
**BILKENT UNIVERSITY**

**DEPARTMENT OF COMPUTER ENGINEERING**

# Senior Design Project

*Petriam*

## Low-level Design Report

*28 February 2022*

**Supervisor:** Assoc. Prof. Dr. Özcan Öztürk

**Jury Members:** Erhan Dolak & Tağmaç Topal

**Project Members:** Osman Buğra Aydın – 21704100

Oğuzhan Angın – 21501910

Mehmet Alperen Yalçın – 21502273

Ertuğrul Aktaş – 21802801

Muhammed Doğancan Yılmazoğlu – 21801804

# 1. Introduction

Animal domestication has an important place in human history as it existed for a long period of human lifetime. Even though it first started for mainly food and clothing purposes, later this relationship grows into a more loving and caring type of relationship between humans and other animals. In our daily lives, pets play an important role as some consider their pets as their friends while some consider their pets as part of their family. Such an intimate relationship between humans and their pets causes owners to look for products and services that help them to take good care of their pets.

As technology developed over time, such developments allowed humans to travel more often for business or entertainment purposes. However, the increased number of travels creates problems for taking care of beloved pets of the pet owners. Such long-distance travel may be uncomfortable and unhealthy for the pets, also travel companies can have strict rules for this kind of pet luggage. Owners can leave them at pet hotels during their travel but these hotels can be expensive and they are not available in every country or city.

Petriam aims to solve sheltering of their pet issue for pet owners during times of need such as long-distance travels. By creating a virtual environment for users to meet each other, pet owners can arrange a shelter and caregiver for their pets whenever they need to travel to a different location. By using Petriam, pet owners can look through a list of other users that are verified pet hosts and choose one of them to look after their pet for the decided price when they travel. In this way, the pet owners can find a shelter for their pets while pet hosts can earn money by hosting other users' pets. Moreover, when a pet owner leaves their pet to a pet host, they can get in contact with the pet host using Petriam to check the status of their pet or solve issues for the pet host about specific issues with the pet.

This report includes a detailed explanation of low-level architecture and software design of the Petriam. The descriptions for the trade-offs of the application will be provided. Lastly, software design that includes packages and class interfaces will be defined explicitly.

## 1.1 Object design trade-offs

### 1.1.1 Supportability vs Performance

Petriam's goal is to increase the number of people that it has helped to find an appropriate host for their pets. Therefore, our main goal is to make our application executable by different mobile operating systems such as IOS and Android to provide

our service to more people. To achieve this goal, we used cross platform tools to implement the application. However, the performance of the applications that are developed by cross platform tools such as React, Flutter is low compared to the native applications developed with Xcode for IOS and Android Studio for Android[1].

### 1.1.2 Expandability vs Cost

Expandability refers to the ability of adapting and implementing new requirements to the upcoming or emerging needs. Since Petriam provides a service that might lead to new functionalities and can be used by any people all over the world, we used Google API which may cost more money than its alternatives like OpenLayers that is free[2]. However, using Google API allows us to implement more comprehensive functions. Also, as a NoSQL database Dynamodb could be used for a cheaper solution but there might be a problem if we try to implement new functions[3]. MongoDb is more adaptable especially for complex queries since it stores Json documents whereas Dynamodb is a key-value NoSQL database[4].

### 1.1.3 Security vs Usability

Security of the information stored in the application is a crucial task to achieve. To increase security, we used a token based approach for logging in the application. The token generated by the server will expire in a certain amount of time so that user needs to enter its credentials again[5]. Apparently, it reduces usability since the user needs to login periodically but protects the account of the user if his phone is physically taken away from him.

### 1.1.4 Reliability vs Supportability

Petriam is a mobile application that is developed by cross-platform tools. Since it is supported by more operating systems, the number of failures or system crashes might increase. We can build a reliable system ensured by verification of various testing tools but there might be some problems related to the operating systems. Native applications can be better in terms of reliability. Nevertheless, Petriam’s main goal is to achieve more users by being compatible with different operating systems.

## 1.2 Interface documentation guidelines

The definition of the classes, attributes and methods will follow a certain format that is described below.

| **class ClassName** |
| --- |
| A brief description of the class |
| **Attributes(Properties)** |
| type nameOfTheAttribute<br>● “type” can be a number, string, boolean, enum, void, null, undefined, any, never, and array |
| **Methods** |
| returnType nameOfTheMethod(parameters) → A brief description of the method unless it is trivial |

## 1.3 Engineering standards (e.g., UML and IEEE)

In this document, any type of diagrams, scenarios or visual graphs will follow the UML standards. UML allows our application to express itself globally in a detailed manner to the people since it is widely used in software engineering as a tool for system decomposition, class, sequence diagrams, etc. For the citations, this report will apply the rules of IEEE standards. With these standards, citations will be easy to read since they are frequently used, especially in the technological domain.

## 1.4 Definitions, acronyms, and abbreviations

**Petriam:** Name of the application
**API:** Application Programming Interface
**IEEE:** Institute of Electrical and Electronics Engineers
**UML:** Unified Modeling Language

# 2. Packages

## 2.1 Server

### 2.1.1 Model

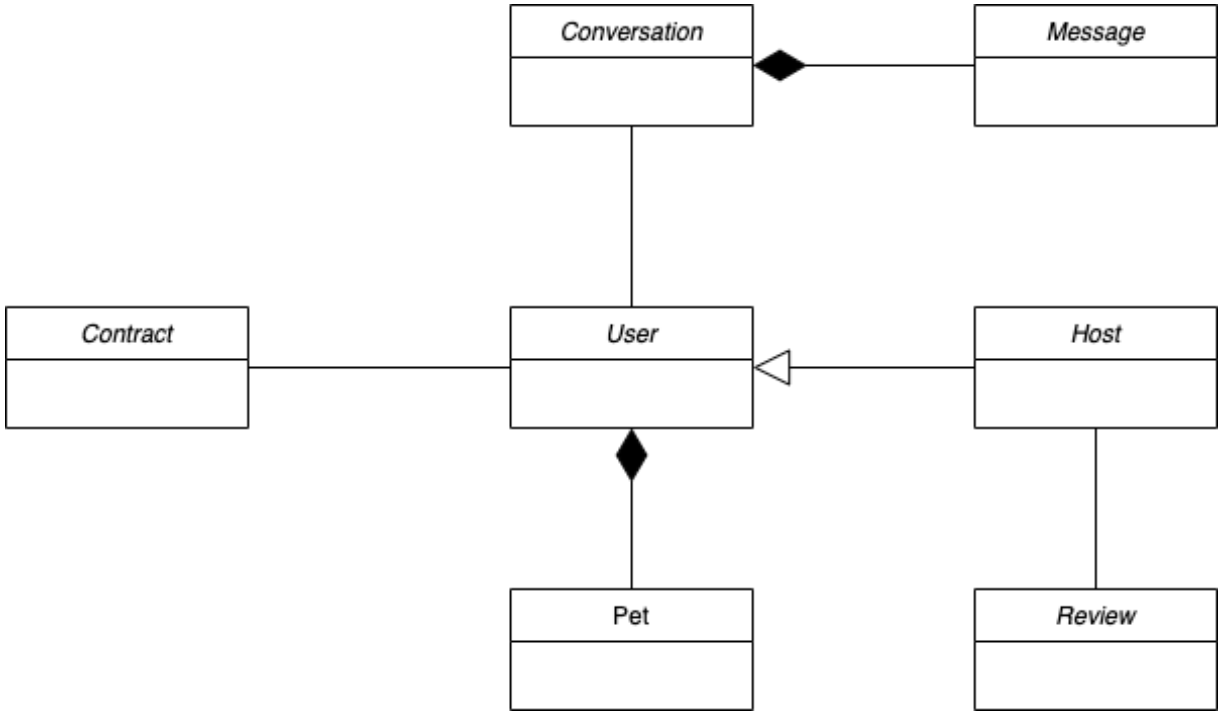


Figure: Model package

**User:** This class represents the main users of Petriam. A user created after signing up. It has some attributes such as password, name or location. Moreover, users are pet owners and want to find a host when they are not able to look after their pets for a while.

**Host:** This class inherits from the user class. Users can be a host if they fulfill some requirements like showing criminal records etc. It has some more attributes than users such as average ratings, comments, hes code.

**Pet:** This class represents the pets of Petriam users. Its attributes are owner id, name, type etc.

**Contract:** This class represents the contracts of Petriam users. When a user sends a request to a host for an agreement, a contract is created with user id, host id, pet,

starting date, ending date, price etc. There is also a status attribute. It can be accepted, rejected, completed etc.

**Conversation:** This class represents the conversations of a user. Users can send messages to hosts for some reasons such as asking some questions before making a contract or wanting some pictures of their pets when their pets are in host. Conversation class has messages and participants attributes.

**Message:** This class inherits from conversation class. A conversation has plenty of messages and a message has sender, date and content attributes.

**Review:** This class represents the reviews of hosts. After a contract is completed, the user can write some comments and give a rating out of five to the host.

### 2.1.2 Controller

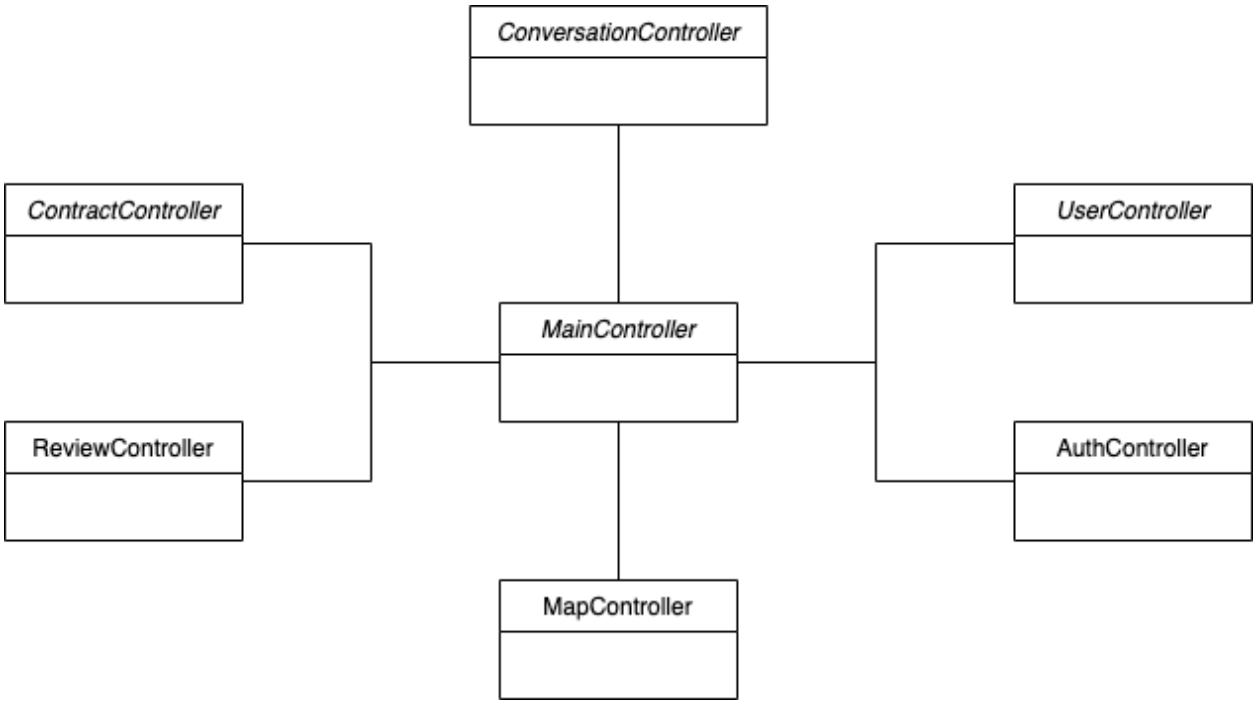


Figure: Controller package

**MainController:** This class is responsible for all other controllers of the system.

**UserController:** This class is responsible for user specific operations. For instance, updating the profile of a user and adding a new pet in the pets array of the user are done by functions of this class.

**MapController:** This class is responsible for map specific operations. It creates pinpoints of hosts on a map and users can see the points and choose a host for their pets.

**ContractController:** This class is responsible for user contract operations. As mentioned before, there are several status of a contract and this controller class changes the status of the contracts.

**ConversationController:** This class is responsible for user conversation operations. For instance, starting a conversation and sending messages are done by this controller class.

**AuthController:** This class is responsible for authentication operations such as log in and sign up

**ReviewController:** This class is responsible for review specific operations. For instance, sending a review for a host is done by this controller class.

## 2.2 Client

### 2.2.1 View

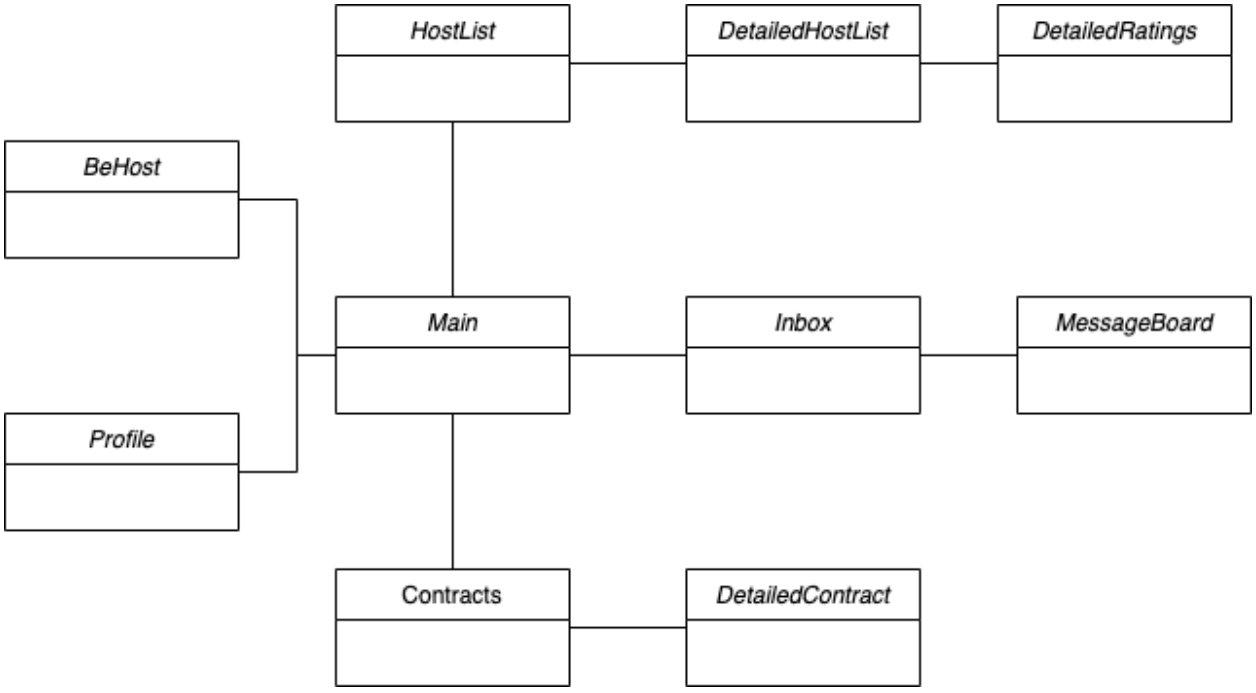


Figure: View package

**Main:** This class is the main screen of the Petriam. There is a map on the view that has pinpoints of the hosts and a search bar to search specific hosts.

**HostList:** This class displays the hosts after searching host on the main page. Users can see some information about the host such as price, address etc. Users can also use a filter bar to see hosts with more specific features.

**DetailedHostList:** This class displays a host after clicking a host on the host list page. There is detailed information about the host like reviews and users can hire a host with this class.

**DetailedRatings:** This class displays the reviews of the host. Users can have more information about this class.

**Profile:** This class displays the profile page of users.  There is some information about users like profile picture, address, pets.

**BeHost:** This class displays a form for becoming a host. Users can give some specific information such as criminal records and send requests to be a host with this class.

**Inbox:** This class displays conversations of a user with other users. There is some information about other users such as name and profile picture.

**MessageBoard:** This class displays a specific conversation of a user. After clicking a conversation on the inbox page, a conversation is shown with a host. Users can send messages with this class.

**Contracts:** This class displays contract history of a user along with their status. Hosts can approve or reject the contract request with this page. There is also a message button to start a conversation.

**DetailedContract:** This class displays a specific contract of a user. After clicking a contract on the contracts page, a contract is shown with detailed information about the contract.

# 3. Class Interfaces

## 3.1 Server

### 3.1.1 Model

| **class User** |
| --- |
| This class represents the base user. Every user is represented by this class including the host users. |
| **Attributes** |
| private String id |
| private String name |
| private String email |
| private String passwordHash |
| private String phone |
| private double[] location |
| private String profileImage |
| private Pet[] pets |
| private User[] blockedUsers |
| **Methods** |
| Getter and setter methods |

| **class Host** |
| --- |
| This class represents a host user. This class is inherited from the user class. |
| **Attributes** |
| private String tc |
| private double averageRating |
| private String[] languages |
| private String hesCode |
| private String criminalRecord |
| private String address |
| private String[] tags |
| private double price |
| private ReviewController reviewController |
| **Methods** |
| Getter and setter methods |

| **class Pet** |
| --- |
| This class represents the pet that owned by a user. |
| **Attributes** |
| private int id |
| private String name |

| |
|---|
| private String type |
| private String ownerId |
| **Methods** |
| Getter and setter methods |

| **class Review** |
|---|
| This class represents a review submitted by a user at the end of a hosting session. |
| **Attributes** |
| private Sting id |
| private Date date |
| private int rating |
| private String comment |
| **Methods** |
| Getter and setter methods |

| **class Contract** |
|---|
| This class represents the contract between the pet owner and the host. |
| **Attributes** |
| private String id |
| private Host host |

| |
|---|
| private User user |
| private String status → Pending, accepted, denied, canceled, aborted, finalized. |
| private Pet[] pets |
| private Date startDate → Hosting period starting date. |
| private Date endDate → Hosting period ending date. |
| private Date contractDate → Contract creation date. |
| private double price |
| **Methods** |
| Getter and setter methods |

| **class Conversation** |
|---|
| This class represents the message conversation between the pet owner and the host. |
| **Attributes** |
| private Message[] messages |
| private User[] participants → User and the host |
| private Message lastMessage |
| **Methods** |
| public Message[] getLastMessages(int range) → Get last x messages in the conversation. |
| Getter and setter methods |

| **class Message** |
| --- |
| This class represents a single message that has been sent from one user to another. |
| **Attributes** |
| private String id |
| private User sender |
| private Date date |
| private String content |
| **Methods** |
| public boolean isLoggedInUser() → Detects if the current user is the owner of this message. |
| public String decideColor() → Detects if the current user is the owner of this message. If true return color name x, else y. |
| public boolean isImage() → Detects if the message content is an image or not |

### 3.1.2 Controller

| **class ReviewController** |
| --- |
| This class manages all the user reviews that are submitted for a host. |
| **Attributes** |
| private Review[] reviews |
| **Methods** |
| public double getAverageRating() |

| |
|---|
| public Review[] sortByRating() |
| public Review[] sortByDate() |
| public Review[] getReviewsBetween(Date start, Date end) |
| Getter and setter methods |

| **class UserController** |
|---|
| This class manages general operations that a user can execute. |
| **Attributes** |
| None |
| **Methods** |
| public Contract getContract(User user, String contractId) |
| public Contract[] getContractHistory(User user) |
| public Host[] getHostsWithFilter(Object filter) → Get the suitable hosts for a user using a parameterized filter. |
| public void updateProfile(User user) |
| public Pet[] getHostingPets(Host host) → Get currently hosted pets. |
| public int calculateDays(Contract contract) → Calculate the total number of days that the pet will be hosted. |
| public int calculateRemainingDays(Contract contract) → Calculate the number of days remaining for a hosting session to end. |
| public void addPet(Pet pet) → Add a pet to the owned pets array of the user. |

| **class AuthController** |
| --- |
| This class manages authentication operations for a user. |
| **Attributes** |
| None |
| **Methods** |
| public boolean login(Sting email, String password) |
| public boolean logout(User user) |
| public boolean signUp(User user) |

| **class MapController** |
| --- |
| This class manages google maps operations such as pinpointing filtered hosts on the map. |
| **Attributes** |
| private Host[] hosts → Hosts that will be shown on the map. |
| **Methods** |
| public void createPinpoints() → Create pinpoints on the map for the found hosts. |

| **class ContractController** |
| --- |
| This class manages all operations on contracts such as approving, canceling, aborting a contract. |
| **Attributes** |

| None |
| --- |
| **Methods** |
| public void approveContract(String contractId) → Host accepts the contract offer. |
| public void cancelContract(String contractId) → User cancels the contract offer before host accepts it. |
| public void denyContract(String contractId) → Host denies the contract offer. |
| public void abortContract(String contractId) → Contract is aborted after being accepted with the agreement of the both parties. |
| public void finalizeContract(String contractId) → After the hosting period contract finalizes successfully. |

| **class ConversationController** |
| --- |
| This class manages the messaging operations between the pet owner and the host. |
| **Attributes** |
| None |
| **Methods** |
| public void startConversation(Host host, User user) |
| public void sendMessage(User sender, User receiver, String message) |
| private boolean isBlocked(User sender, User receiver) |

| **class MainController** |
| --- |
| This class contains and manages all other controllers. |
| **Attributes** |
| private AuthController authController |
| private ConversationController conversationController |
| private ContractController contractController |
| private MapController mapController |
| private UserController userController |
| **Methods** |
| Getter and setter methods |

## 3.2 Client

### 3.2.1 View

| **class MainView** |
| --- |
| This class represents the main screen of the application. It has a map to display found hosts. |
| **Attributes** |
| private Host[] hosts |
| **Methods** |
| private void pinpointHosts(Hosts[] hosts) |

| **class BeHostView** |
| --- |
| This class represents the host application page. A user can apply for a host account upgrade by filling required fields and submitting the form. |
| **Attributes** |
| private User currentUser |
| **Methods** |
| public void submitForm() |
| private boolean checkInputFormats() → Check if all input fields filled with correct format |

| **class ProfileView** |
| --- |
| This class represents the user profile screen. A user can inspect and edit their details on this screen. |
| **Attributes** |
| private User currentUser |
| **Methods** |
| public void makeFieldsEditable() → Make fields editable in order to let user edit profile info |
| public void saveFields() → Save edited information |
| private void populateInformationFields(User user) |
| private boolean checkFieldsValidity() → Check if field alterations are valid or not |

| **class HostListView** |
| --- |
| This class represents the listing page for the found hosts. |
| **Attributes** |
| private Host[] hosts |
| **Methods** |
| public goToHostDetail(Host host) |
| private listFoundHosts(Hosts[] hosts) |

| **class DetailedHostListView** |
| --- |
| This class represents the page that has detailed information about the selected host. |
| **Attributes** |
| private Host host |
| **Methods** |
| public void startConversation(Host host) |
| public void startContract(Host host) |
| private void populateFields(Host host) |

| **class ContractsView** |
| --- |
| This class represents the contracts listing page that lists the contracts belonging to the current user. |
| **Attributes** |

| |
|---|
| private Contract[] contracts |
| **Methods** |
| public goToContractDetail(Contract contract) |
| private void listContracts(Contract[] contracts) |

| **class DetailedContractView** |
|---|
| This class represents the page that has detailed information about the selected contract. |
| **Attributes** |
| private Contract contract |
| **Methods** |
| private void populateFields(Contract contract) |

| **class DetailedRatingsView** |
|---|
| This class represents the page that has detailed rating information about the selected host. |
| **Attributes** |
| private Reviews[] reviews |
| **Methods** |
| public void filterReviews(Reviews[] reviews) |
| public void sortReviews(Reviews[] reviews) |

| |
|---|
| private void listAllReviews(Reviews[] reviews) |

| **class InboxView** |
|---|
| This class represents the page that lists the started conversations with the other users. |
| **Attributes** |
| private Conversation[] conversations |
| **Methods** |
| private void listConversations(Conversation[] conversations) |
| public void selectConversation(Conversation conversation) |

| **class MessageBoardView** |
|---|
| This class represents the conversation page that has the messages sent by/to the current user for the selected conversation. |
| **Attributes** |
| private Conversation conversation |
| private Message[] messages |
| private User otherUser → The user that is conversing with the current user. |
| **Methods** |
| public void sendMessage(Message message) |
| private void showReceiverInfo(User otherUser) |

```
private void showMessages(Messages[] messages)
```

# 4. References

[1] A. Klubnikin, “Cross-platform vs Native Mobile app development: Choosing the Right Development Tools for your...,” Medium, 14-Apr-2021. [Online]. Available: https://andrei-klubnikin.medium.com/cross-platform-vs-native-mobile-app-development-choosing-the-right-dev-tools-for-your-app-project-47d0abafee81. [Accessed: 24-Feb-2022].

[2] T. Bush, “5 powerful alternatives to google maps API: Nordic Apis |,” Nordic APIs, 09-Mar-2020. [Online]. Available: https://nordicapis.com/5-powerful-alternatives-to-google-maps-api/. [Accessed: 24-Feb-2022].

[3] AWS Pricing Calculator. [Online]. Available: https://calculator.aws/#/. [Accessed: 24-Feb-2022].

[4] “What is nosql? NoSQL databases explained,” MongoDB. [Online]. Available: https://www.mongodb.com/nosql-explained. [Accessed: 24-Feb-2022].

[5] auth0.com, “JSON web tokens introduction,” JSON Web Token Introduction. [Online]. Available: https://jwt.io/introduction. [Accessed: 24-Feb-2022].