KFC Single Sign-On Portal

Automated Application Synchronization

Design Document

CECS 491B, Sec 11

March 13, 2019

Krystal Leon
013986607

Revision History

Date	Version	Description
3/11/19	1.0	First draft.

Table of Contents

1. Introduction

This document is made in order to understand the low level design for the automated application synchronization feature of the KFC Single-Sign-On application portal. This feature includes the functionalities for individual applications to register to the portal, publish themselves into the portal, request a new API key, and delete themselves from the portal. Diagrams are included to show the flow of the functionalities.

2. Application Services

The following diagrams represent the flow of calling the operations to create, update, read and delete an Application entry in the data store.

2.1 Create an Application

When the Create service is called, the Database Context and Application are passed to the Application Repository. The repository will first check if the application exists in the database. If the application does not already exist, then a new record of the application will be created and return back to the call. (Figure 2.1.1) If the application does already exist, then the application record will not be created and null will be returned instead. (Figure 2.1.2)

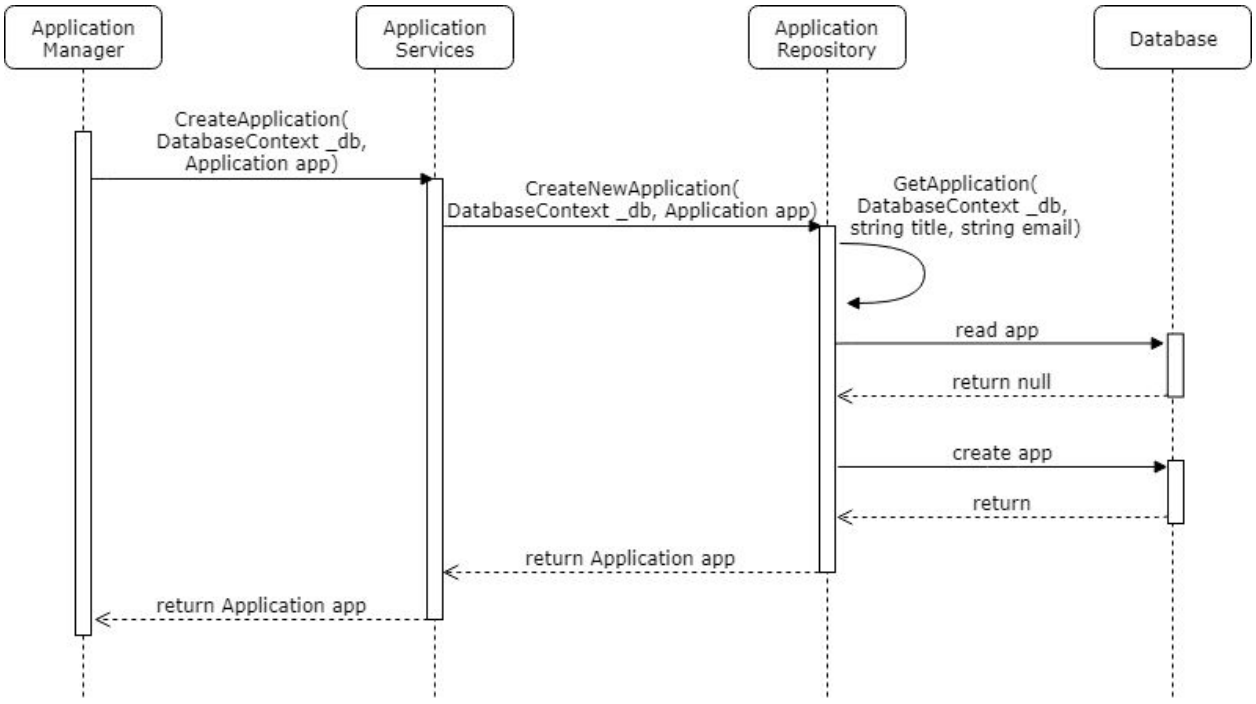

Figure 2.1.1. Creating a Non-Existing Application

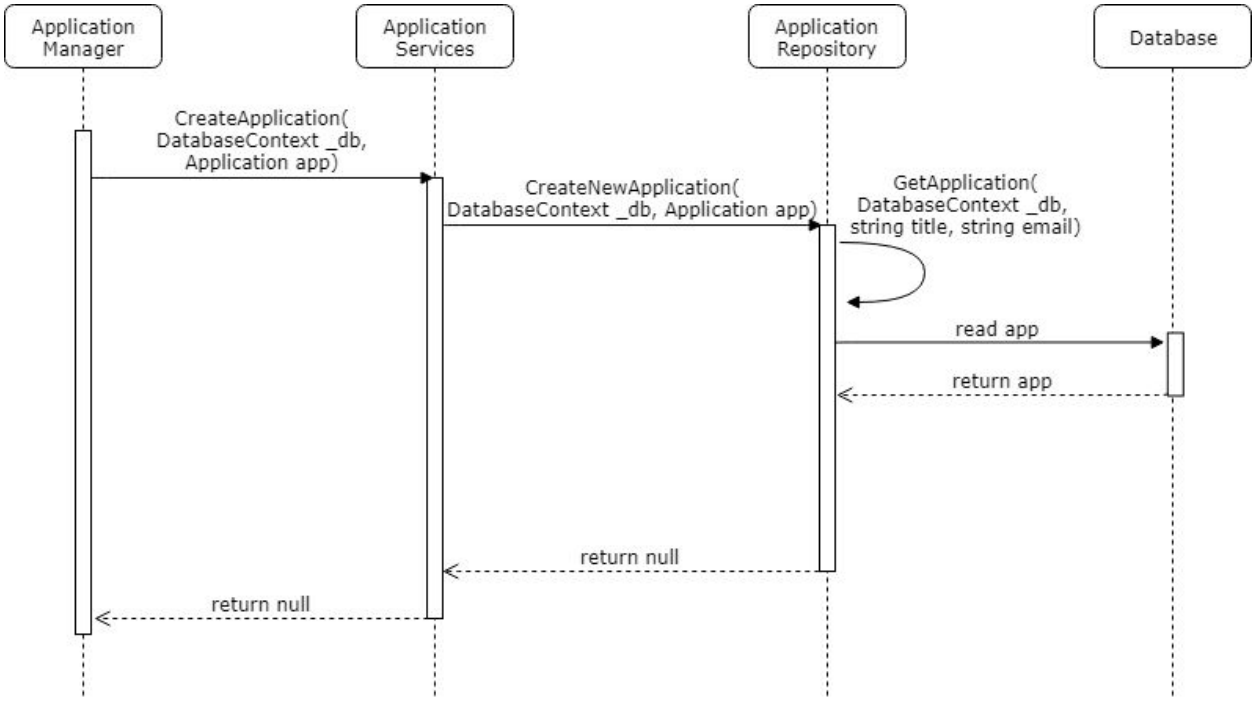

Figure 2.1.2 Creating an Existing Application

2.2 Read an Application

2.2.1 Read By Id

When the Read service is called, the Database Context and Application Id are passed to the Application Repository. The repository will attempt to retrieve the application. (Figure 2.2.1) If the application is found, it will be returned. If it is not found, null will be returned instead. (Figure 2.2.2)

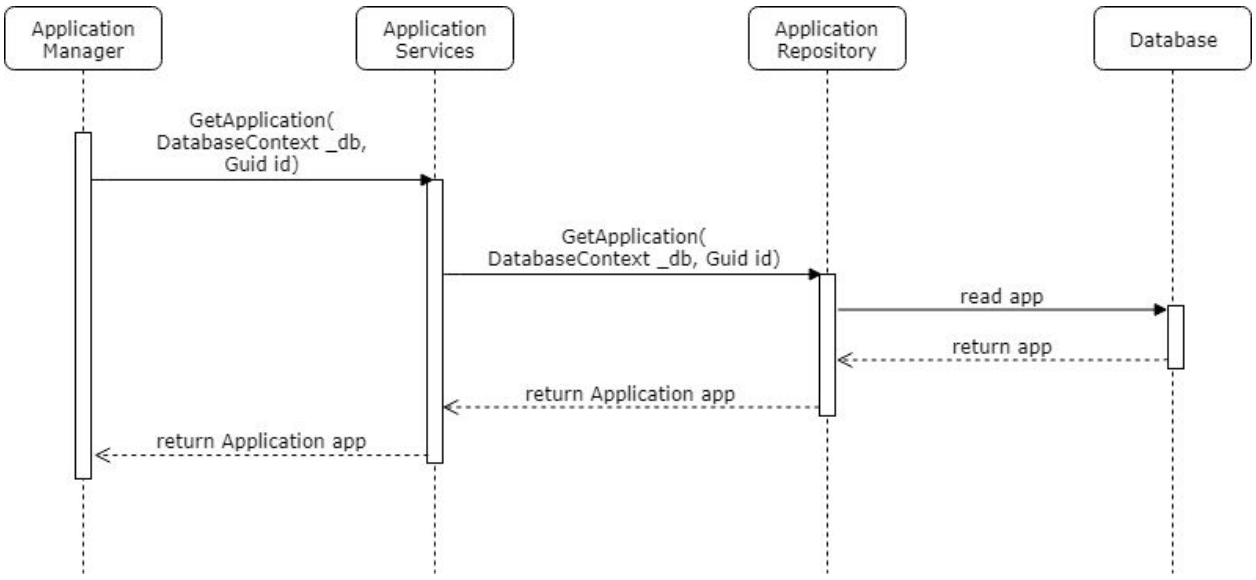

Figure 2.2.1 Reading an Existing Application

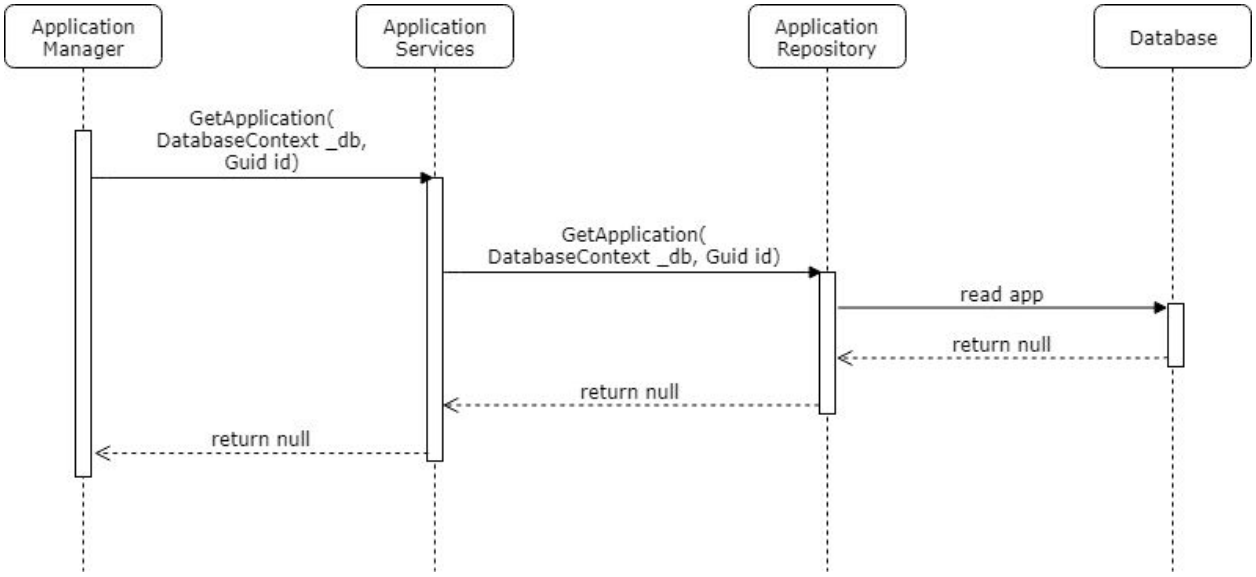

Figure 2.2.2 Reading a Non-Existing Application

2.2.2 Read By Title and Email

When the Read service is called, the Database Context, Application title, and Application email are passed to the Application Repository. The repository will attempt to retrieve the application. If the application is found, it will be returned. (Figure 2.2.3) If it is not found, null will be returned instead. (Figure 2.2.4)

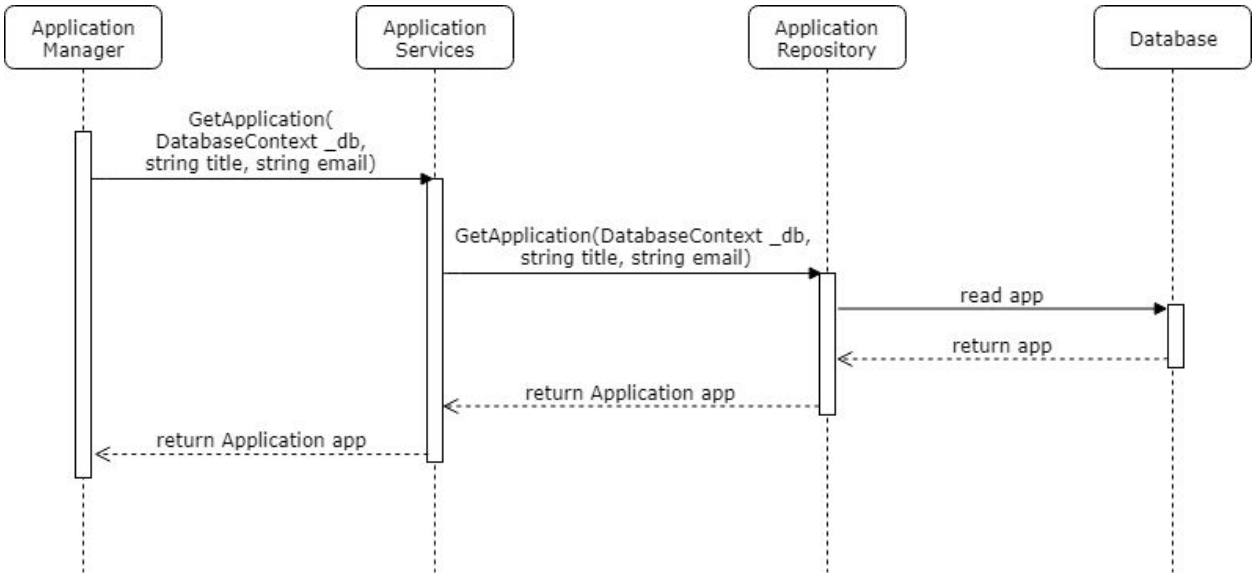

Figure 2.2.3 Reading an Existing Application

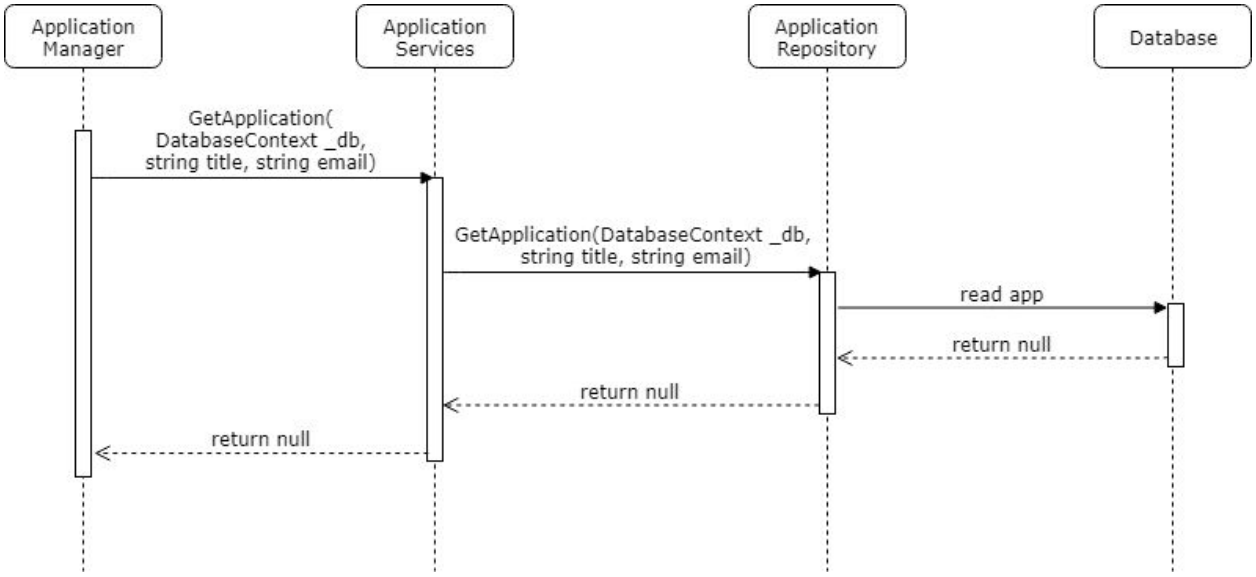

Figure 2.2.4 Reading a Non-Existing Application

2.3 Update an Application

When the Update service is called, the Database Context and Application are passed to the Application Repository. The repository will first check if the application exists in the database. If the application exists, then the current application will be modified, then returned back to the call. (Figure 2.3.1) If the application does not exist, then null will be returned instead. (Figure 2.3.2)

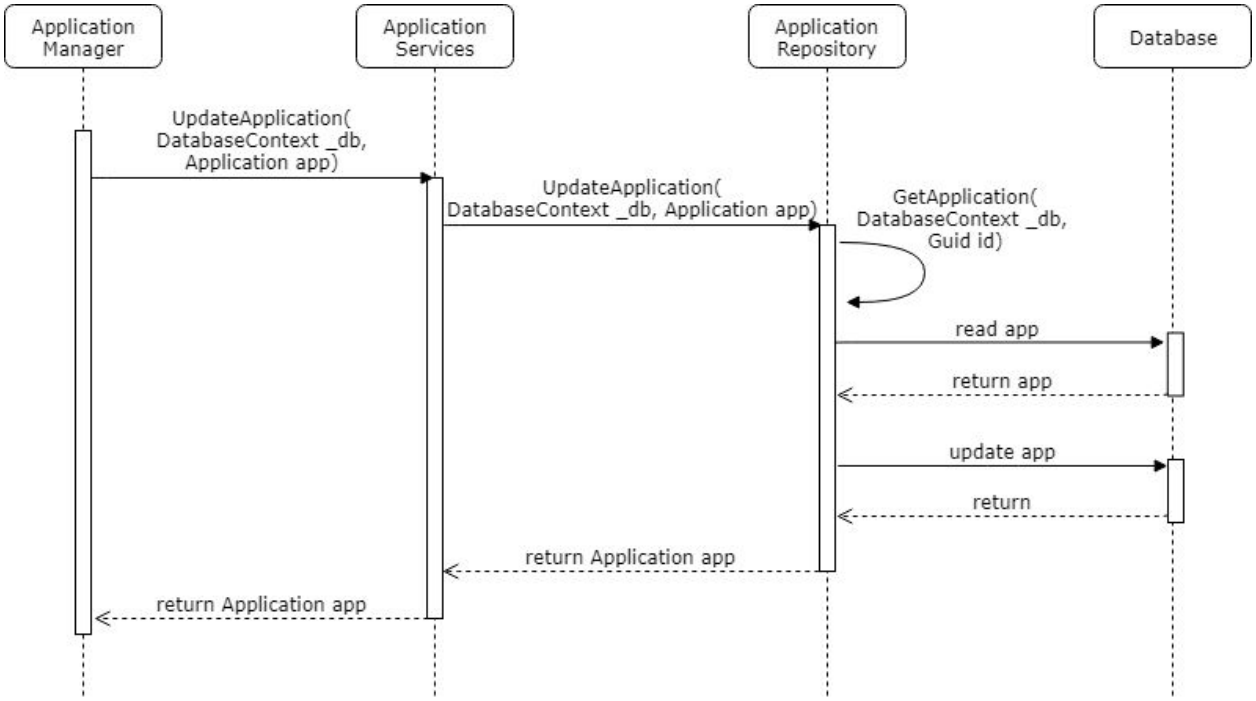

Figure 2.3.1 Updating an Existing Application

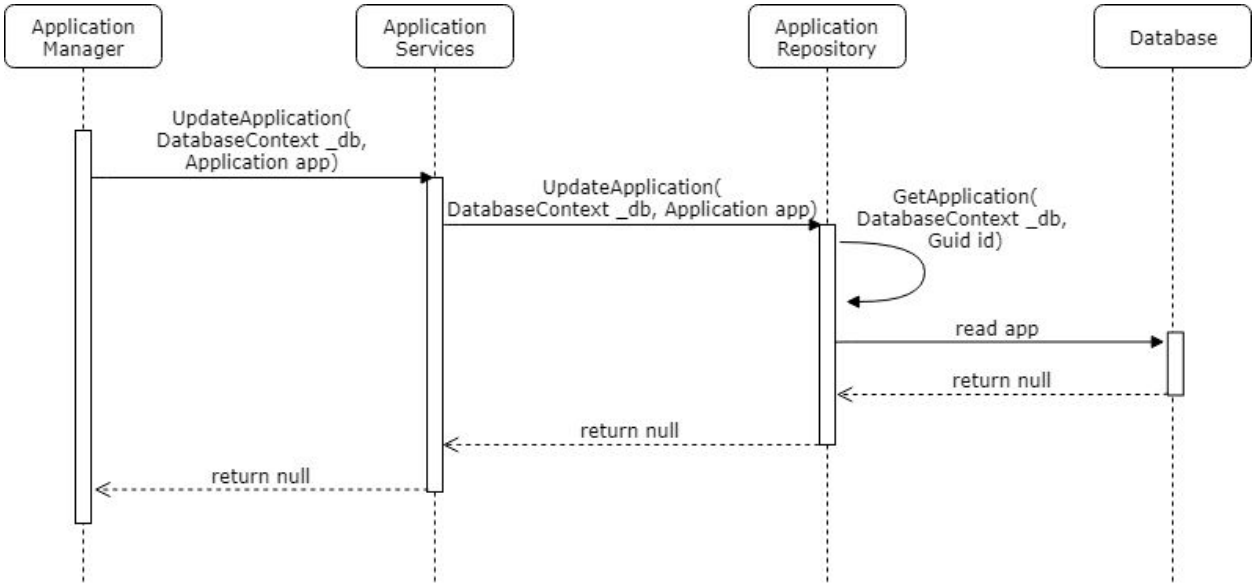

Figure 2.3.2 Updating a Non-Existing Application

2.4 Delete an Application

When the Delete service is called, the Database Context and Application Id are passed to the Application Repository. The repository will attempt to retrieve the application. If the application is found, it will be deleted then returned. (Figure 2.4.1) If it is not found, null will be returned instead. (Figure 2.4.2)

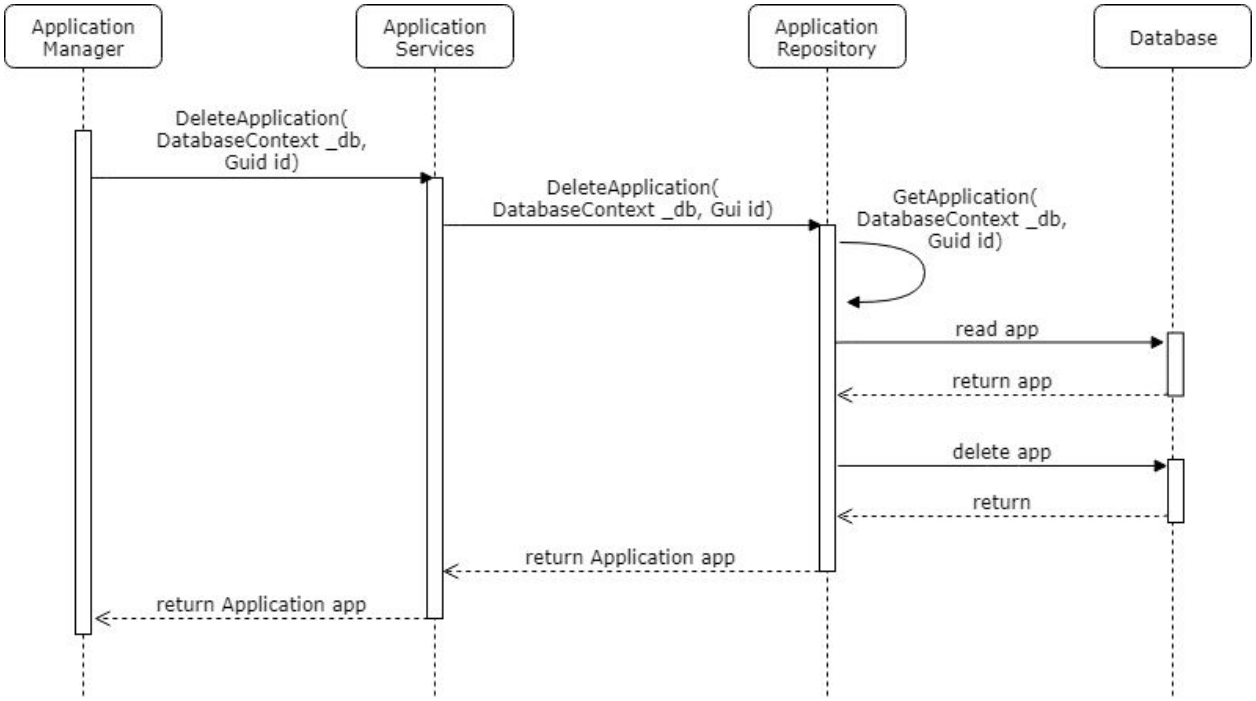

Figure 2.4.1 Deleting an Existing Application

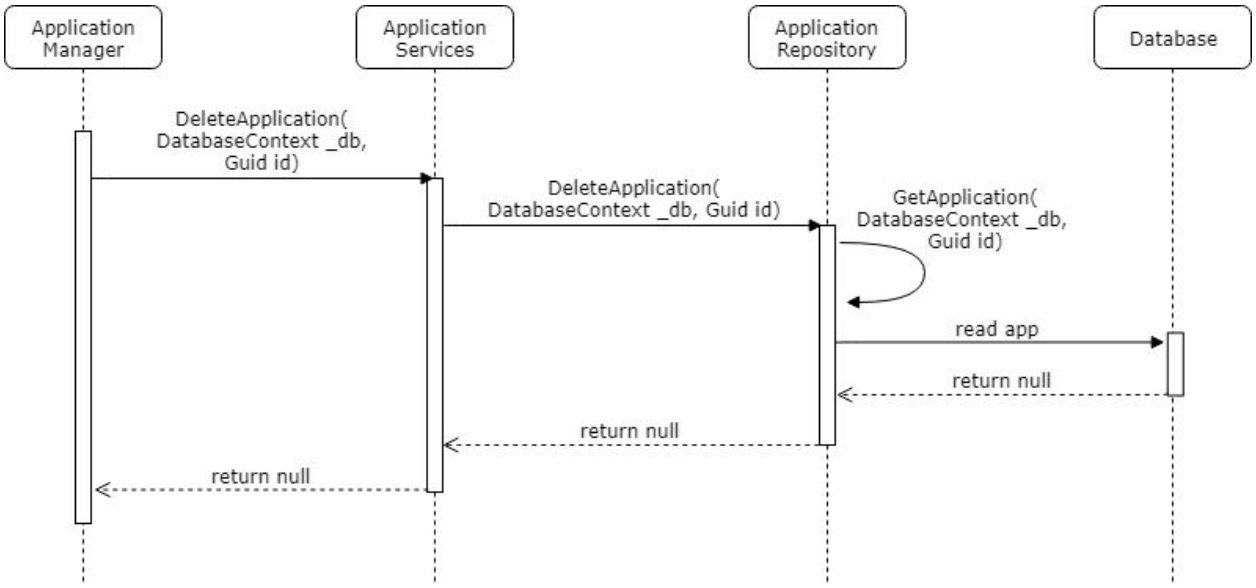

Figure 2.4.2 Deleting a Non-Existing Application

3. ApiKey Services

The following diagrams represent the flow of calling the operations to create, update, read and delete an ApiKey entry in the data store.

3.1 Create an ApiKey

When the Create service is called, the Database Context and ApiKey are passed to the ApiKey Repository. The repository will first check if the ApiKey exists in the database. If the ApiKey does not already exist, then a new record of the ApiKey will be created and return back to the call. (Figure 3.1.1) If the ApiKey does already exist, then the ApiKey record will not be created and null will be returned instead. (Figure 3.1.2)

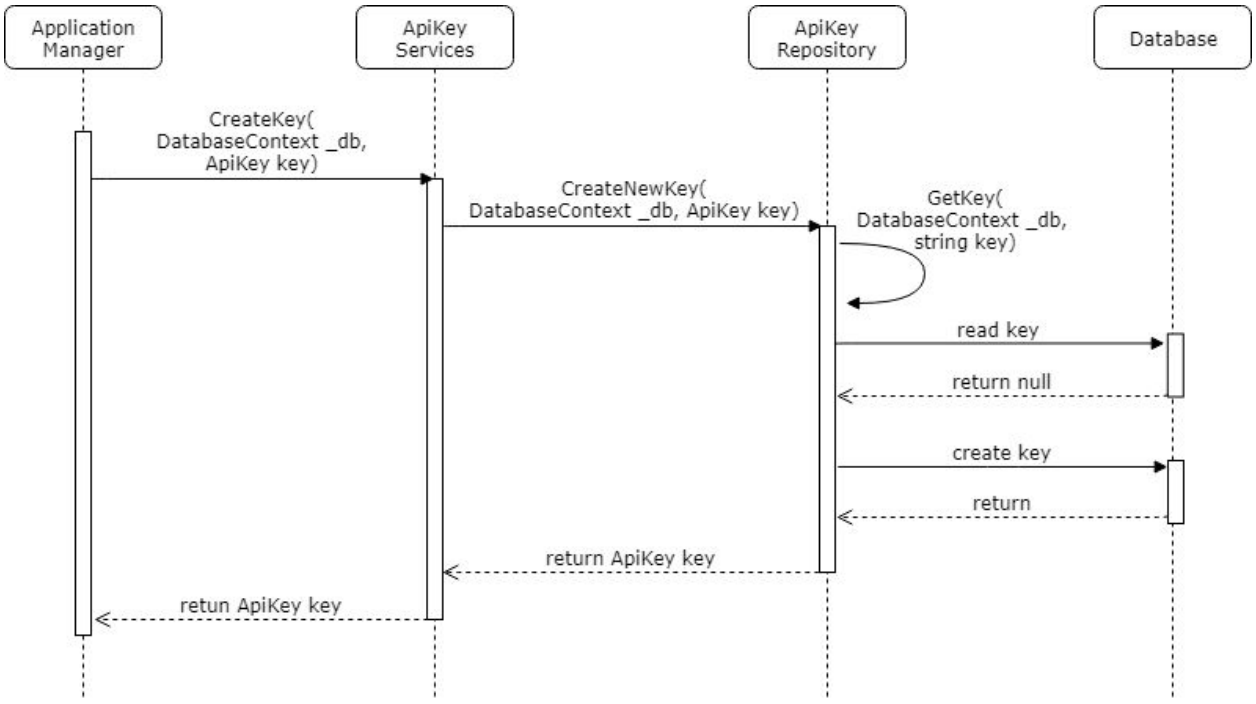

Figure 3.1.1. Creating a Non-Existing ApiKey

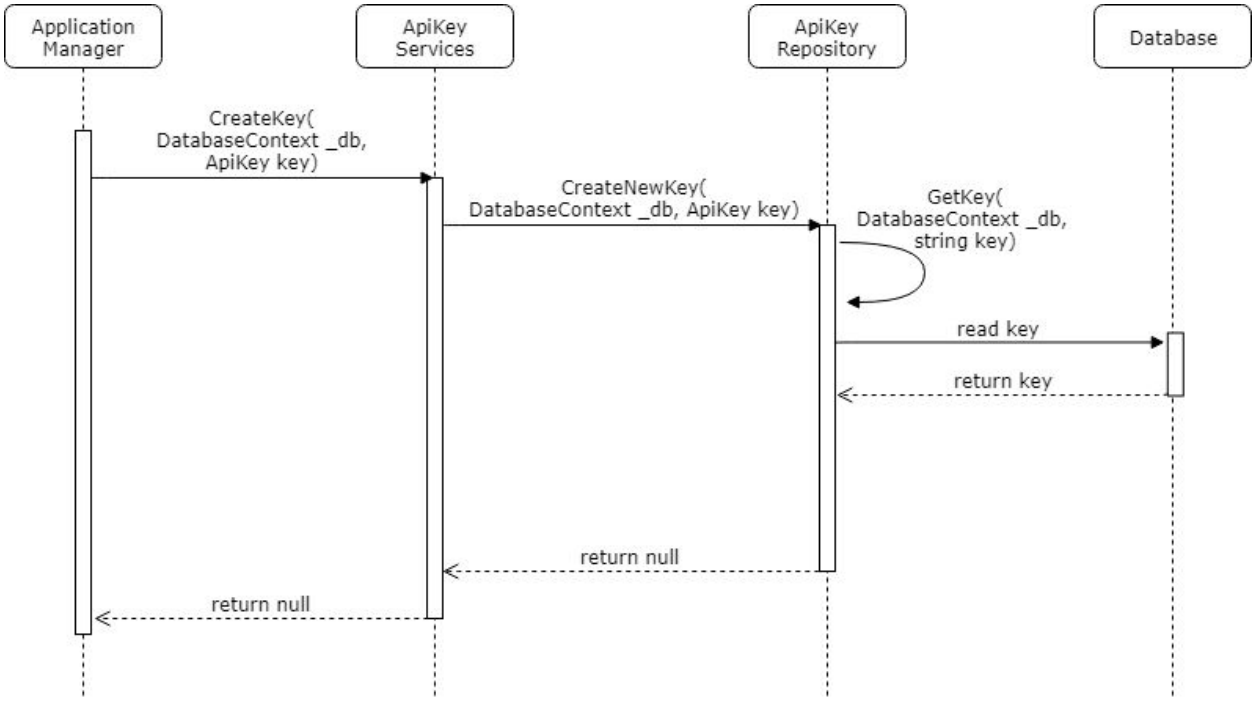

Figure 3.1.2 Creating an Existing ApiKey

3.2 Read an ApiKey

3.2.1 Read By Id

When the Read service is called, the Database Context and ApiKey Id are passed to the ApplicatApiKey ion Repository. The repository will attempt to retrieve the ApiKey. (Figure 3.2.1) If the ApiKey is found, it will be returned. If it is not found, null will be returned instead. (Figure 3.2.2)

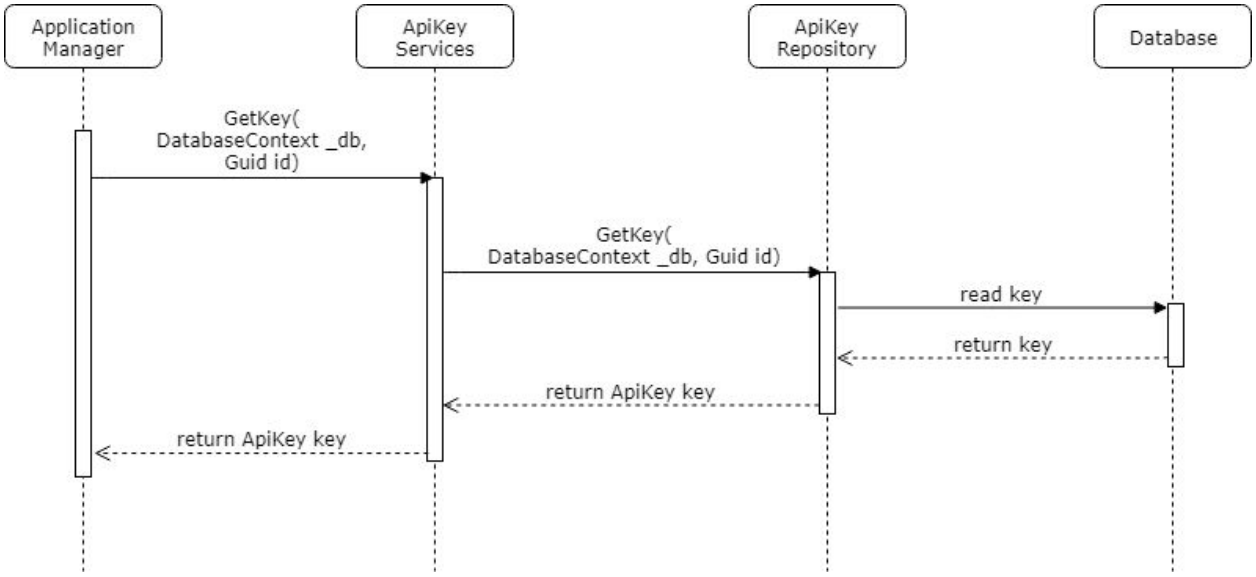

Figure 3.2.1 Reading an Existing ApiKey

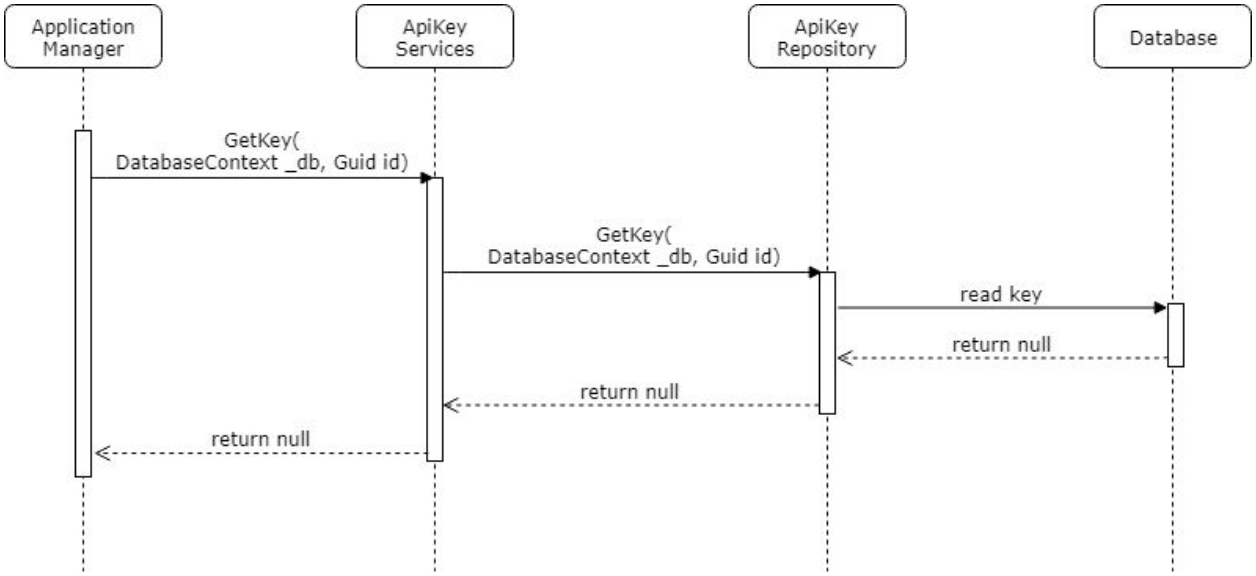

Figure 3.2.2 Reading a Non-Existing ApiKey

3.2.2 Read By Key value

When the Read service is called, the Database Context and ApiKey key value are passed to the ApiKey Repository. The repository will attempt to retrieve the ApiKey. (Figure 3.2.3) If the ApiKey is found, it will be returned. If it is not found, null will be returned instead. (Figure 3.2.4)

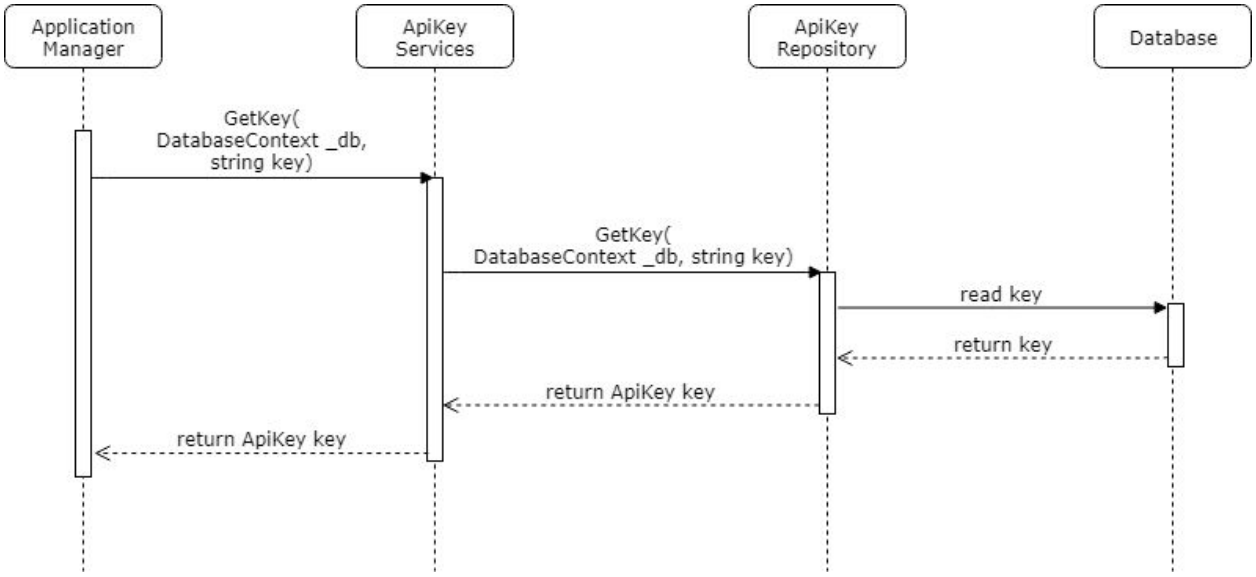

Figure 3.2.3 Reading an Existing ApiKey

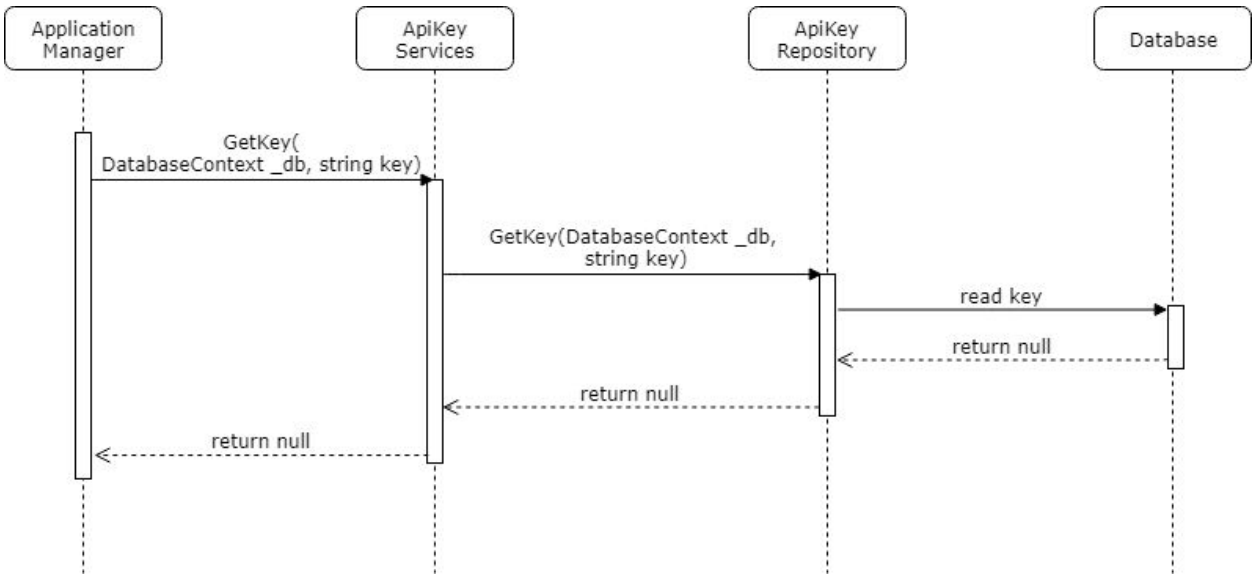

Figure 3.2.4 Reading a Non-Existing ApiKey

3.2.3 Read By Application Id and IsUsed Field

When the Read service is called, the Database Context, ApiKey's Application Id, and IsUsed value are passed to the ApiKey Repository. The repository will attempt to retrieve the ApiKey. (Figure 3.2.5) If the ApiKey is found, it will be returned. If it is not found, null will be returned instead. (Figure 3.2.6)

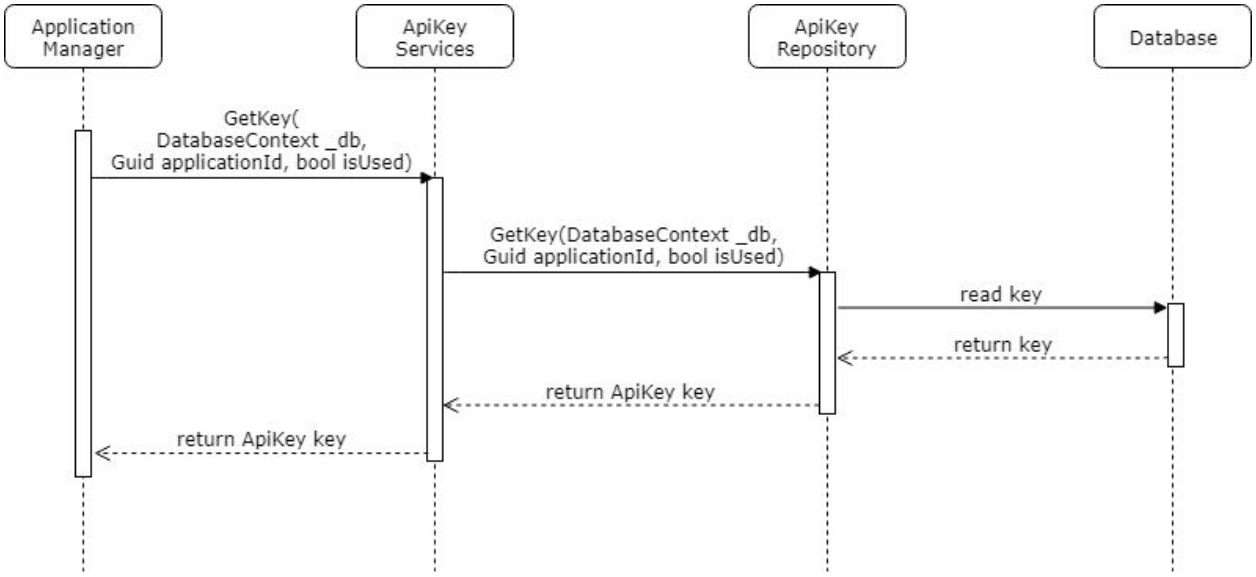

Figure 3.2.5 Reading an Existing ApiKey

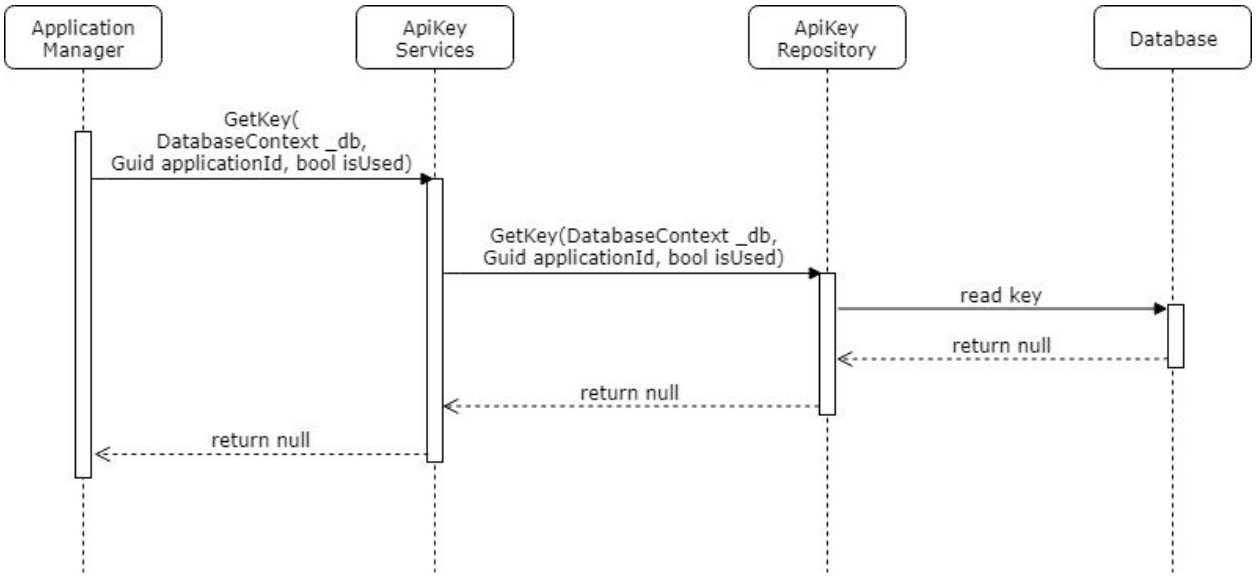

Figure 3.2.6 Reading a Non-Existing ApiKey

3.3 Update an ApiKey

When the Update service is called, the Database Context and ApiKey are passed to the ApiKey Repository. The repository will first check if the ApiKey exists in the database. If the ApiKey exists, then the current ApiKey will be modified, then returned back to the call. (Figure 3.3.1) If the ApiKey does not exist, then null will be returned instead. (Figure 3.3.2)

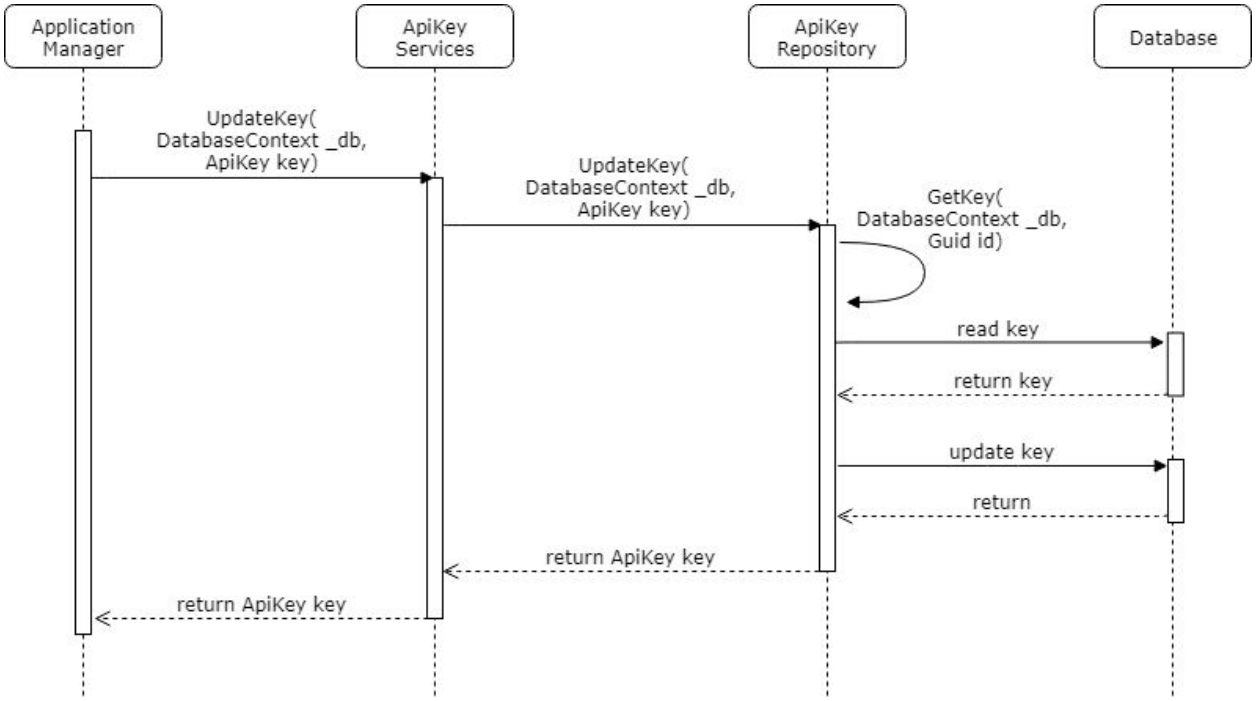

Figure 3.3.1 Updating an Existing ApiKey

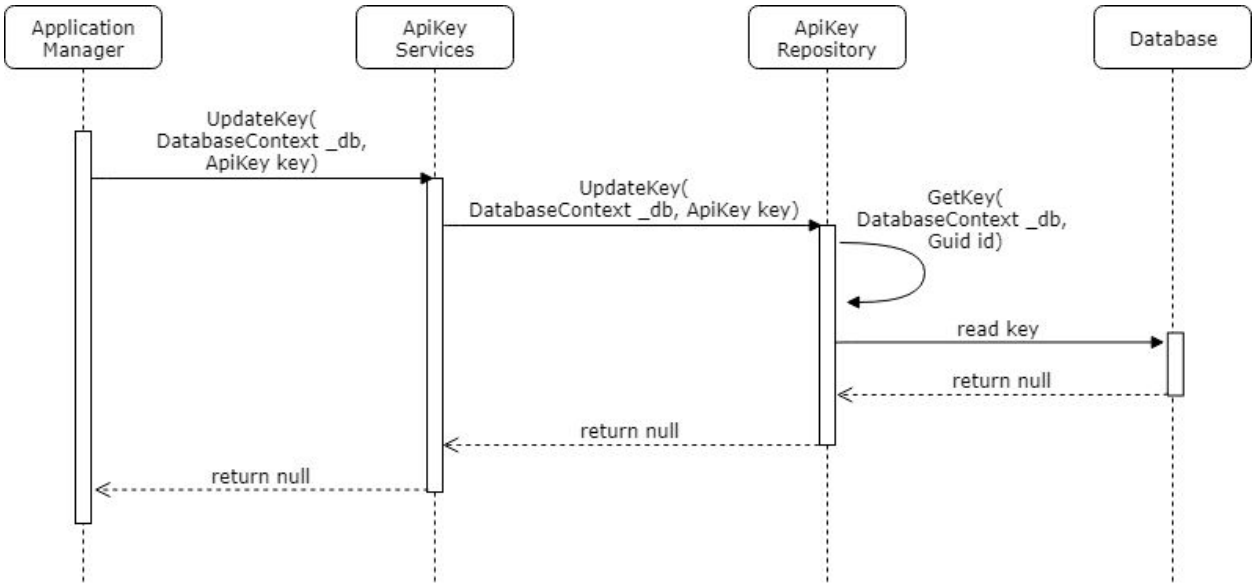

Figure 3.3.2 Updating a Non-Existing ApiKey

3.4 Delete an ApiKey

When the Delete service is called, the Database Context and ApiKey Id are passed to the ApiKey Repository. The repository will attempt to retrieve the ApiKey. If the ApiKey is found, it will be deleted then returned. (Figure 3.4.1) If it is not found, null will be returned instead. (Figure 3.4.2)

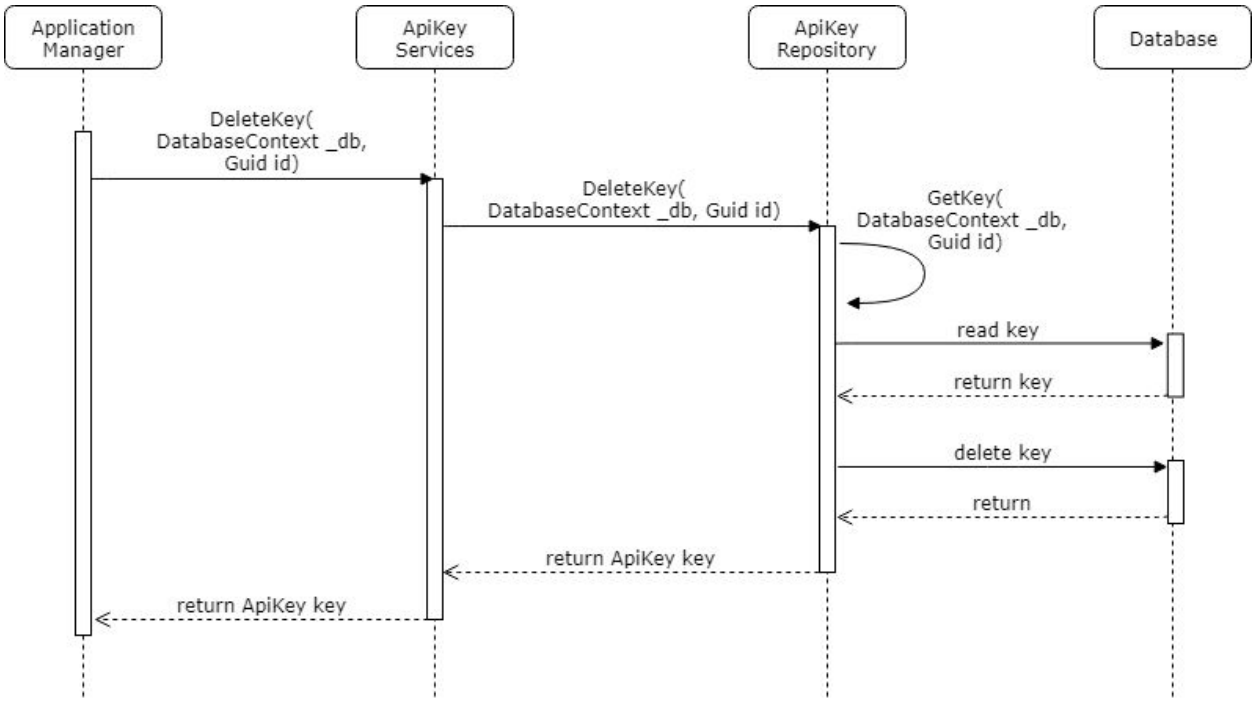

Figure 3.4.1 Deleting an Existing ApiKey

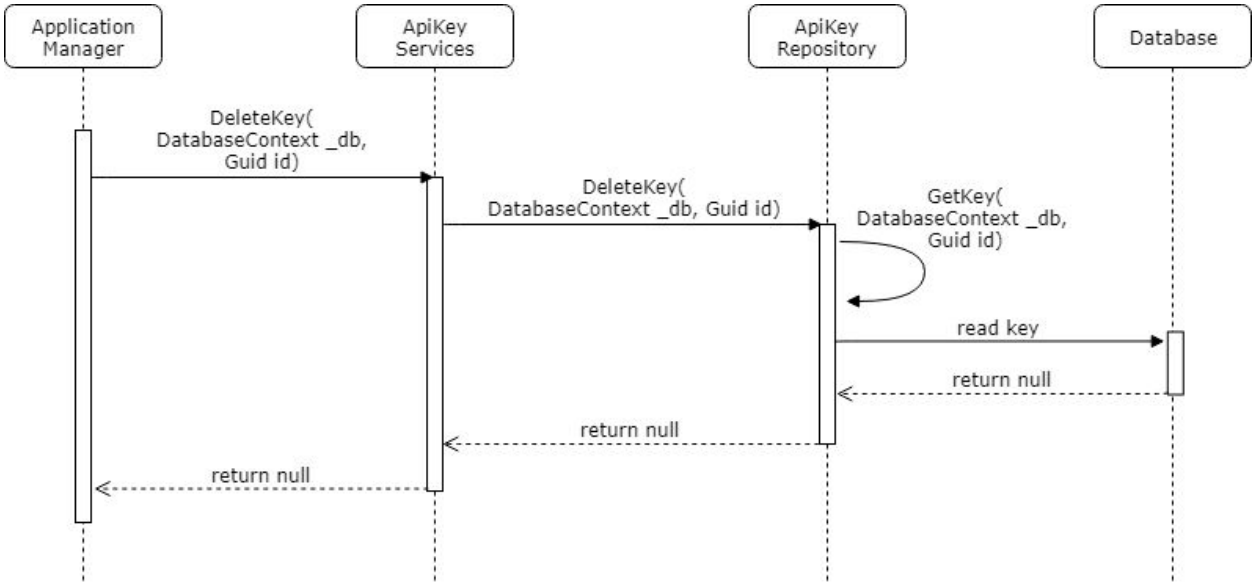

Figure 3.4.2 Deleting a Non-Existing ApiKey

4. Functionalities

The following diagrams represent the flow of registering an application, publishing an application, deleting an application, and requesting a new API key in the KFC SSO portal.

4.1 Register an Application

When a user wishes to register their application into the SSO portal, they must input an application title, a url to direct to their application, a contact email, and a url to route to user deletion within their application. This data is then sent through an HTTP POST request from the KFC frontend server to an endpoint in the KFC Web API. The request is then routed to the Application Manager to validate all the request entries. If all the entries are valid, then the manager calls the Application service to create an application. If the application does not exist, then a unique key value is generated by the Token service. This key value is then used to create a new ApiKey. If successful, an HTTP Status Code of 200 is returned to the frontend server with the api key value to publish the new application. (Figure 4.1.1)

If the application already existed before creation, then an HTTP Status Code of 400 is returned with a failure message. (Figure 4.1.2)

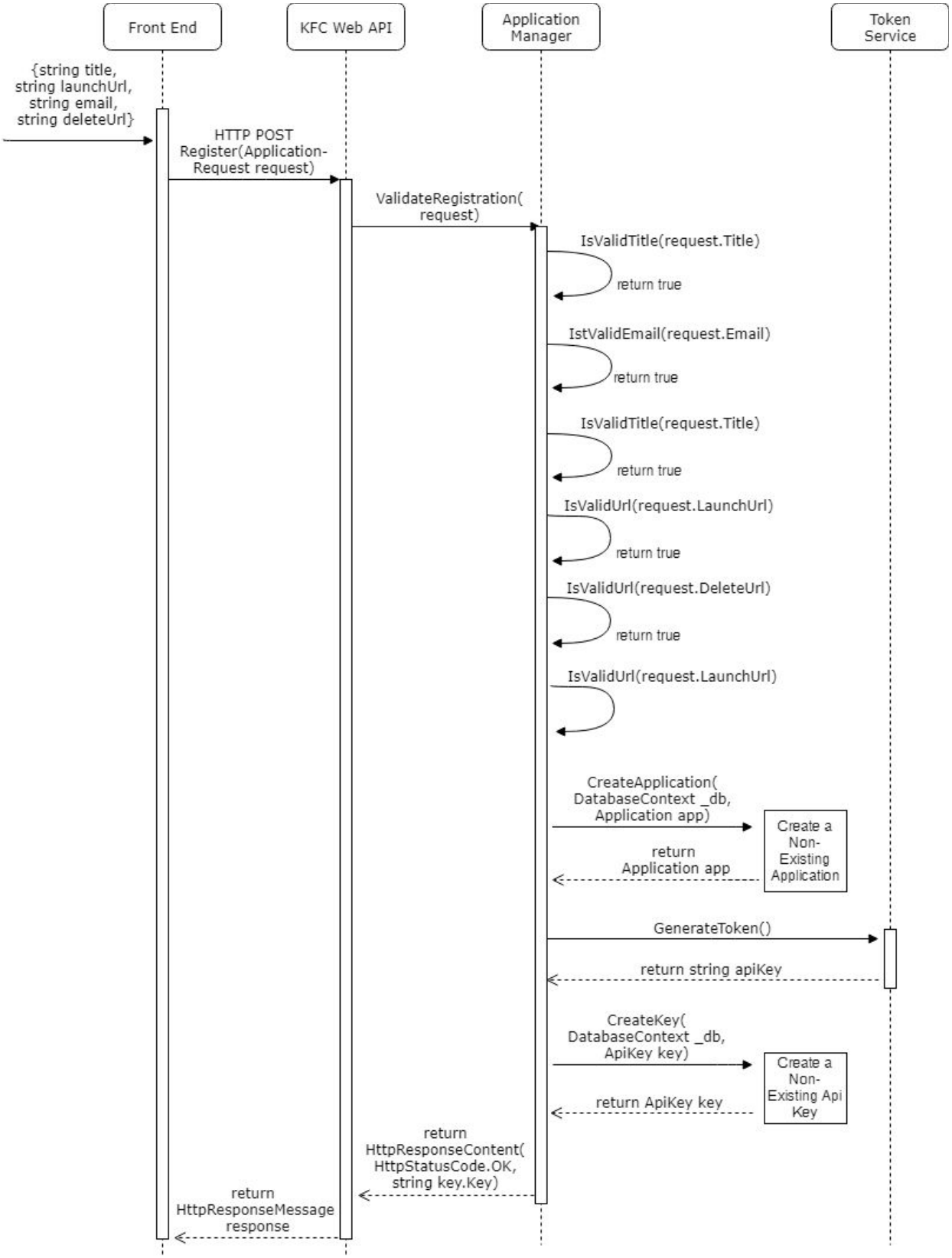

Figure 4.1.1 Registering a Non-Existing Application

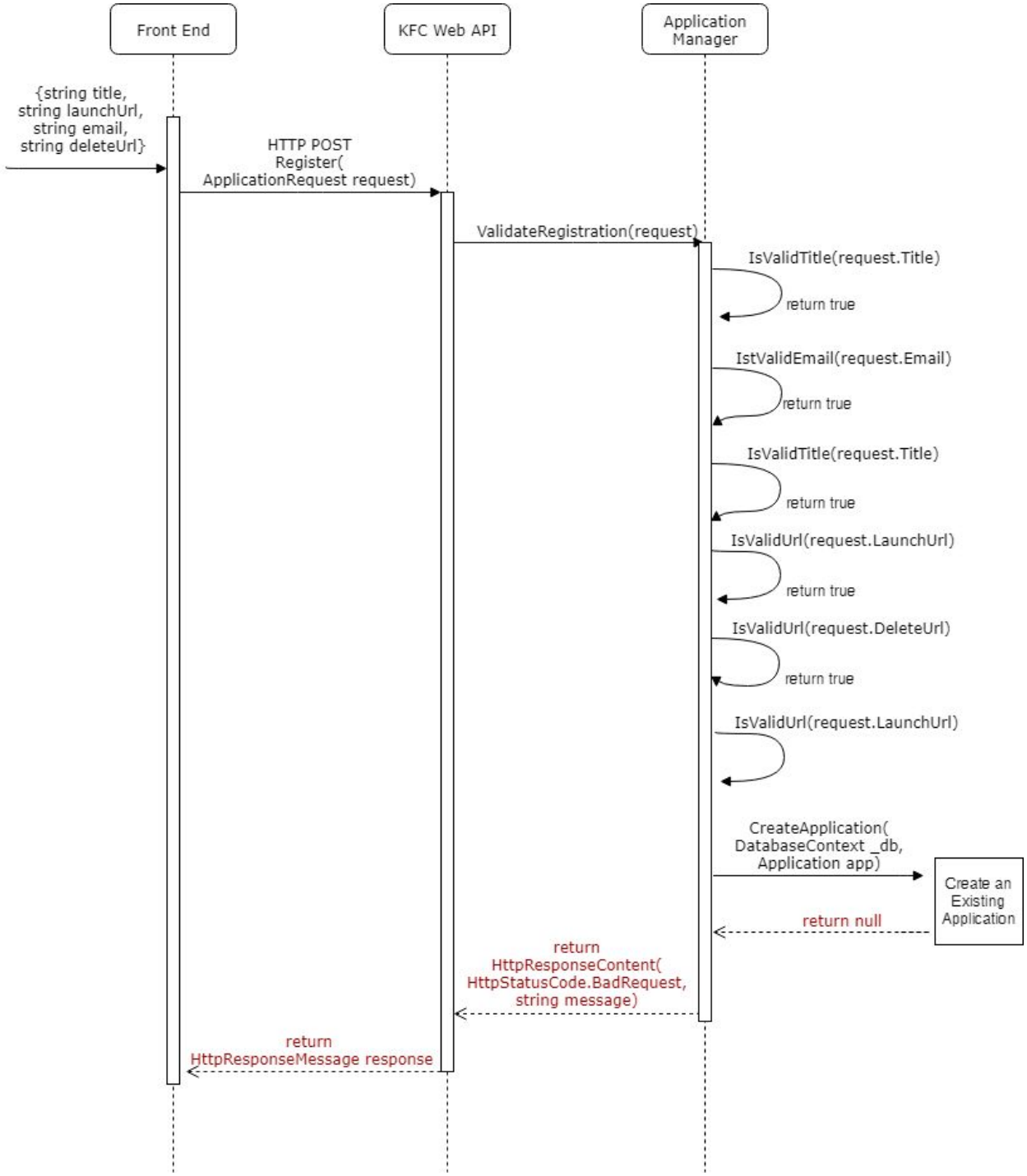

Figure 4.1.2 Registering an Existing Application

4.2 Publish an Application

When a user wishes to publish their application into the SSO portal, they must input an API key value, an application title, a logo url, and an application description. This data is then sent through an HTTP POST request from the individual application server to an endpoint in the KFC Web API. The request is then routed to the Application Manager to validate all the request entries. If all the entries are valid, then the manager calls the ApiKey service to find the api key value. If the key value exists, is assigned to the correct application, and is not used yet, then the manager calls the Application service to find the application to check that it is still registered. If the application is still registered, then the application is updated with the new published data and the ApiKey is updated to show the key has been used. (Figure 4.2.1)

If the published key value does exist, but the application trying to publish does not, then an HTTP Status Code of 400 and a failure message is returned to the individual application's frontend server. (Figure 4.2.2)

If the published key value does not exist, is assigned to a different application, or is already used, then an HTTP Status Code of 400 and a failure message is also returned to the individual application's frontend server (Figure 4.2.3)

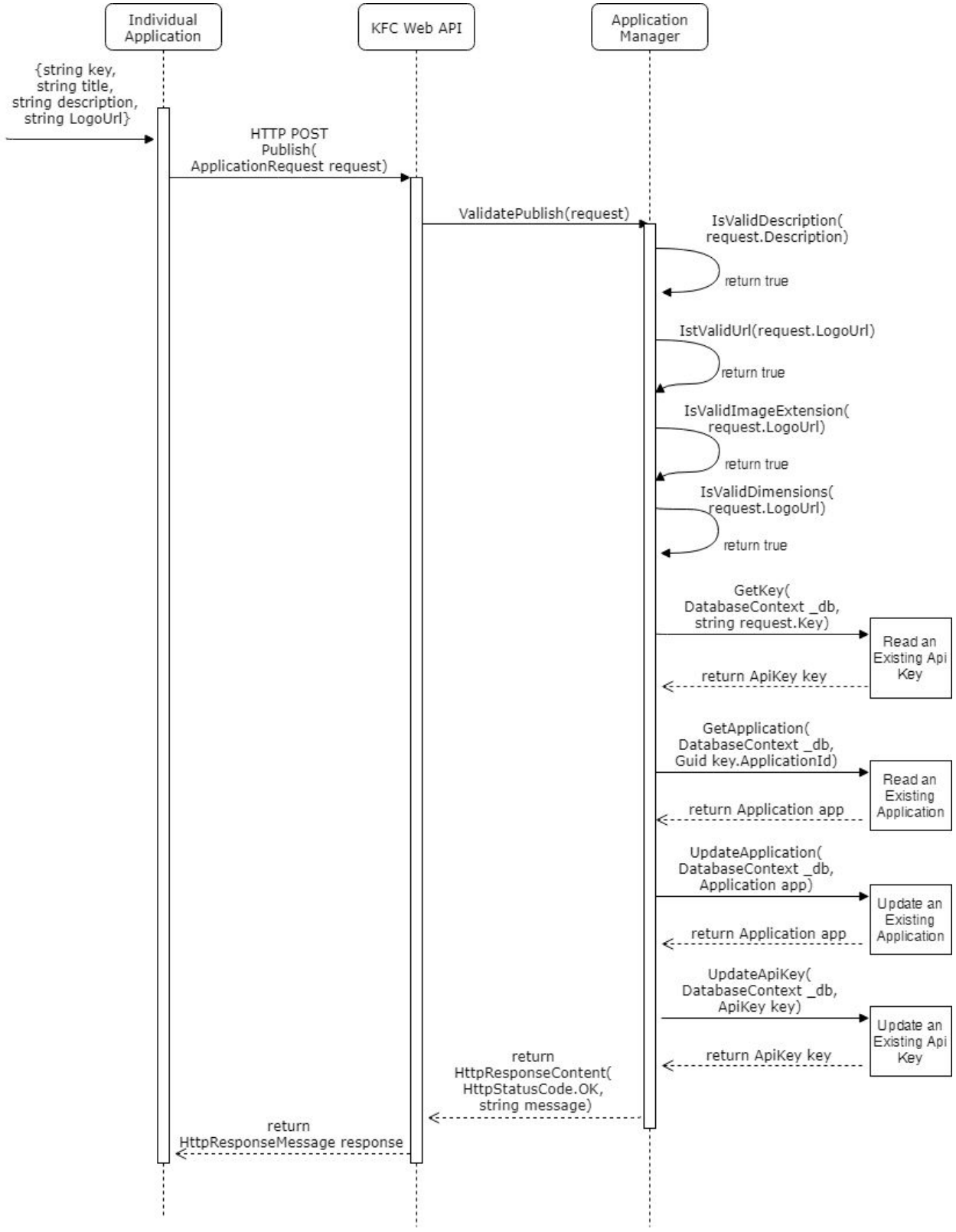

Figure 4.2.1 Publishing an Existing Application

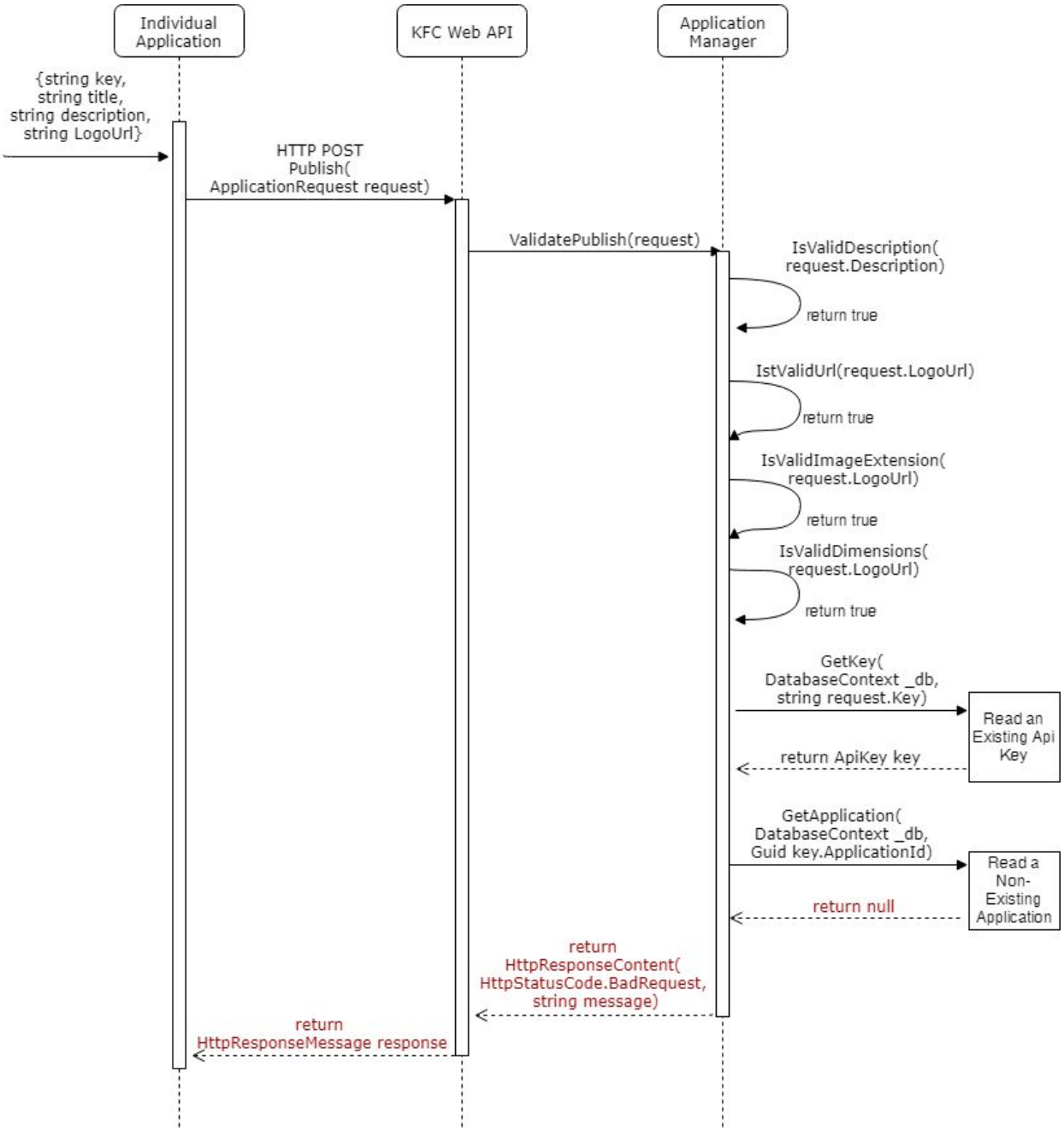

Figure 4.2.2 Publishing a Non-Existing Application

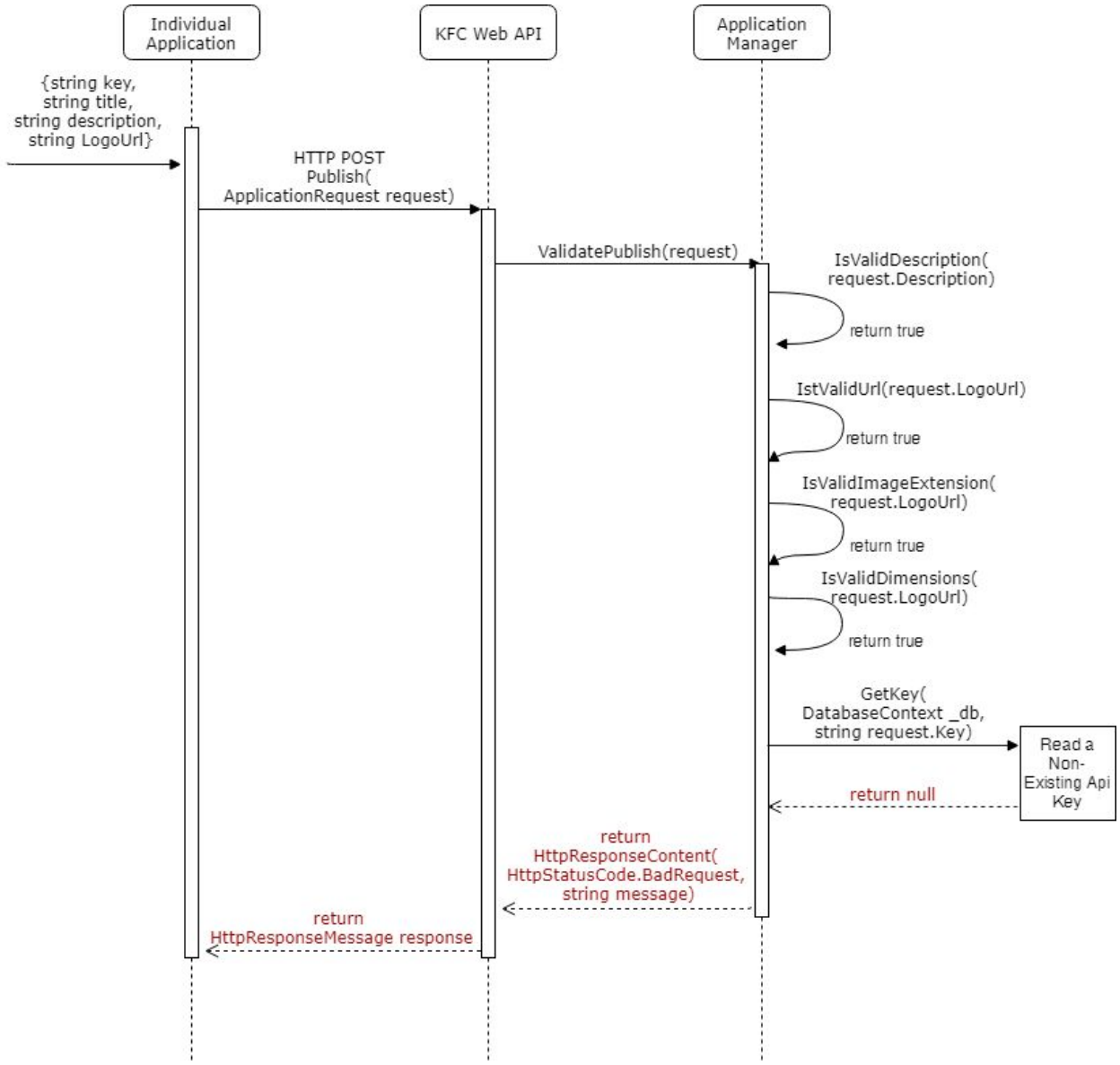

Figure 4.2.3 Publishing an Existing Application With an Invalid Key

4.3 Delete an Application

When a user wishes to delete their application from the SSO portal, they must input the application title and a contact email. This data is then sent through an HTTP POST request from the KFC frontend server to an endpoint in the KFC Web API. The request is then routed to the Application Manager to validate all the request entries. If all the entries are valid, then the manager calls the Application service to find the application. If the application exists, then the

service to delete the application is called. If all is successful, then an HTTP Status Code of 200 and a success message is returned to the KFC frontend server. (Figure 4.3.1)

If the application does not exist, then the application is not deleted and an HTTP Status Code of 400 and a failure message is returned to the KFC frontend server. (Figure 4.3.2)

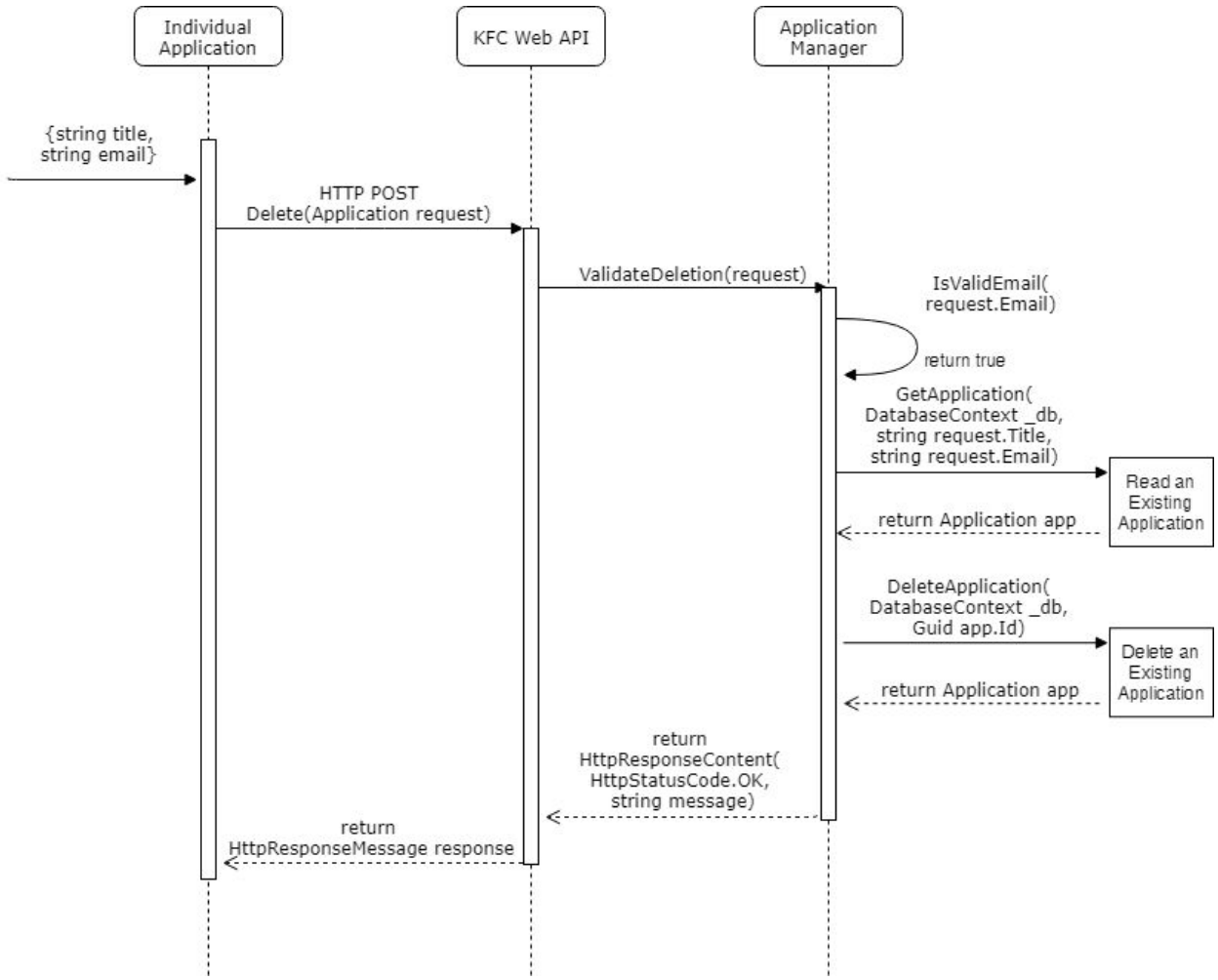

Figure 4.3.1 Deleting an Existing Application

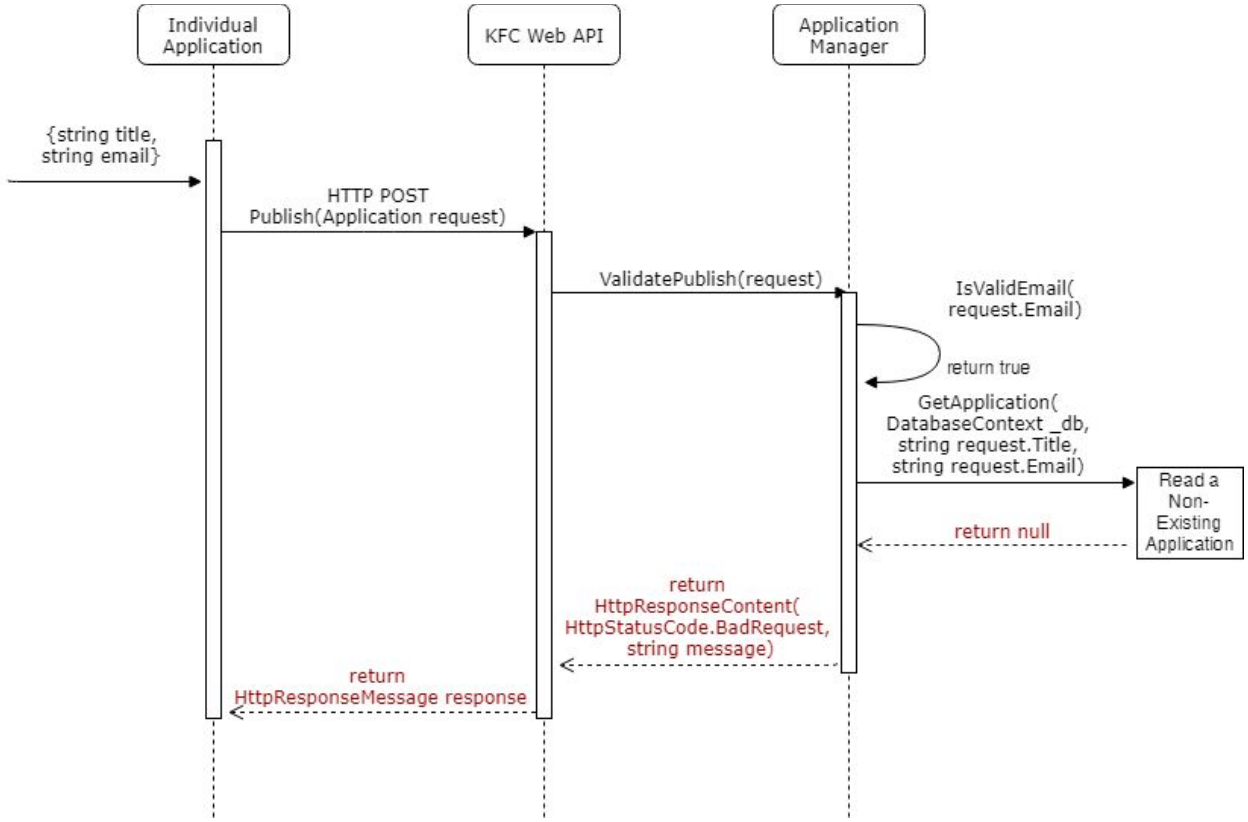

Figure 4.3.2 Deleting a Non-Existing Application

4.4 Request a New API Key

When a user wishes to request a new API Key for their application, they must input the application title and a contact email. This data is then sent through an HTTP POST request from the KFC frontend server to an endpoint in the KFC Web API. The request is then routed to the Application Manager to validate all the request entries. If all the entries are valid, then the manager calls the Application service to find the application. If the application exists, then the Token service is called to generate a new unique key value. The manager then calls the service to read the previous API Key of the application that has not been used. If it exists, the API Key will be updated with its IsUsed field assigned as True in order to invalidate it. This previously generated key value is then used to create a new ApiKey. If all is successful, then an HTTP Status Code of 200 and the new API Key value is returned to the KFC frontend server. (Figure 4.4.1)

If the application does not exist, then a new API key is not created and an HTTP Status Code of 400 and a failure message is returned to the KFC frontend server. (Figure 4.4.2)

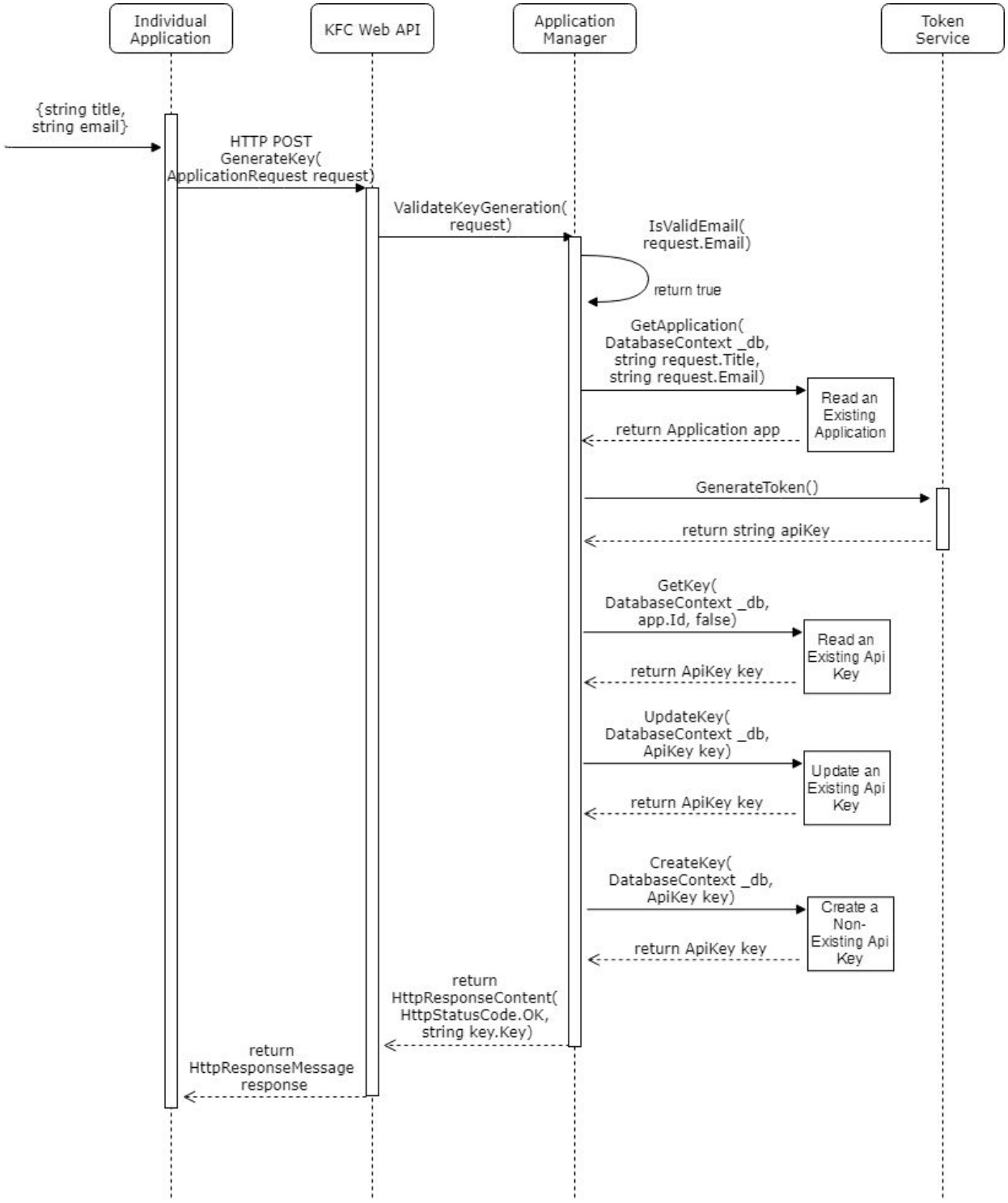

Figure 4.4.1 Requesting a New API Key for an Existing Application

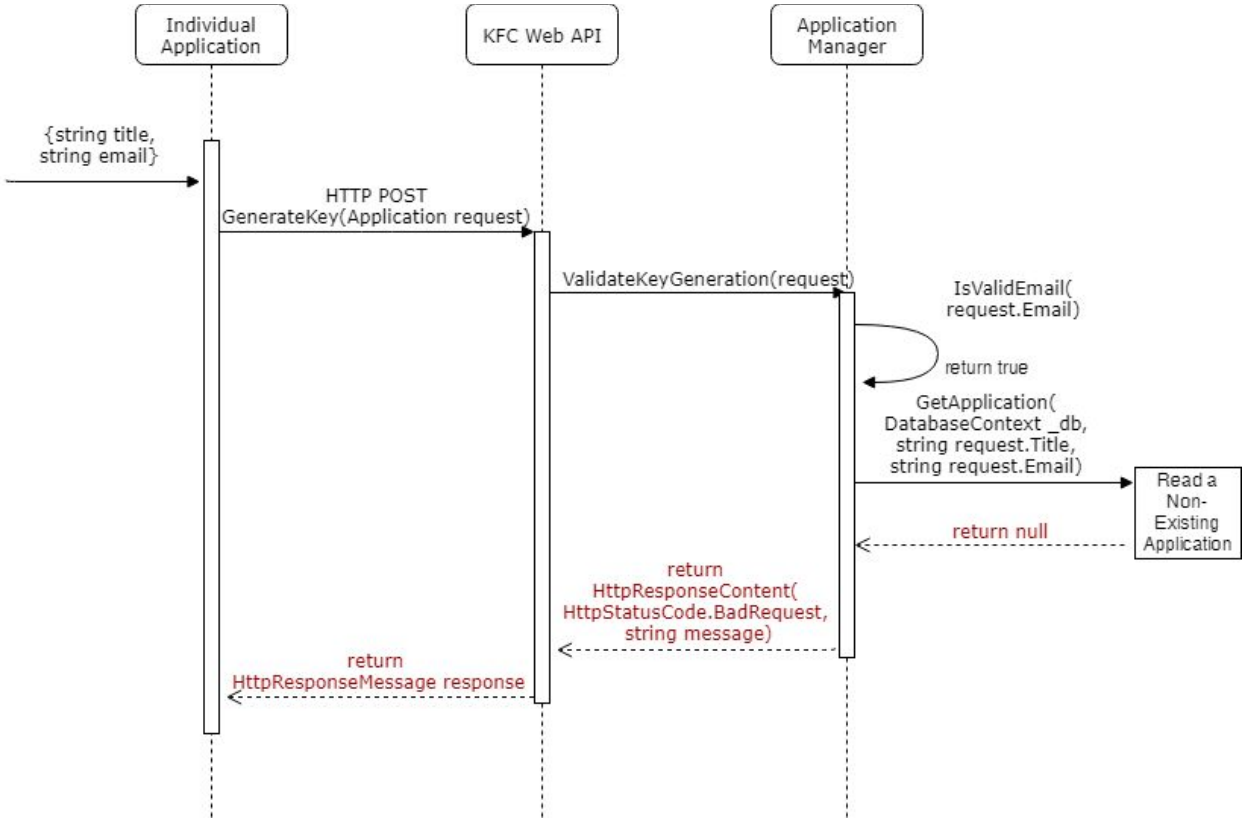

Figure 4.4.1 Requesting a New API Key for a Non-Existing Application

5. Additional Errors

- If any request entries are null or invalid, then an HTTP response with a status code of 400 and a failure message will be returned.
- If the database fails to save any changes after the Application or ApiKey services, then a *System.Data.Entity.Validation.DbEntityValidationException* exception will be thrown and a HTTP response with a status code of 500 and a failure message will be returned.
- If any null values are passed into the Application Repository or ApiKey Repository, then null is returned.