Software Requirements Specification

For

By

Millennium

Name	Student Number
Charl Pretorius	u22519042
Ivan Horak	u21456552
Matthew Botha	u20514612
Justin Hudson	u21543152
Jarod Jeffery	u18003193

Contents

Introduction:

BeakPeek aims to grow the bird watching community in South Africa by making it easier to find, identify and predict the occurrence of bird species.

BeakPeek will allow users to see what bird species can be found in their area and provide conservationists with an analytical overview of the data to easily gauge the growth and decline of local bird species.

User Characteristics:

User 1 – Birders: The primary user of the application, birders can use the application to view all kinds of birds in South Africa and where to find them using SABAP2's public database. Birders can view what birds can be seen in the pentad they are currently in. They can view the list of birds that they have seen and add their sighting to the database if they want to. Birders can identify birds by taking a photo of it. Birders can identify birds through specialised filters. They can also listen to what a bird sounds like.

User 2 – Conservationists: Conservationists can use the data uploaded by users and by SABAP2 to receive an indication of what a specific bird population is doing, more sightings than expected could indicate an increase in the birds' population and vise versa. Further can use this data to track a specific bird should the species be sufficiently rare.

User 3 – Admin: Manually confirms Conservationists authenticity and will interact with the administrative systems.

Constraints:

1. Only a mobile application will be developed.
2. Only South African Bird data through SABAP2 will be available.
3. The application will only be available in English.
4. Application requires internet connection at least once to download bird information for offline use.
5. Application will only be published on the Google App Store.

Functional Requirements:

An Asterix (*) indicates an optional requirement and a hashtag indicates a feature that is not yet implemented.

1. User Profile Creation and Management:

1.1. Register an account using either their email or Google account.
1.2. Login into their account using social sign-in (Google account) or their normal details.
1.3. Manage their profile.
1.4. Manage their life-list. (all the birds they've seen)

2. Interactive Map:

2.1. Display an Interactive South African map.
2.2. Show user's current location.
2.3. View bird species by pentad.
2.4. View the percentage chance that the user has of seeing a specific bird in a specific pentad.
2.5. View likely bird species near the user.
2.6. View a heatmap of the birds' locations.

3. Identify and Find Birds:

3.1. Search for a specific birds' location data.
3.2. View Pentad information of bird sightings.
3.3. View birds near you.

4. Enrich User Knowledge:

4.1. Play a bird quiz minigame*.
4.2. View all text-based information regarding a birds' location.
4.3. See what a bird looks like*.
4.4. Hear what a bird sounds like*.

5. Allow Users to Compete:

5.1. Earn experience points based on the rarity of the bird sighted to increase their level.
5.2. Earn special badges based on challenges.#
5.3. Participate in daily, weekly, and monthly challenges.

6. Manage and Process Data from Database:

6.1. Contribute to the user bird sighting database.#
6.2. Rank birds by rarity*.
6.3. Ingress of data from SABAP2.
6.4. Process data from SABAP2.

7. Administrative System and Users:

7.1. Authorise new admins.#
7.2. Manage database systems.#

Subsystems:

1. User Profile Creation and Management
2. Interactive Map
3. Identify Birds
4. Enrich User Knowledge
5. Allow Users to Compete
6. Specialised Data View for Conservationists
7. Manage and Process Data from Database
8. Administrative System and Users

Non-Functional Requirements:

1. Performance

1.1. The system should be able to handle concurrent users.

- Managed by Microsoft Azure, implements load balancing.

1.2. The system should be able to handle concurrent messages and notifications.

- Uses asynchronous message queues to handle concurrent notifications.

1.3. The system should be able to handle low network speeds.

- Offline functionality, such as downloaded maps. Data is also compressed when sent.

1.4. There should be no more than a two second response time.

- Implements caching on client- and server-side.
- Database queries are optimised from SABAP2 into the BeakPeek database.

2. Usability

2.1. The system should be easy to use.

- An extensive usability test has been performed before launch aiming to ensure all best practices are met through user feedback. This test includes users form a diverse range of ages and technical expertise.

2.2. The system should be easy to navigate.

- Consistent navigation patterns are used.
- Well known material metaphors are used for icons.

2.3. The system should be easy to understand.

- Help buttons are used to provide contextual help.
- Clear descriptive labels for UI elements are used.

2.4. The system should be easy to learn.

- A brief tutorial is given to first time users, guiding them through the application.

3. Reliability

3.1. The system should be available 20/7 (Four-hour maintenance window is acceptable between 22:00 and 02:00).

- Azure has redundant servers to ensure availability.
- Major App maintenance will be done only during the allowed window.
- Almost all futures will continue to work even if the servers are down.

3.2. Should be able to recover from crashes and failures.

- Automate Restore Processes.

- Separation of functionality ensuring one system failing does not affect the whole application.
- Robust error handling and crash reports from users are sent to administrators.

3.3. The system must survive upgrades and device migration.
- Rollback implementation to previous versions.
- Development for Android devices.
- User data is stored online with Azure.

3.4. The system should be scalable past 10 000 active users per month*.
- Azure credits will be used to cater for the use of multiple servers.
- If Azure servers are shut down the app will still function but will not receive regular data updates.

3.5. The system must function with low bandwidth.
- Most sub-systems can be used fully offline if the user has downloaded the appropriate data.

4. Maintainability

4.1. The system should be easy to debug.
- GitHub is used for version control.
- Testing ensures bugs can be found efficiently.
- Separation of concerns allows errors to be found quickly.

4.2. Be able to be updated without affecting the user experience.
- CI/CD pipelines for automated deployment.
- For app updates users will have to update their application from the app store to receive improvements but can continue to use the application without doing so.

4.3. Retrieve updated SABAP2 data without affecting users.

- New SABAP2 data automatically loaded the once a month.
- SABAP2 data ingress is updated automatically on a monthly basis.

4.4. The system should be easy to maintain.

- The system is well abstracted with a modular architecture.
- Tests are automated.

5. Security Through Azure

Our security standards are achieved through the use of Microsoft Azure and Microsoft Identity. Data integrity is ensured through the use of checksums. Audits logs are created to track administrator's activity.

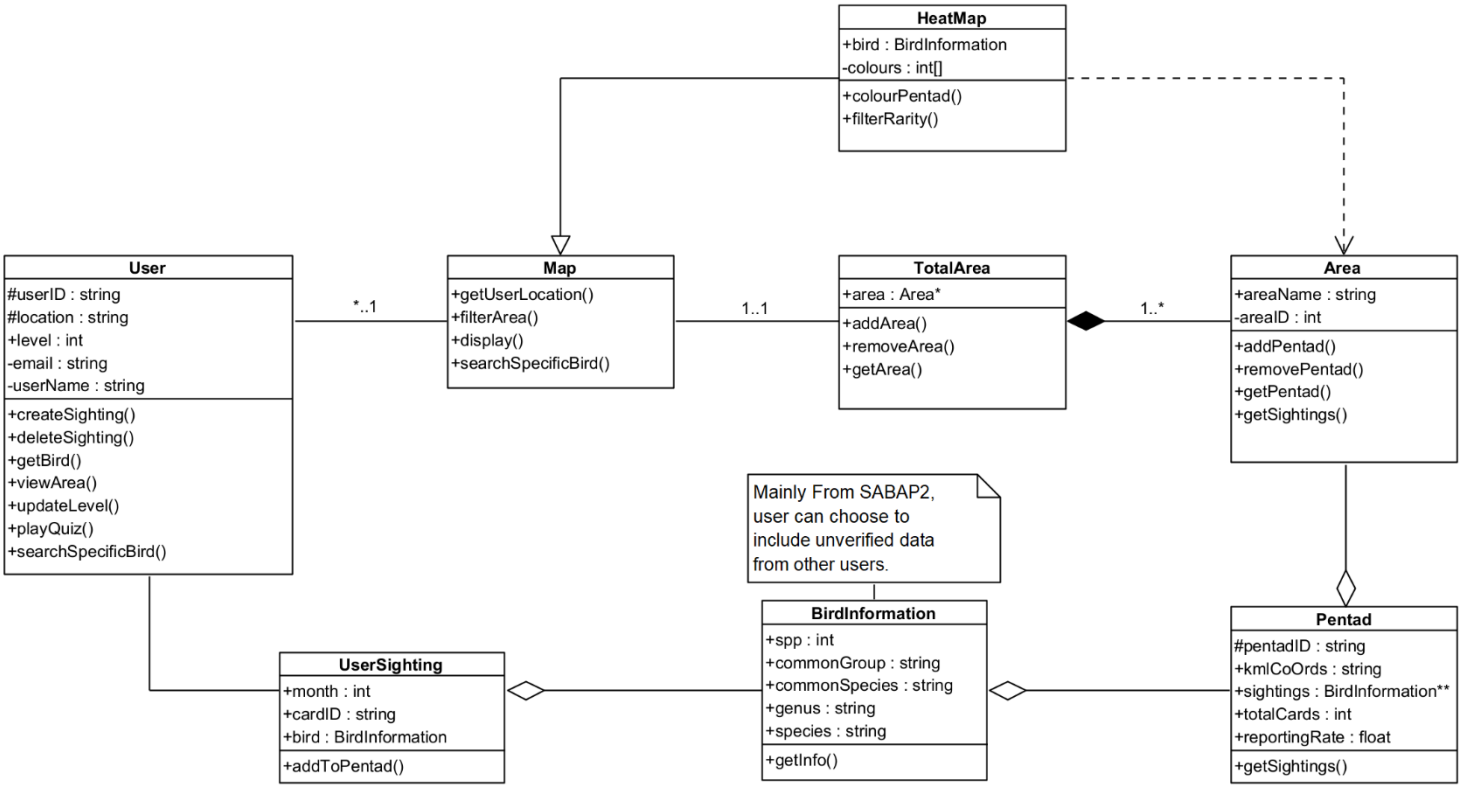
5.1. Be able to protect user data from unauthorised access.
5.2. Be able to protect user data from unauthorised modification.
5.3. Be able to protect user data from unauthorised deletion.
5.4. Be able to protect user data from unauthorised disclosure.

6. Efficiency

6.1. The system should not drain more than 5% battery per hour.

- Idle processing is optimised.
- Most processing is outsourced to the server.
- All data is downloaded to the phone with the application and only updated once a month.

UML Class Diagram:



Use Cases:

1. Management for systems personal to each user.
    a. Achievements and Experience
    b. Profile and User Settings
    c. Life List

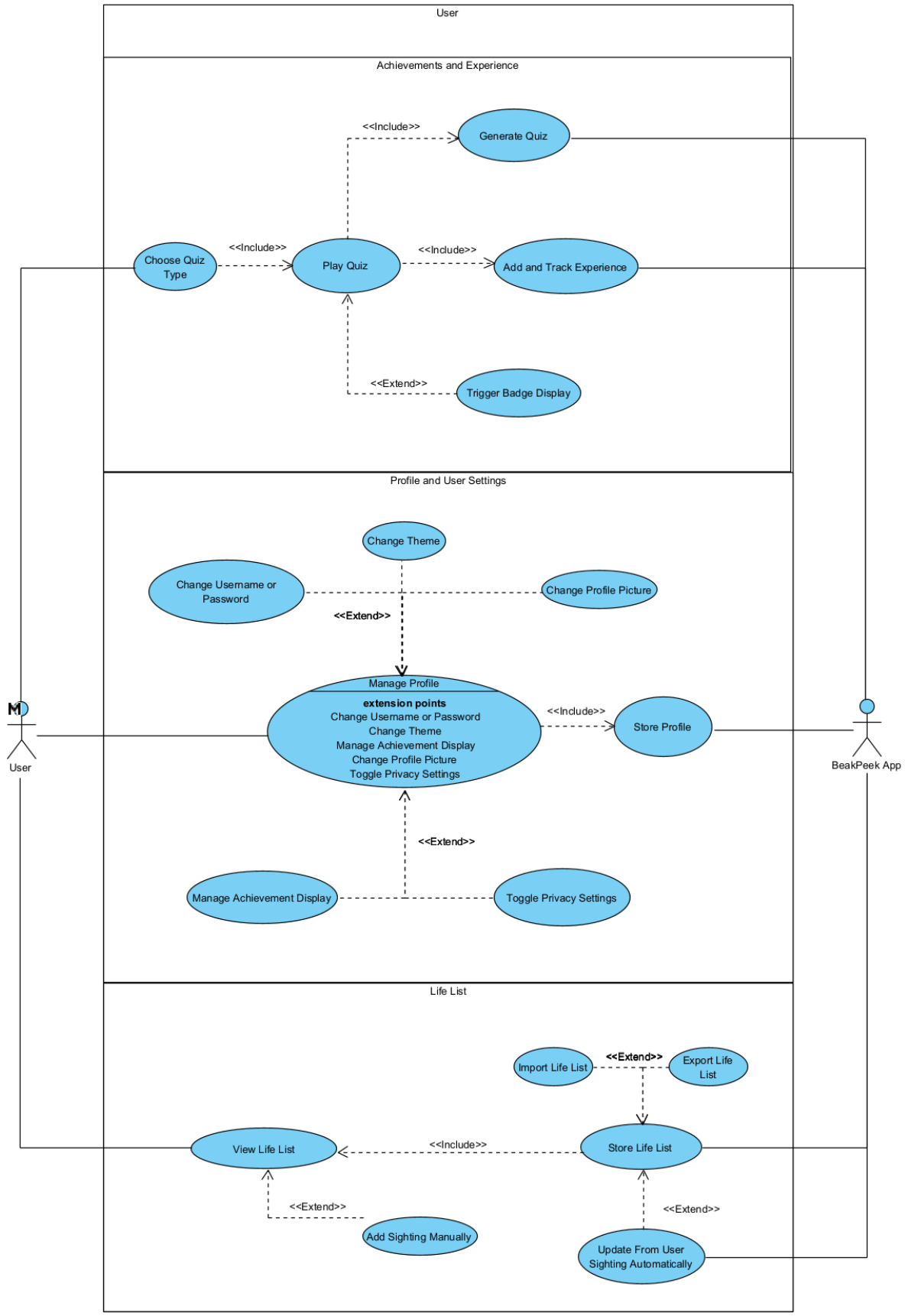
2. Integrated Map.
    a. Map
    b. View Area Birds.
    c. Heat Map.

3. User Login / Sign-up.
    a. User Sign Up
    b. User Login
    c. Guest Login

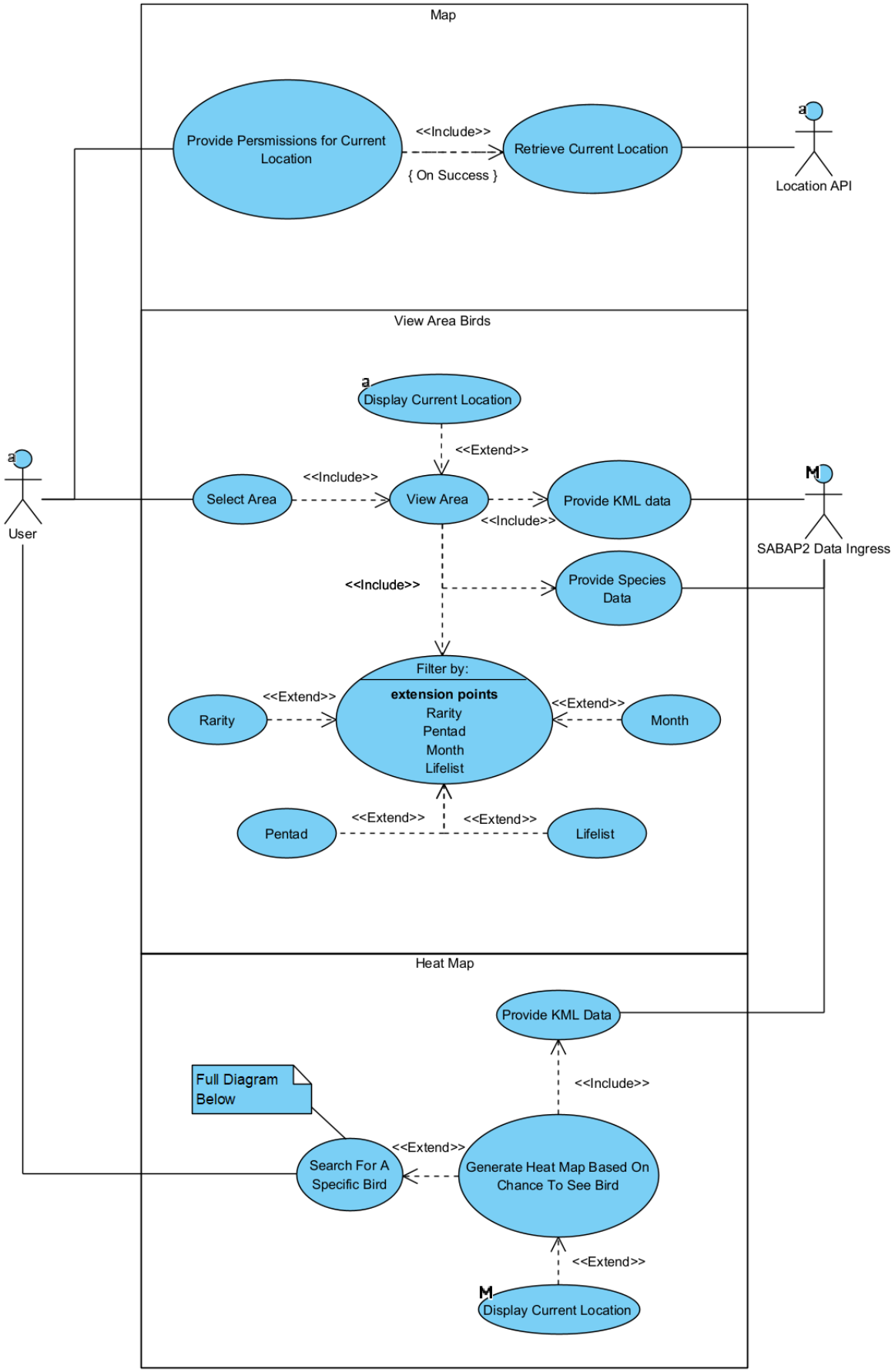
4. Search for a specific bird. (Birder)

Use Case Diagram: BeakPeek System

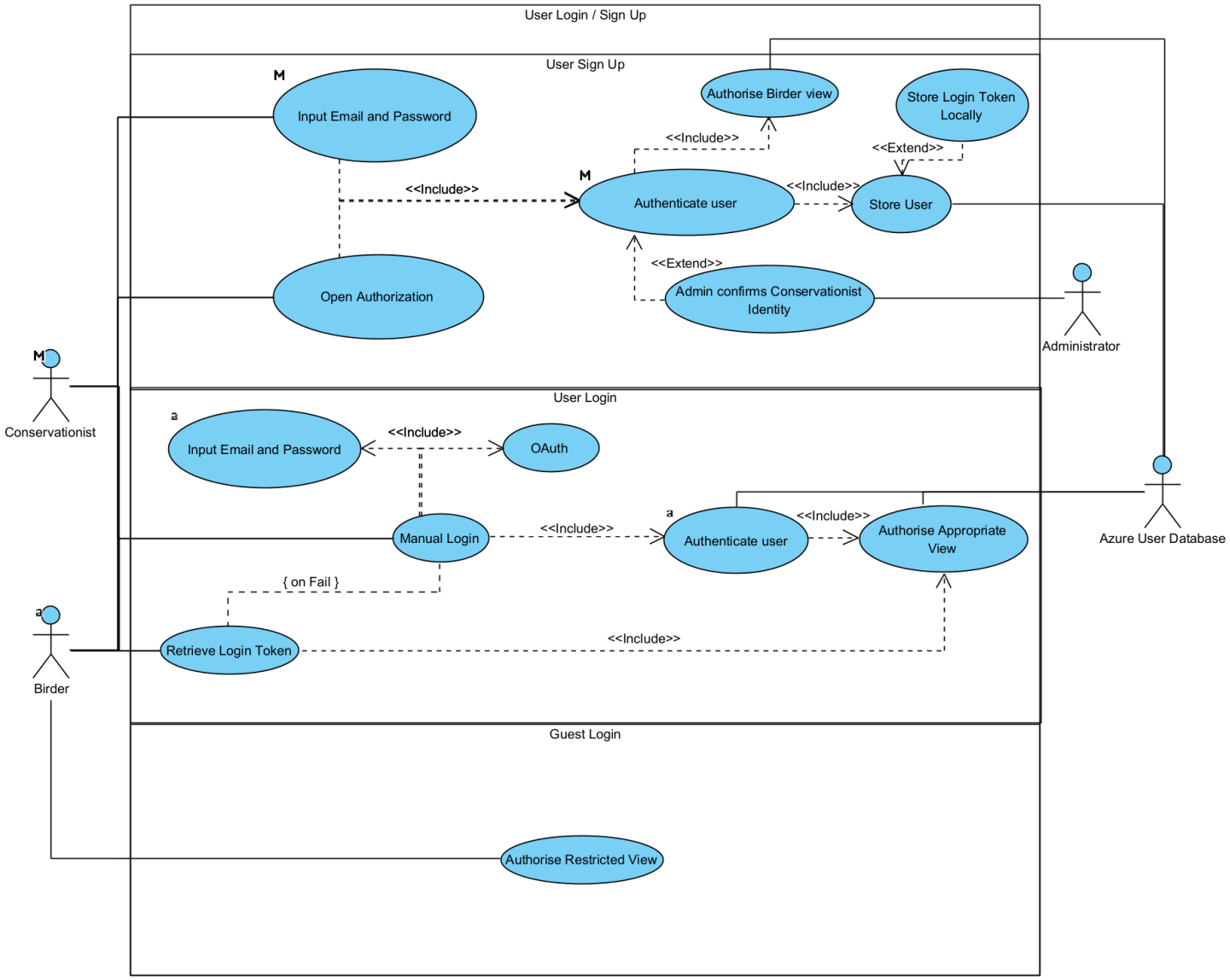
1. Personal to User:



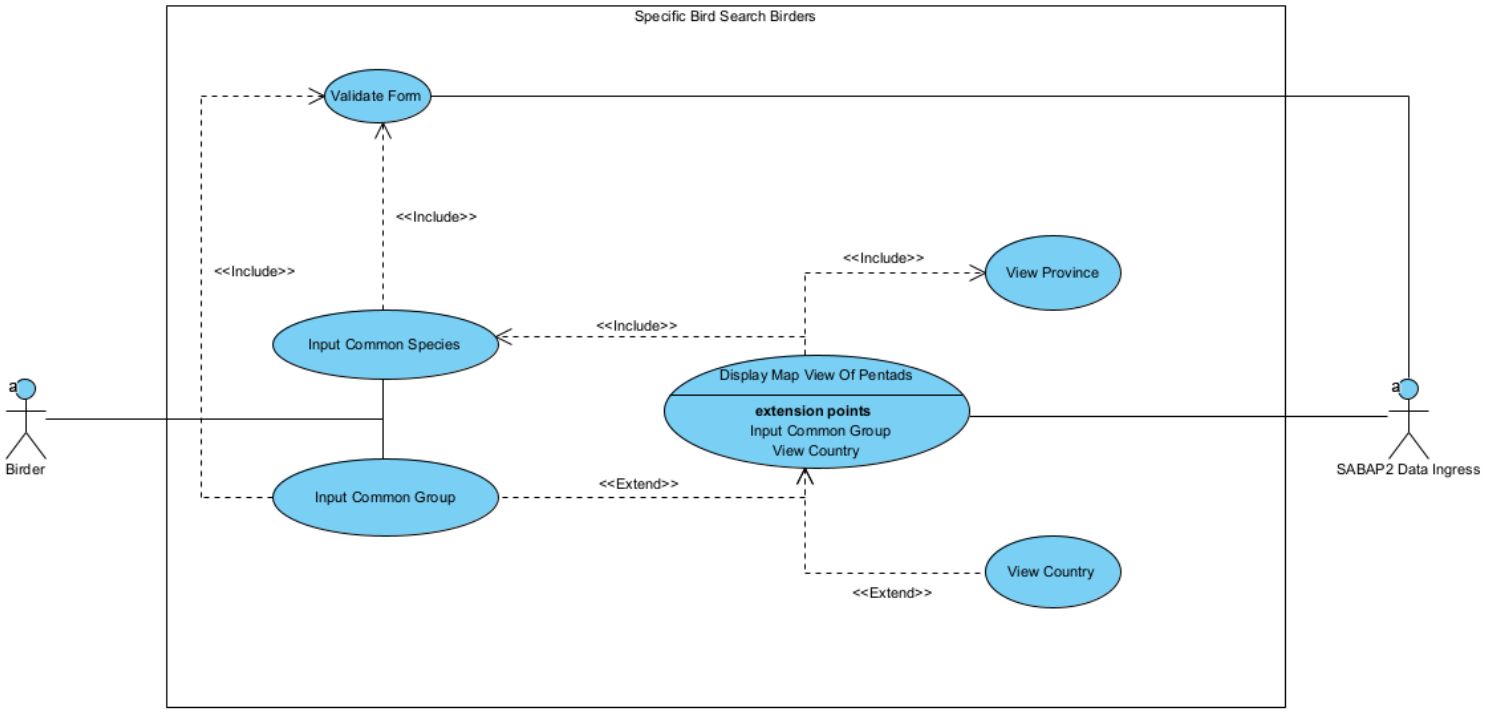
2. Integrated Map:



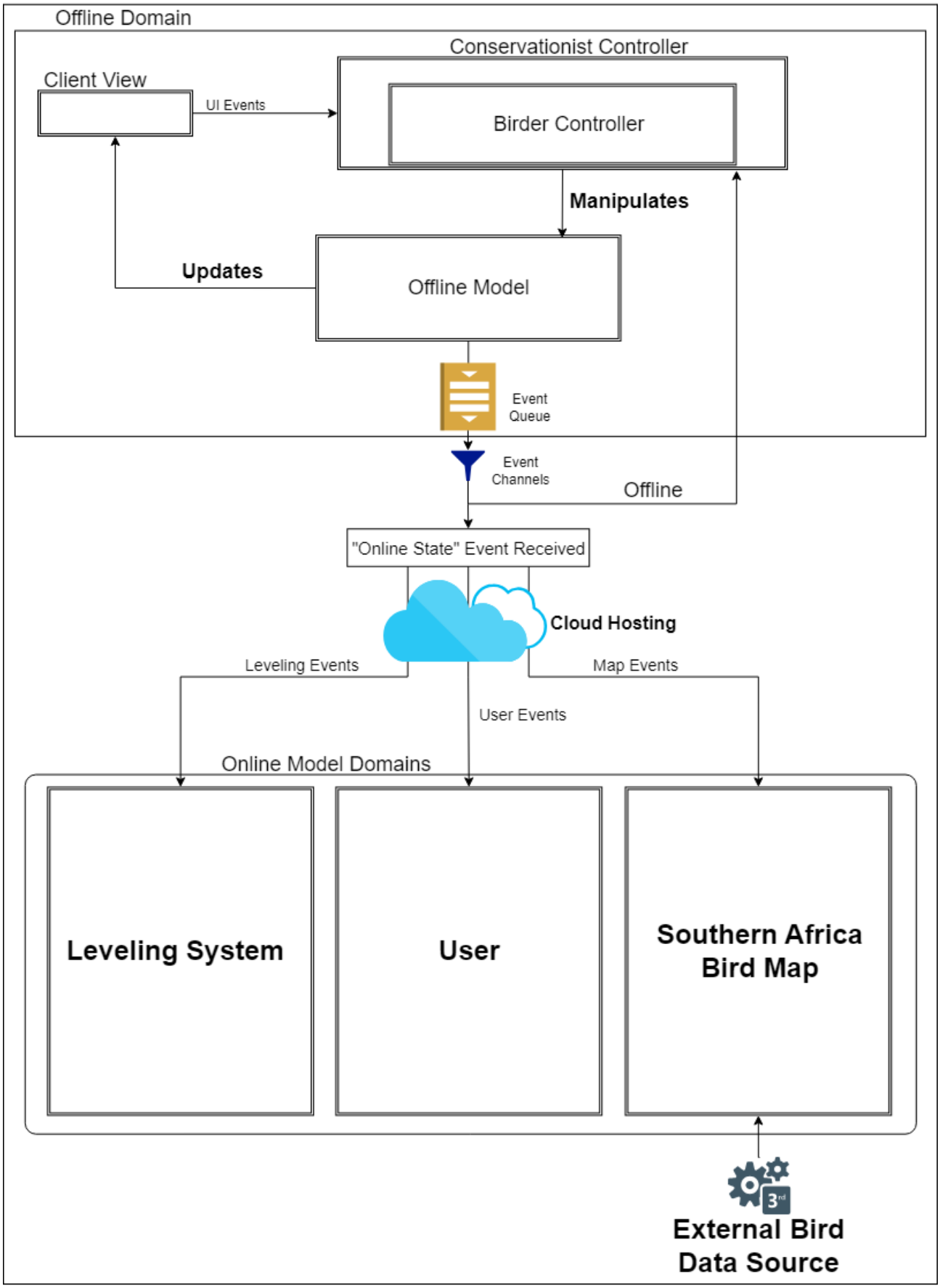
3. User Login / Sign-Up:



4. Specific Bird Search for Birders:



BeakPeek Architectural Diagram



Architectural Diagram Reasoning

For our architectural design we chose to combine a Model View Controller (MVC) with an Event Driven Architecture (EDA) and a Domain Driven Design (DDD). Our decision was based on the following reasoning:

MVC – We considered MVVM and MVC for our offline model yet decided on MVC as we prioritise useability above testing ease, and we believe MVVM would have been overkill for our simple UI. Databinding is easier in a MVVM model, however, MVC has more existing features in the ASP.NET framework, these features have made our back-end development process far more efficient and reliable. The model component also provides an easy access point for our event queue to be created that links the MVC model with our Event Driven Architecture.

EDA – This was an easy choice as EDA lends itself perfectly to our quality requirements, it not only allows us to easily incorporate parallel processing by splitting up the event queue received from the offline model to the proper destination domain where our DDD can then efficiently process the request, but it works as a perfect separation between the online and offline models. Should an action need to be performed offline it is redirected to the MVC, but if the user has an internet connection the request is instead sent to server hosted by Azure where the actions can be completed by more powerful hardware. Events that always require an internet connection can remain in the queue until the user connects to the internet again at which point they are automatically executed. This greatly boosts the applications performance and allows for real time response from the system.

Furthermore, the heterogeneous pathways improve our fault tolerance as should a system in our online model go down it can be attempted to be performed offline and the applications other systems will still be reachable, and it avoids continuous polling.

The use of EDA does lead to event duplication in our DDD but we believe the pro’s greatly outweigh this.

DDD – By using a DDD we mainly aim to satisfy our maintainability and reliability quality requirements. With clear domain borders separating back-end components testing and integration of new components becomes easier. Should one domain’s system fatally error our other domains would be able to keep functioning thus increasing the reliability of the application.

Our industry mentor is also considering turning the application into a white-collar application. By using a DDD it allows for a more flexible application where certain domains can be easily swapped out by simply connecting to the correct event stream from the EDA with the new system. This flexibility greatly increases the applications maintainability and lends itself to more organised code.

Service Contract

Provided Software

The client shall provide access to the Azure framework required for the software that Millennium will develop. Millennium will produce a working, user-friendly application that can show the pentad location of where South African birds can be located and what the percentage chance is of location each bird listed in that pentad. This data will be sourced from SABAP2 and its accuracy and functionality relies on the continued maintenance of SABAP2.

Technology Stack

The mobile application will be developed using Microsoft Azure for user management and the storage of bird data. The Dot Net framework will be used to communicate with the bird information database on Azure. Flutter will be used for all front-end development and communicated directly to Azure for user authentication and authorisation.

Project Management

For the duration of the project an agile methodology will be adopted. A weekly meeting will be held with the industry mentor and university mentor to provide progress reports and sprint reviews; this aims to ensure efficient development of the mobile application that will not only meet but surpass the clients' requirements. Team members will meet every weekday briefly to discuss progress and any problems that may have been encountered and to implement changes to the sprint plan should any be necessary.

Security

Azure implements form validation as developed by Microsoft. Users will not be able to access SABAP2 data directly: through the MVC model users will always be separated from the main database systems by the controller. User bird sightings are separated into their own database that will be indicated as untrustworthy and users will have the option to include it in the data they view.

Taking Laws into Consideration

The mobile application will comply with South African regulations, specifically provision will be made to comply to laws such as the POPI Act; ensuring that user data is secured. A user agreement policy allowing BeakPeek to store their data will be made and user data will not be made publicly available unless specifically stated that it will be (for example usernames).

Reasoning for Technology Choices

Flutter Framework

We chose Flutter for its robust cross-platform capabilities and extensive library of packages and integration tools. Unlike competitors such as React Native and Angular Ionic, which are built on top of web development frameworks, Flutter is purpose-built for cross-platform development. This results in superior performance, as reflected in benchmarks favouring Flutter.

Microsoft Azure

Microsoft Azure was selected due to its comprehensive range of services and integrations, particularly its seamless integration with the .NET framework. This synergy is facilitated by both being developed and maintained by Microsoft. Azure offers advantages over AWS in areas such as .NET framework integration and services like Azure Active Directory B2C (Microsoft Entra Identity), which is invaluable for user management.

.Net ASP NET Core

We opted for .NET ASP.NET Core because it facilitates rapid development of Web APIs and provides excellent integration with databases via the .NET Entity Framework. This ensures consistency in data types between the frontend and database, streamlining our data pipeline. It was chosen over alternatives like FastAPI due to its strong typing and better compatibility with our technology stack.

Microsoft SQL Server

Microsoft SQL Server was selected for its seamless integration with .NET and Microsoft Azure. Its stable environment is well-suited for handling large datasets, and its lack of extraneous features aligns well with our use of .NET and other tools, avoiding the complications introduced by additional features found in databases like PostgreSQL.

GitHub Actions

We chose GitHub Actions for its extensible YAML-based workflow configurations, which simplify the creation and management of workflows for various tools and automations. Its built-in nature within GitHub and extensive support for numerous tools, including Microsoft Azure, make it a superior choice over other test automation tools.