maintaining user engagement and ensuring the app is useful in real-world scenarios where internet access may be unpredictable, thus fostering a seamless user experience regardless of connectivity.

Quantifications:

- Ensure offline data synchronisation within 30 seconds of regaining internet connectivity.
- Provide 99.9% uptime for the online services.
- Implement a local storage mechanism that can handle up to 1000 offline posts per user.

3. Usability

The application must be user-friendly and accessible to users of all ages. Given the target audience, which includes avid bird-watchers, hikers, and nature enthusiasts, the interface should be intuitive and easy to navigate. High usability ensures that users can efficiently interact with the application, post photos, join groups, and view maps without unnecessary complications. By reducing the learning curve, the app enhances user satisfaction and encourages more frequent use. Additionally, an emphasis on accessibility ensures that users with disabilities can also benefit from the app, broadening its usability and appeal.

Quantifications:

- Conduct usability testing with at least 20 users from diverse age groups.
- Ensure that 95% of users can complete core tasks (posting photos, joining groups) without guidance.
- Achieve a system usability score (SUS) of 80 or higher.

4. Scalability/Performance

Efficient image loading and data handling are critical for the scalability and performance of Lookout. As the user base grows and the volume of photos increases, the app must be able to handle large amounts of image data quickly and efficiently. Scalability ensures that the app can accommodate an increasing number of users and a growing dataset without degrading performance. This is essential for maintaining a smooth user experience and ensuring that the app remains responsive and fast. This includes optimising backend processes to handle peak loads effectively and ensuring the app performs well even under high user demand.

Quantifications:

- Support up to 10,000 concurrent users with a response time of less than 2 seconds for any request.
- Optimise image loading times to be under 1 second for images under 5MB.
- Ensure the system can handle a 20% increase in user base per month without performance degradation.

5. Reliability

Reliability is essential for building user trust in Lookout. The application must consistently perform its functions accurately, from posting photos to updating real-time location data. Reliability also involves ensuring that data is not lost and that offline posts are correctly synced once online. A reliable app ensures users can depend on it for accurate and timely information, especially in critical scenarios such as tracking animal sightings or reporting security concerns. Additionally, implementing robust error handling and recovery procedures will minimise downtime and enhance the overall user experience.

Quantifications:

- Achieve a Mean Time Between Failures (MTBF) of 6 months for critical systems and 3 months for non-critical systems.
- Ensure a Mean Time To Repair (MTTR) of less than 2 hours for minor issues and less than 5 hours for major issues.
- Implement redundant data storage solutions to prevent data loss, with data recovery time under 1 hour.

Architectural Styles

Our architecture employs a Service-Oriented Architecture (SOA) to divide the system into independent, modular services, each responsible for specific functions such as authentication, post management, group interactions, and map services. SOA enhances flexibility and reusability across different applications, allowing services to be independently scaled based on demand. By enabling each service to operate autonomously, we ensure high reliability and resilience, with robust failover and redundancy mechanisms. This separation also supports integrability and interoperability, facilitating the addition of new features and services without disrupting existing ones.

In addition to SOA, we adopt a Repository Architecture. This structure enhances security and scalability, The Data Access manages database interactions, ensuring data consistency and integrity. We chose the Repository Architecture specifically because it aligns well with our mono-repo setup, which houses both the frontend and backend in a single repository. This integration facilitates streamlined development and deployment processes, enhancing collaboration and consistency across the project.

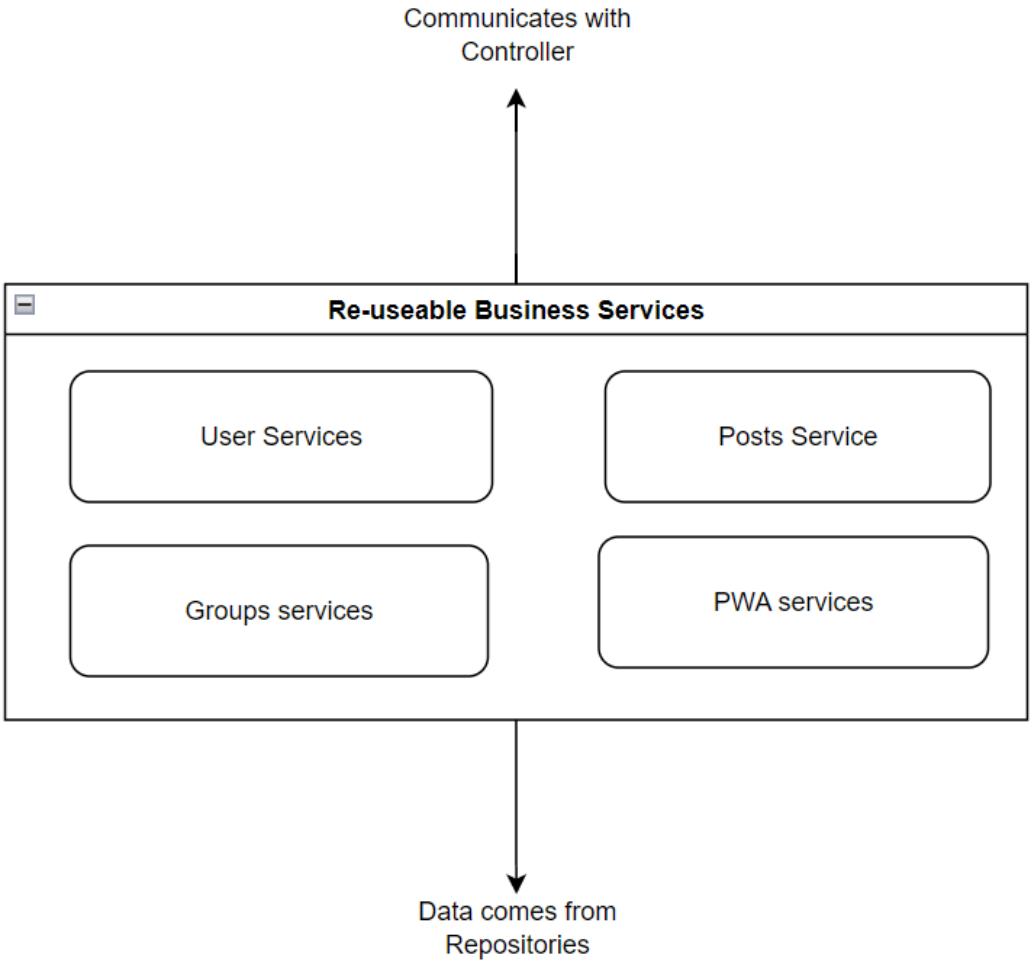
We also integrate the Model-View-Controller (MVC) pattern to further organise our codebase and promote separation of concerns. The Model represents the data and business logic, handling data access and including reusable business services. The View is responsible for presenting data to users and handling client-side logic, ensuring a responsive user interface. The Controller acts as an intermediary, processing user inputs, managing data flow, and coordinating interactions between the Model and View. MVC enhances maintainability by allowing developers to work on different components independently, reducing the risk of unintended side effects and facilitating collaboration.

Overall, our architectural approach—comprising SOA, Repository Architecture, and MVC—ensures a robust, flexible, and scalable system. It supports efficient development, easy maintenance, and seamless integration of new features, while maintaining high performance, security, and reliability. This comprehensive strategy is designed to meet the diverse needs of our users, providing a seamless and engaging experience for nature enthusiasts, conservationists, and hikers using Lookout.

Architectural Patterns

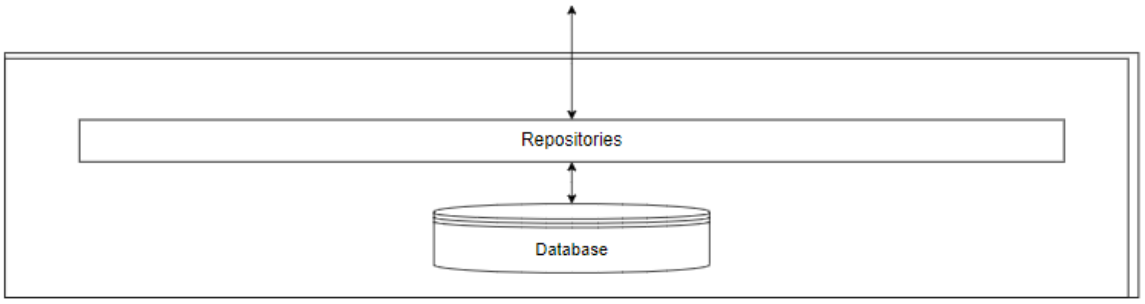
1. Services-Oriented Architecture (SOA)

We chose SOA for its ability to divide the system into independent services, each responsible for a specific function, which communicates over a network. This pattern enables separate services for authentication, posts, groups, maps, and more. SOA enhances flexibility, allowing service reuse across different applications, and offers high scalability as services can be independently scaled based on demand. Its independent service nature ensures high reliability and resilience with robust failover and redundancy mechanisms.
SOA aligns well with our quality requirements by supporting scalability/performance and reliability, as services can be scaled and maintained independently without affecting the entire system.



2. Repository Architecture pattern

The Repository pattern was chosen for its ability to encapsulate data access logic, promoting code reusability and maintainability. It also allows us to manage data consistency and integrity efficiently. By isolating data access in the repository, we ensure that business logic remains decoupled from specific database concerns, enhancing scalability and performance. This pattern aligns well with our mono-repo setup, where both the frontend and backend are integrated within a single repository, streamlining development and deployment processes. Additionally, the repository pattern contributes to the security, scalability, and availability of the system by providing well-defined interfaces and ensuring that distinct layers can be independently secured and scaled based on demand. Repository Architecture pattern is great for security because it prevents the business logic from directly communicating with the database.

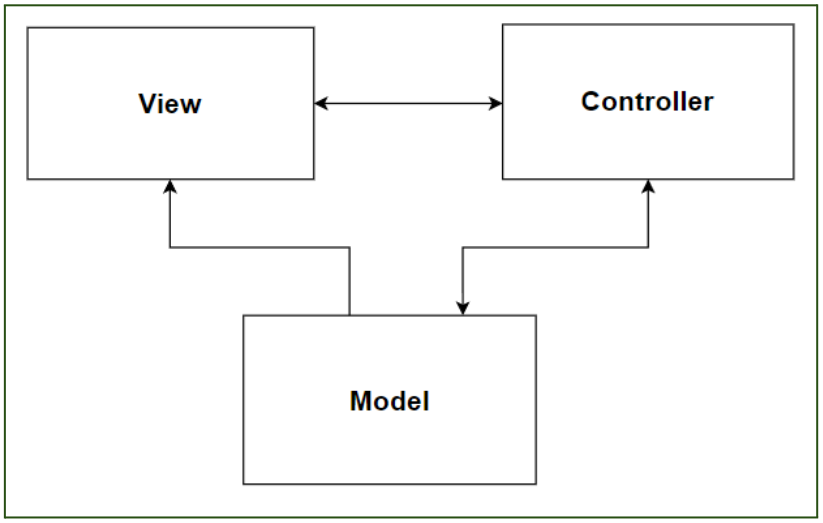


3. Model-View-Controller (MVC)

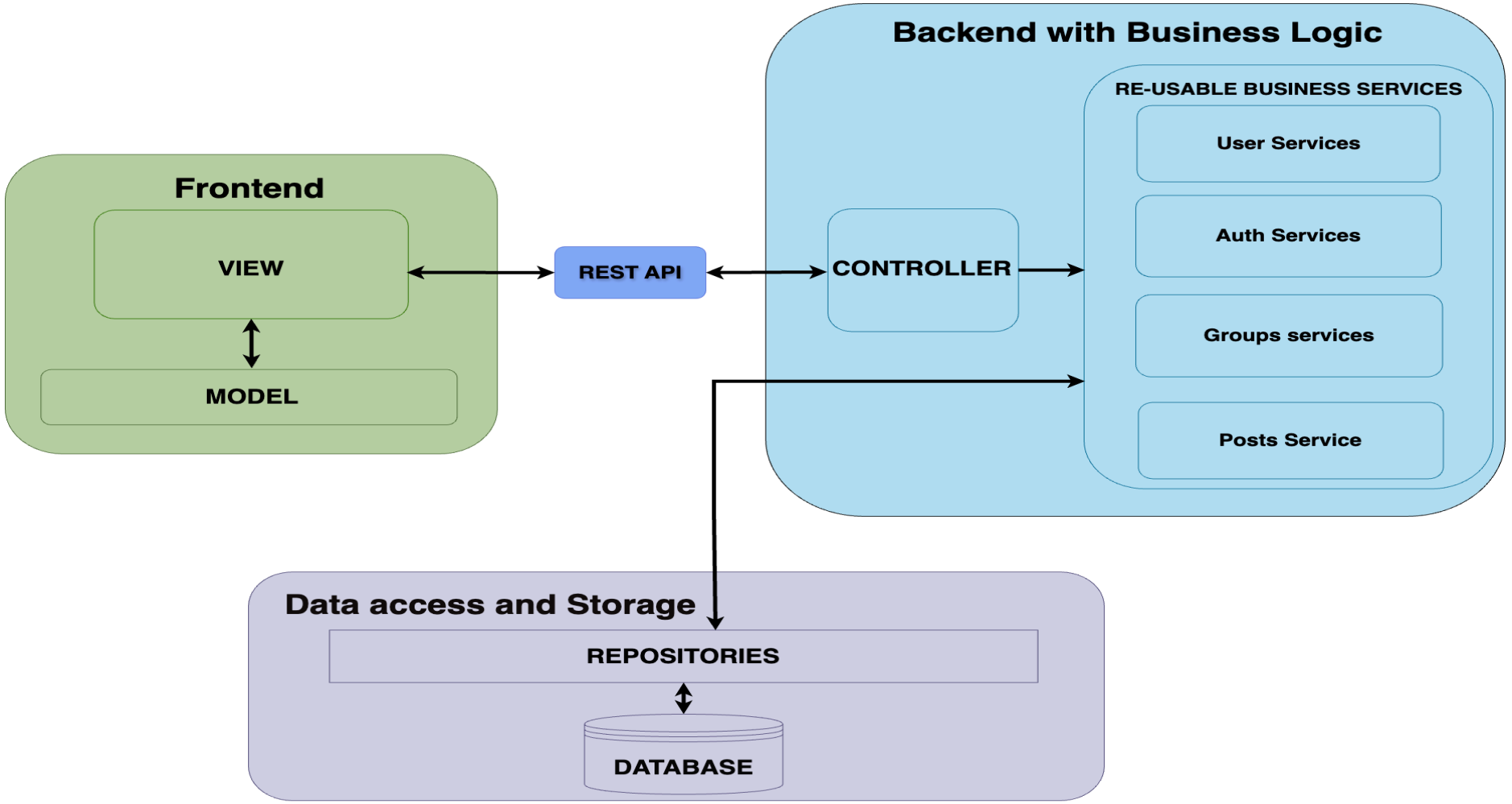
The MVC architectural pattern is essential for designing and organizing code in a maintainable and scalable manner. This pattern divides the application into three interconnected components, which helps separate the internal representations of information from the ways that information is presented and accepted by the user.

- Model:
  - Represents the data and business logic of the application.
  - Handles data access and includes reusable business services.
  - Interacts with the database to fetch and store data, providing it to the Controller and View as needed.
- View:
  - Acts as the presentation layer, displaying data to the user in a specific format.
  - Receives data from the Model and sends user inputs to the Controller.
- Controller:
  - Intermediary between the View and the Model, processing user inputs.
  - Manages the flow of data, updating the Model or View as necessary.
  - Handles user interactions such as creating geo-tagged points, managing authentication, and coordinating with the Model for data operations.

MVC was chosen for its responsive design due to the dedicated View component and its simplicity in implementation and maintenance. It also offers excellent scalability, aligning with our quality requirements for usability, scalability/performance, and reliability. The clear separation of concerns within MVC allows for efficient parallel development and reduces the risk of unintended side effects during updates.



Architecture Diagram



team.segfault.capstone@gmail.com

Constraints

Architectural Constraints

- The application must be implemented as a Progressive Web App (PWA) to ensure cross-platform compatibility and ease of access on both desktop and mobile devices.
- The use of a web framework is mandatory, and the solution must be deliverable to mobile devices using PWA technologies.
- The system architecture must support offline functionality, allowing the application to store data locally and synchronise with the server when an internet connection is available.
- The application must use Google OAuth for user authentication to ensure secure and reliable login functionality.
- The solution must be hosted on cloud platforms such as AWS, Azure, or GCP to leverage scalable and reliable infrastructure services.
- The system architecture must be developed through Domain Driven Design as per the request of our clients so that we can better understand the problem that we are solving.
- The system architecture must also make use of the Real-time Architecture for animal sightings in particular as requested by the clients.

Performance Constraints

- The architecture must support efficient image loading and handling to ensure smooth performance, even with a large volume of image data.
- The solution should aim to load all pages within a 1-second timeframe to provide a responsive user experience.

Availability Constraints

- The system must support offline functionality, enabling users to capture and save data locally and synchronise it with the server when online.

- Cloud hosting on platforms like AWS, Azure, or GCP ensures high availability and reliability, leveraging the infrastructure's built-in redundancy and failover mechanisms.

- The application should be designed to handle increased loads without sacrificing availability, utilising scalable cloud resources to accommodate growing user demand.

Technology Choices

Front-end Technologies

React: is a popular JavaScript library used for building user interfaces, especially single-page applications where dynamic user interactions are frequent. This was requested by the client due to its component-based architecture, performance, and strong PWA support.

- **Pros:**
  - **Component-Based Architecture**: Promotes reusability and maintainability of UI components, essential for a scalable PWA.
  - **Virtual DOM**: Enhances performance by reducing the number of direct manipulations to the actual DOM.
  - **Strong Ecosystem**: A wealth of libraries and tools for routing, state management, and more.

- **Cons**:
  - **Learning Curve**: Requires understanding JSX, a syntax extension, and component lifecycle methods.
  - **Additional Setup**: Needs supplementary libraries (e.g., Redux for state management, React Router for routing).

Fit with Lookout's Architecture: React's component-based structure and extensive ecosystem align well with Lookout's need for a dynamic and responsive PWA. It supports the requirement to create an interactive user interface that can handle real-time updates efficiently.

Backend Technologies

Spring Boot with Kotlin: Spring Boot is a Java-based framework used for creating stand-alone, production-grade Spring applications quickly. Kotlin is a modern, concise, and safe programming language that fully interoperates with Java. This was requested by the client due to its rapid development capabilities, scalability, and modern syntax.

- **Pros**:
  - **Rapid Development**: Spring Boot's auto-configuration and starter dependencies accelerate development.
  - **Microservices Support**: Facilitates building scalable and maintainable microservices architectures.
  - **Kotlin Compatibility**: Kotlin's concise syntax reduces boilerplate code, and its null safety features enhance reliability.
- **Cons**:
  - **Complexity**: Can be complex for beginners due to its extensive features and configurations.
  - **Memory Usage**: Java-based frameworks can be more memory-intensive compared to some other languages.

Fit with Lookout's Architecture: Spring Boot's robust ecosystem and Kotlin's concise syntax are well-suited for Lookout's backend. They support building scalable and maintainable services, aligning with the need for real-time data processing and integration with a PostgreSQL database.

Python: Python is a versatile, high-level programming language renowned for its simplicity and extensive library support. In Lookout's architecture, Python is leveraged both for building the backend services and developing AI models that predict points of interest based on collected data. This dual usage is driven by Python's robust ecosystem, ease of integration, and strong community support.

- **Pros**:
  - **Extensive Libraries and Frameworks**: Python boasts powerful libraries such as TensorFlow, PyTorch, and scikit-learn, facilitating the development and deployment of sophisticated AI models.
  - **Scalability and Flexibility**: Python can handle various aspects of the application, from serving RESTful APIs to managing real-time data processing, making it adaptable to evolving project needs.
  - **Rapid Development and Prototyping**: Python's concise and readable syntax accelerates development cycles, allowing for quick

iteration and prototyping, which is essential for meeting Lookout's tight delivery requirements.

- **Cons**:
    - **Performance Limitations:** Python is generally slower compared to compiled languages like Java or Kotlin. This can impact the performance of real-time features, especially under high load, potentially affecting Lookout's requirement for near real-time data processing.
    - **Deployment Overhead:** Managing Python environments and dependencies can be more involved, especially when ensuring consistency across development, testing, and production environments.

Fit with Lookout's Architecture: Python is well-suited as a backend technology for Lookout's architecture, handling core tasks like user authentication, API development, and real-time data processing. Its powerful web frameworks like Django and Flask efficiently manage server-side logic, while libraries for machine learning enable predictive AI models. Python's extensive support for database management and integration with tools like Prometheus makes it ideal for the scalable, data-driven features of Lookout. However, performance optimization may be necessary to ensure smooth real-time functionality and efficient handling of multiple tasks.

Database Technologies

PostgreSQL: PostgreSQL is an advanced, open-source relational database system known for its robustness and standards compliance. This was requested by the client due to its reliability, performance, and advanced features make it the best choice for managing Lookout's data.

- **Pros**:
    - **ACID Compliance**: Ensures reliable transactions.
    - **Extensibility**: Supports advanced data types and custom functions.
    - **Performance**: Handles complex queries efficiently.
- **Cons**:
    - **Setup Complexity**: Initial setup and configuration can be complex.
    - **Resource Intensive**: May require more resources compared to some lightweight databases.

Fit with Lookout's Architecture: PostgreSQL's robustness and support for complex queries and transactions align well with Lookout's need for reliable and efficient data storage and retrieval. Its extensibility and performance capabilities support the application's geospatial and real-time data requirements.

Deployment Technologies

AWS: Amazon Web Services (AWS) is a comprehensive cloud platform offering a wide range of services for computing, storage, and networking.

- **Pros**:
  - **Scalability**: Easily scales resources up or down based on demand.
  - **Security**: Offers robust security features and compliance certifications.
  - **Global Reach**: Extensive global infrastructure ensures low latency and high availability.
- **Cons**:
  - **Cost**: Can become expensive with large-scale usage.
  - **Complexity**: Steeper learning curve due to the wide range of services and configurations.

Fit with Lookout's Architecture: AWS's scalability, security, and extensive service offerings align well with Lookout's requirements for a reliable and scalable deployment environment. It supports the use of various services such as EC2, S3, RDS, and Lambda, which are essential for a PWA.

GCP: Google Cloud Platform (GCP) offers a suite of cloud computing services similar to AWS.

- **Pros:**
  - **Big Data and Machine Learning:** Strong capabilities in data analytics and machine learning.
  - **Performance:** High-performance network and compute options.
  - **Competitive Pricing:** Often more cost-effective for certain workloads.
- **Cons:**
  - **Service Range:** Slightly fewer services compared to AWS.
  - **Ecosystem:** Smaller community and ecosystem than AWS.

Fit with Lookout's Architecture: GCP's strong data analytics and machine learning capabilities can be beneficial for Lookout, especially for advanced features like AI predictions. Its high-performance infrastructure ensures a reliable deployment environment.

Azure: Microsoft Azure is a cloud computing service offering a range of services for building, testing, and managing applications.

- **Pros:**
    - **Integration with Microsoft Products:** Seamless integration with Microsoft tools and software.
    - **Hybrid Cloud Support:** Strong support for hybrid cloud deployments.
    - **Enterprise Support:** Robust support for enterprise applications and workloads.
- **Cons:**
    - **Complexity:** Can be complex to set up and manage.
    - **Service Range:** Some services may not be as mature as those offered by AWS.

Fit with Lookout's Architecture: Azure's integration with Microsoft products and enterprise support can be advantageous for organisations already using Microsoft tools. Its hybrid cloud capabilities offer flexibility in deployment options.

Deployment Decision: Given its comprehensive services, scalability, and global infrastructure, AWS is the optimal choice for deploying Lookout. While GCP is a strong contender, AWS is chosen due to its broader range of services and larger ecosystem, which better supports the overall architecture needs of Lookout. On the other hand, Azure offers strong enterprise support, however AWS is preferred for Lookout due to its comprehensive service range and extensive global infrastructure.

Final Technology Stack

- **Frontend:** **React** for its component-based architecture, performance, and strong PWA support.
- **Backend:** **Spring Boot with Kotlin and Python** for its rapid development capabilities, scalability, and modern syntax.
- **Database:** **PostgreSQL** for its robustness, performance, and advanced features.
- **Deployment**: **AWS** for its comprehensive services, scalability, and global infrastructure.