



# LoveSync

Where hearts beat as one.

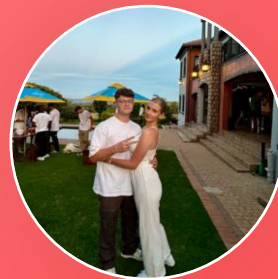
## Project Proposal



Iné Smith  
221076



Ruan Klopper  
231280



Jarryd Carelse  
221267



Wolf Botha  
21100255



## Introduction

# What is LoveSync?

Infie Inc. is a startup dedicated to revolutionising the dating landscape through innovative technology and exceptional user experiences.

At Infie, the mission is to provide clients with unforgettable connections and experiences that transcend the ordinary, offering unique opportunities to explore the boundless possibilities of human connections.

Infie is embarking on a new project to develop a cutting-edge dating app. The objective of this new dating app project is to create a platform that seamlessly connects individuals in meaningful and authentic ways. The app will serve as a space for people to discover, connect, and forge lasting relationships in an engaging and user-friendly environment.



Infie's Vision for a Cutting-edge Dating App

# Transforming the Dating Landscape with Technology



Infie is a forward-thinking startup committed to revolutionising the dating industry through innovative technology.

## Our Mission:

To create unforgettable connections, transcending the ordinary to explore the boundless possibilities of human interaction.

## Project Overview:

We aim to develop a cutting-edge dating app that seamlessly connects individuals in meaningful ways, fostering authentic relationships and engaging user experiences.

## Purpose:

This proposal outlines our comprehensive strategies for object mapping, programming language selection, framework integration, and API design to ensure the success of our visionary project.

Join us as we embark on this exciting journey to redefine the dating experience, leveraging cutting-edge technology to bring people closer together.



# Object Mapping Strategies

Object mapping refers to the process of establishing relationships between objects in a software application and the corresponding data in a database or other storage systems. In our app, object mapping promotes clean and maintainable code by providing a clear separation between the application's business logic and the data access layer. This separation makes it easier to update or modify either the application code or the database schema without affecting the other.

**Productivity:** By abstracting away the complexities of database interactions, object mapping frameworks can streamline development efforts, allowing developers to focus more on implementing business logic rather than dealing with low-level data access operations.

## Proposed Strategies for Efficient Implementation:

1. Choose the Right Object-Relational Mapping (ORM) Framework
2. Optimise Query Generation
3. Mapping Configuration
4. Batch Operations
5. Optimistic Locking
6. Database Indexing



By implementing these strategies, developers can ensure efficient object mapping in their application architecture, leading to improved performance, maintainability, and scalability of the overall system.



# Optimising Data Flow with Effective Object Mapping



The Role of Object Mapping: Translating data between the frontend and backend, crucial for a dynamic and responsive app like Infie's dating platform.

Key Entities & Data Handling: JavaScript objects and JSON will be used for flexible, schema-less data representation, facilitating easy manipulation and storage.

State Management in React / React Native: Highlighting state management libraries like Redux or Context API to manage and map application state globally, ensuring a seamless flow of data across components.

Immutable Data Patterns: Using libraries like Immutable.js can enhance performance (especially in complex applications) by preventing unwanted side-effects and optimising component re-renders.

Data Fetching and Caching: Using Apollo Client for GraphQL or React Query for REST APIs, minimising unnecessary network requests and enhancing user experience.

With careful selection of state management and data handling techniques, we ensure that Infie's app maintains a robust, efficient, and scalable data architecture, ready to handle the dynamic interactions of a modern dating platform.



Selection of

# Programming Language

## Our options for programming languages



### JavaScript

*Pros:* Easy to learn, versatile (web, mobile, backend), good for beginners.

*Cons:* Security risks, debugging challenges, browser inconsistencies.



### Swift

*Pros:* Excellent performance, ideal for native iOS apps, large community.

*Cons:* Limited cross-platform capabilities, steeper learning curve.

### Kotlin

*Pros:* Primary language for Android, concise syntax, interoperable with Java.

*Cons:* Limited cross-platform capabilities, smaller community compared to Java.



### Dart

*Pros:* Cross-platform with Flutter, hot reload feature, object-oriented.

*Cons:* Newer language, learning curve for Flutter, smaller community.



Selection of

# Programming Language



## Web and Mobile Champion:

With frameworks like React Native and Ionic, JavaScript allows us to build a single codebase that runs on everything – web browsers and mobile devices. This translates to reduced development time and maintenance costs – a win-win!



## A Community of Millions:

JavaScript has a vast and active developer community, offering a treasure trove of libraries, frameworks, and resources. Pre-built solutions for common functionalities are readily available, saving us time and effort.



## Real-time Ready:

JavaScript excels in real-time applications, which is perfect for Lovesync. Libraries like Socket.IO or websockets enable users to see updates and changes instantaneously. Imagine the joy of a real-time connection!



## Popularity Pays Off:

Being one of the most popular languages globally, JavaScript has a wealth of learning materials and a vast pool of experienced developers. Finding talent and troubleshooting issues becomes much easier.



Selection of

# Programming **Language**



## **Web Browsers:**

Imagine Lovesync seamlessly running on any web browser, accessible from any device with an internet connection. JavaScript makes this a reality, reaching a vast user base without needing separate web



**Mobile Apps:** Lovesync's magic can extend to smartphones and tablets. Powerful JavaScript frameworks like React Native and Ionic allow us to build native-looking mobile apps for both Android and iOS using the same JavaScript codebase. This translates to significant savings in development time and maintenance



## **The Future is Open:**

JavaScript's versatility doesn't stop there. As technologies evolve, JavaScript's reach might extend to other platforms, keeping Lovesync adaptable and future-proof.



Selection of

# Programming Language

## Why does this **matter**

**Wider User Reach:** Lovesync becomes accessible to a much larger audience, regardless of their device or operating

**Faster Development:** Building a single codebase saves time and resources compared to developing separate apps for each

**Reduced Maintenance:** Updates and bug fixes only need to be implemented once in the JavaScript code, simplifying maintenance across platforms.

Selection & Rationale of

## Potential Frameworks

## And Tools.

Evaluated frameworks for state management, navigation, UI components, data fetching, and testing.



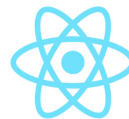
JavaScript



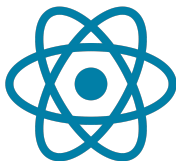
Axios



kubernetes



React Native



React



Redux



React Router





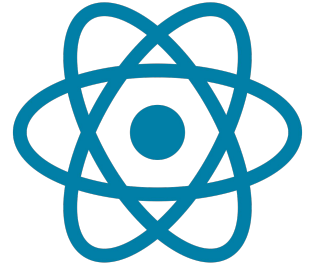
## Selection Process

- Conducted thorough research, reviewed documentation, and explored real-world use cases.
- Considered factors like community support, ease of integration, scalability, and alignment with LoveSync's requirements.
- Leveraged expertise within the team and considered factors like learning curve and long-term support.



## 1. Redux for State Management:

- Centralised state management facilitates predictable data flow and consistent state across LoveSync components.
- Ideal for handling complex data interactions and real-time updates within the dynamic LoveSync environment.



## 2. React for User Interface:

- Component-based architecture and virtual DOM enable dynamic and interactive UI development.
- Modular approach ensures efficient rendering and seamless integration with other frameworks and libraries.



### 3. JavaScript for Frontend Development:

- Widely adopted language with vast community support, ensuring compatibility and versatility.
- Enables rapid development and ensures LoveSync's frontend remains adaptable to evolving requirements.



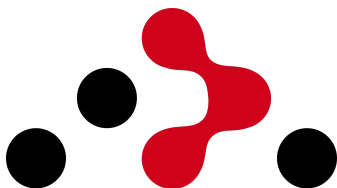
### 4. Axios for Data Fetching:

- Reliable HTTP client library simplifies asynchronous data fetching from external APIs.
- Streamlines handling of HTTP requests and responses, enhancing LoveSync's data retrieval capabilities.



## 5. React Router for Navigation:

- Declarative routing simplifies navigation between views and URLs in LoveSync's web application.
- Supports dynamic routing features essential for seamless user navigation and interaction.



# kubernetes

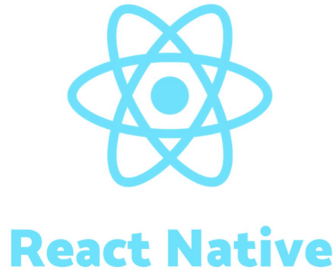
## 6. Kubernetes for Container Orchestration:

- Scalable and resilient container orchestration ensures efficient management of LoveSync's microservices architecture.
- Enables automated scaling and updates, optimising resource utilisation and application performance.



## 7. React Native for Cross-Platform Mobile Development:

- Facilitates development of mobile apps for iOS and Android using a shared codebase.
- Provides native-like performance and access to platform-specific features, ensuring a consistent user experience.



## 8. Docker for Containerisation:

- Lightweight containers offer portability and consistency in LoveSync's deployment environment.
- Simplifies development, testing, and deployment processes, enhancing efficiency and reliability.





# API Endpoints

## 1. Architecture Overview:

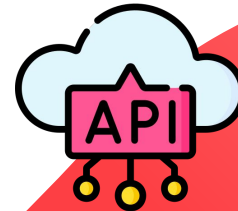
- LoveSync's API architecture serves as the bridge between the frontend (built with React and JavaScript) and the backend, facilitating communication and data exchange between the two layers.

## 2. Endpoints Specification:

- LoveSync's API defines specific endpoints, each serving a distinct purpose or functionality within the application. These endpoints allow the frontend to interact with the backend by making HTTP requests and receiving responses.

## 3. Protocols and Data Exchange Mechanisms:

- LoveSync's API communicates using HTTP(S) protocol, enabling the frontend to send requests to the backend servers and receive responses. Data exchanged between the frontend and backend is primarily in JSON format, ensuring compatibility and ease of parsing.



#### 4. Authentication Endpoints:

- LoveSync's API includes endpoints for user authentication, such as `/api/authenticate` and `/api/logout`. These endpoints handle user login, authentication, and logout processes, ensuring secure access to the application's features.



#### 5. User Profile Endpoints:

- LoveSync's API provides endpoints for managing user profiles, allowing the frontend to retrieve and update user profile information. Examples include `/api/user/profile` for retrieving profiles and `/api/user/profile/update` for updating them.



## 6. Matchmaking Endpoints:

- LoveSync's API offers endpoints for matchmaking functionality, enabling the frontend to retrieve potential matches for users based on their preferences and interact with these matches. Examples include `/api/matches` for retrieving matches and `/api/matches/like` for liking potential matches.



## 7. Messaging Endpoints:

- LoveSync's API includes endpoints for messaging between users, allowing the frontend to send and receive messages. Examples include `/api/messages` for retrieving messages and `/api/messages/send` for sending messages.

## 8. Error Handling and Status Codes:

- LoveSync's API follows standard HTTP status codes to indicate the outcome of requests made by the frontend. Detailed error messages in JSON format are provided in case of errors, helping the frontend developers understand and handle errors effectively.





LoveSync's API architecture provides a structured and organised approach for frontend-backend communication, enabling seamless interaction between the frontend (built with React and JavaScript) and the backend. It offers endpoints for authentication, user profile management, matchmaking, messaging, and robust error handling to ensure a smooth user experience.





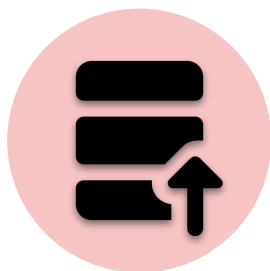
## Technical Details

# Database Schema Design I

The blueprint of how data will be organised in the database.

**Why?** The LoveSync app's performance and user experience will depend greatly on a well-designed database schema. If the data is stored in a organised and well-planned way, it will be faster to retrieve / update data, making the dating app more smooth and responsive.

## Choice between:



### SQL database

Traditional databases that use SQL (structured query language) for creating, managing & manipulating data. Very structured and well-organised, and are great for complex queries.



### NoSQL database

More flexible and scalable database, able to handle a large variety of data types - even unstructured data. No fixed schema is required, and is great for scalability and use across multiple servers.



# Database Schema Design I

The blueprint of how data will be organised in the database.

**Why?** The LoveSync app's performance and user experience will depend greatly on a well-designed database schema. If the data is stored in a organised and well-planned way, it will be faster to retrieve / update data, making the dating app more smooth and responsive.



Choice between:

**NoSQL Database** due to being able to easily handle different data types (like text, images, user preferences, etc.) and it's scalability - especially important as the user base grows.


**SQL database**

Traditional databases that use SQL (structured query language) for creating, managing & manipulating data. Very structured and well-organised, and are great for complex queries.



**NoSQL database**

More flexible and scalable database, able to handle a large variety of data types - even unstructured data. No fixed schema is required, and is great for scalability and use across multiple servers.

Choice:  
 **mongoDB**



# Database Schema Design II

## Main Components of the Database Schema



### User Profiles

A record containing all the info of a user with a user ID, stored as a document for easy retrieval / updating all user's info at once.

*Examples: Name, Surname, Gender, Bio, Interests, Photos.*



### Matches & Preferences

Keep track of dating preferences and any matches they make.

References will be used to connect users' profiles to their matches & preferences - allowing for easy match finds.

*Examples: Preference of age range, location, interests, who has been matched.*



### Messages

The messaging schema will store the conversations between users, as well as metadata - very important aspect of app.

*Examples: Actual messages, photos, voicenotes, info about sender/receiver, time sent.*



### Security Measures

Sensitive info needs to be protected, and encrypted in secure algorithms. Also using secure channels like HTTPS.

*Examples: Encrypted passwords, usernames, personal info.*



## Technical Details

# Deployment Strategies I

Ensuring LoveSync is reliable, has good performance and complies with South African laws & user expectations.

### Cloud Provider

**Why?** For remote computing services (such as servers, databases, analytics). Essentially, renting powerful computers to run the LoveSync app.

### AWS - Amazon Web Services

AWS has a data centre located in Cape Town, which will reduce latency for South African users, whilst also being compliant with South Africa's data protection laws (POPI act).





Technical Details Continued

# Deployment Strategies II



## Deployment Methodology:

### CI/CD - Continuous Integration / Continuous Deployment

This is a method to constantly deliver app code to the users by introducing automation into several stages of the app development process. It reduces the cycle times of development, increases efficiency and ensures high-quality software being developed.

**CI/CD** comes down to the following concepts:

#### Continuous Integration

The automation of integrating code changes (from multiple programmers) into a single software project, after which automated builds and tests are run, finding bugs quicker, increasing efficiency and software quality.

#### Continuous Deployment

Every change that passes through all the stages of the production pipeline is released to the users with no human intervention - only a failed test will prevent it, ensuring that users always have the latest version of LoveSync.



Technical Details Continued

## Deployment Strategies III

### Containerisation



The LoveSync app will be encapsulated with its dependencies into self-contained units called “containers”. Similar but much lighter and faster than virtual machines, containers provide isolated environments for running parts of the application, allowing it to run everywhere and to be easily started/stopped/duplicated. A popular and good choice is **Docker**.

EXAMPLE: By containerising each service of the LoveSync app (authentication, messaging), we ensure that each container has its specific dependencies needs.



# kubernetes

### Orchestration

As LoveSync grows, it can become very complex to manage all the different containers on different servers. Orchestration is the automated configuration, coordination and management of computer systems and services.

**Kubernetes** is an well-known open-source platform, that will provide everything LoveSync may need.

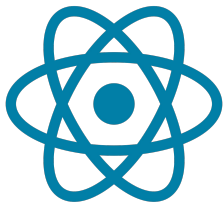
Including processes such as scheduling (where containers should run to optimise resource use), load balancing (distributes traffic across multiple instances of LoveSync), health monitoring (checking the health of containers, and replacing them if needed), scaling (increases/decreases the amount of containers based on demand).



# Concluding **Thoughts**

## Technological Foundation

LoveSync requires a modern & robust tech stack incorporating:



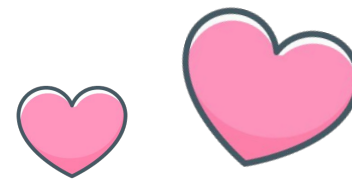
This foundation will ensure LoveSync is scalable, easy to maintain and flexible for future user needs and new developments of features & functions.

## User Experience Focus

The main focus of the LoveSync development process should be centred around its users. It's very crucial to create an engaging user experience that's seamless, reliable, easy to navigate and overall just something users will enjoy. Thus, with careful planning of object mapping, selection of the right languages for efficiency, the suitable frameworks and libraries to enhance our UI/UX, with a strategically-planned API and well-designed database, along with containerisation and orchestration, will ensure LoveSync is a successful and flourishing app.



# Next Steps



## Development & Local Testing

Begin with the development phase, focusing on the central core functionalities, such as user profiles, matchmaking algorithms and message systems.

## Marketing Campaign

Create a marketing plan that resonates with the South African market, across different social media platforms and marketing mediums.

## Monitoring & Optimisation

After launching, there should be continuous monitoring of the app's performance, bugs and user feedback. We'll use these to refine features and optimise the app.

## Community Engagement

Establish means for users to give ongoing feedback, and to create a sense of community around LoveSync. This is crucial for making the app the best it can be, and align LoveSync with what users expect.



# Thank you for listening.

**Iné Smith**

221076

**Ruan Klopper**

231280

**Jarryd Carelse**

221267

**Wolf Botha**

21100255

