

---

# NOMADS

---

## NOMAD TRAVEL

(NT)

### SOFTWARE DESIGN DOCUMENT

Prepared by:  
Camille Lewis  
Mustafa Sameen  
Nathaniel Curl

RELEASE DATE  
MARCH 22, 2023

# Table of Contents

Executive Summary	4
Project Description	5
<b>Document Versioning</b>	<b>6</b>
<b>Features</b>	<b>7</b>
Feature Matrix	8
Feature Discussion	8
P.1 - Language	8
I.1 - CLI	9
C.1 - MainApp	9
C.2 - UserProfile	9
C.3 - DataGenerator	9
C.4 - DataProcessor	10
C.5 - Country	10
T.1 - Error Handling	10
I. 2 - GUI	10
<b>System Design</b>	<b>10</b>
Architecture Overview	11
High-level System Architecture	11
Major Components	11
InputAdapter	11
MainApp	11
User	12
Country	12
DestinationGenerator	12
Model	12
View	12
Controllers	12
Detail Design	12
InputAdapter	12
InputAdapter	13
TextInterface	13
GraphicInterface	13
MainApp	15
MainApp	15
User Interface	15
DestinationGenerator	15
DestinationGenerator	16

Generate	16
Order	16
User	17
Country	17
Model	17
UserModel	18
CountryModel	18
View	19
login-view	19
register-view	20
destination-generator-view	20
favorites-view	20
update-user-view	20
country-card-view	20
Controller	20
CountryCardController	20
DestinationGeneratorController	20
FavoritesController	21
LoginController	21
RegisterController	21
UpdateUserController	21
Data Formats	22
User Database	22
Preferences Database	22
Countries Database	23

## Executive Summary

NOMAD Travel (NT) is a mobile application that simplifies users' logistical travel planning. NOMAD Travel is intended to operate in the market similarly to Expedia, Canoe, and Hopper but includes unique features. NOMAD Travel will utilize travel regulation information from respective governments to provide the user with visa-dependent travel options. NOMAD Travel will utilize personalized e-commerce models and will be distributed as a stand-alone application on mobile devices.

This document provides nontechnical information regarding the purpose and behavior of NT.

## Project Description

NOMAD Travel is attempting to break into the personal travel planning industry. The app is intended to target a gap that currently exists in the travel industry: travel regulation information, such as visa travel status, is not easily accessible to consumers of personal travel planning. NOMAD Travel will provide customized travel destinations and opportunities to users depending on their visa and location of origin. Nomad travel will also be able to recommend destinations to users based on preferences they set. NOMAD Travel will use information from travel laws and local governments to connect users to destinations, depending on whether or not they can easily travel to the destination with their Visa.

NOMAD travel must be standalone and not require any network connectivity. The NOMAD Application engines must be installable directly on end user hardware. To appeal to the largest Market, NOMAD travel should be hardware and operating system agnostic.

NOMAD travel will feature an intuitive and user-friendly Graphical User Interface (GUI) that allows travelers to navigate and interact with the app's features and content seamlessly. The GUI will be designed to be visually appealing and easy to understand. It will provide users with a comfortable and engaging experience and enhance the user experience to promote user engagement with the app.

NOMAD travel features a recommendation system that will intake user preferences including climate and activities and output ideal destinations for the user. The recommendation system is designed to offer the user desirable and logical choices for their next vacation, business trip, or getaway. It will save users time and effort with planning logistics of travel, activities, and accommodations. This efficiency will encourage users to travel more frequently using NOMAD.

## Document Versioning

Date	Owner	Comment
Mar 6, 2023	Camille Lewis	Feature Matrix & Description, Data Formats
Mar 6, 2023	Mustafa Sameen	Architecture Overview, Detail Design
Mar 6, 2023	Nathan Curl	Diagram, Architecture

Mar 12, 2023	Camille Lewis	Edited Apploop and component descriptions for v2
Mar 15, 2023	Mustafa Sameen & Camille Lewis	Edited and updated the whole document to meet up to date/GUI standards.
Mar 15, 2023	Nathan Curl	Uploaded Diagram for GUI
Mar 21, 2023	Mustafa Sameen, Camille Lewis, Nathan Curl	Updated the whole document to meet up to date standards

## Features

The feature matrix enumerates the features requested for the project and the discussion section provides details regarding the intent of the feature. The ids will be used for traceability. Features that all stakeholders have agreed can be removed should strike-through the feature id and have a comment added to discuss the feature being dropped.

Priority Codes:

H - High, a must have feature for the product to be viable and must be present for launch

M - Medium, a strongly desirable feature but product could launch without

L - Low, a feature that could be dropped if needed

## Feature Matrix

ID	Pri	Feature Name	Comment	BRD ID
P.1	H	Language	Implementation in JAVA	
I.1	H	CLI		ux1
C.1	H	MainApp		e1,e2,e3,d1,d2,d3,d4
C.2	M	UserProfile		e1,e2
C.3	H	DataGenerator		e1,e2,e3,d1,d2,d3,d4
C.4	M	DataProcessor		d1,d2,d3,d4
C.5	M	Country		e3,d1,d2,d3,d4
T.1	M	Error Handling		e5
I.2	H	GUI		ux1, ux2, ux3, ux4, ux5

## Feature Discussion

### P.1 - Language

NOMAD must be multi-platform and run as a standalone application. JAVA is the language selected for this application to run all necessary functions and possible expansions that may include a future GUI or more complex user/data interactions. This language will also support our actions necessary for CLI support.



## I.1 - CLI

NOMAD Travel has a fairly simple user interaction pattern that is well suited for CLI operation. NOMAD Travel provides a command line interface for executing travel searches in a command line terminal.

### C.1 - MainApp

Apploop is the main class of the software. It contains a DataReader/Writer and a Scanner to retrieve user input. All NOMAD user interactions will use the same basic event loop.

1. Display welcome message
2. Display user prompts to sign in or create a new profile
3. If new user, Get user input
4. Create User Profile, (jump to step 8)
5. If returning user, prompt user to view destinations or view favorites
6. If user selects view favorites display list of favorites from user profile favorites file, prompt user to return to view destinations or view favorites
7. If user selects view destinations (jump to step 8)
8. Use User Profile to run Destination Generator
9. Display Destination List to User
10. Prompt User to select a Country Card, favorite Countries, or Exit App
11. If user selects a country card, display country information
12. Prompt user to favorite country card or return to Destination List
13. If user favorites a country card, Save user favorited Countries to favorites file within user profile using a dataReaderWriter
14. Return to country card
15. If user selects return to Destination List, display list of destinations
16. Exit application if user selects exit

### C.2 - UserProfile

A class that contains information collected from user input and a scanner that always runs within AppLoop. Contains user's Name, Nationality, and an ArrayList<Countries> of favorited destinations.

### C.3 - DataGenerator

To create a list of recommended destinations for the user, the application will create an ArrayList<Countries> of Country objects. The DataGenerator utilizes a User Profile object and a DataProcessor Object to create the ArrayList of countries. The ArrayList of countries will be displayed to the user after the DataGenerator has stopped running and successfully created an ArrayList of countries.

## C.4 - DataProcessor

The data processor will use user input from a scanner initialized in the AppLoop to parse known .csv databases using a hashmap with key info from the user profile to create a list of destination recommendations that is made of country objects.

## C.5 - Country

A class that contains information, based on user nationality, on the countries' Visa, Name, and Capital City.

## T.1 - Error Handling

If the user enters invalid information into the scanner, the user will be displayed an error message and prompted to re-enter correct information.

## I. 2 - GUI

The GUI element of our application can be divided into three main components. There is a login component where the user can enter their login information or create a new user. Second, there is a menu component, this feature will be displayed on the side of the application. Thirdly, there is the menu option view panel where users can switch between Destination View, Favorites View, and Update User View.

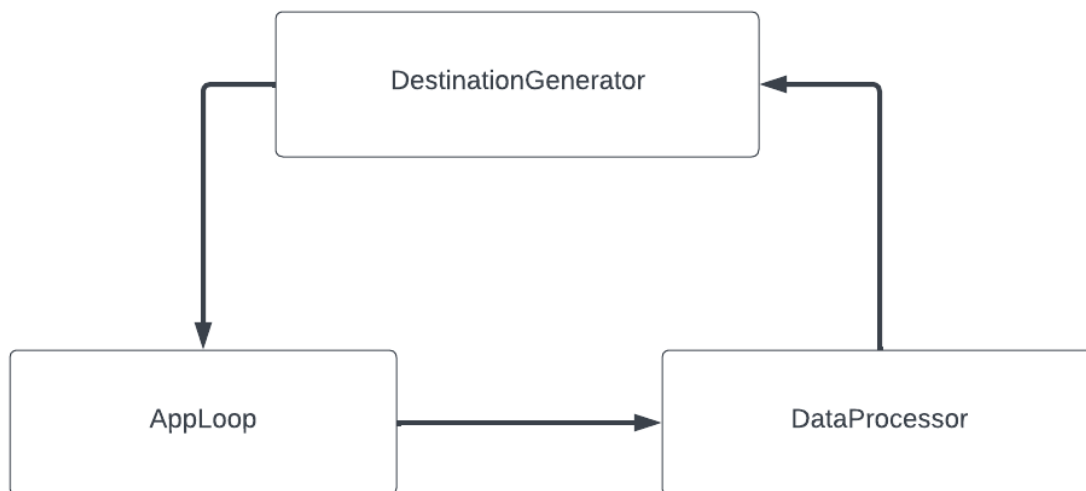
# System Design

This section describes the system design in detail. An overview of the major system components and their roles will be provided, followed by more detailed discussion of components. Component discussion will reference the technical features the component helps satisfied.

**Note:** Design co-evolves with implementation. It is important to start with some sense of components and their interactions, but design docs should be updated during implementation to capture the actual system as-built.

## Architecture Overview

### High-level System Architecture



### Major Components

#### InputAdapter

I.1, I.2

User interaction is handled via the InputAdapter. When NOMAD Travels is started, depending on how it is started, only the CLI or GUI will run. The Input Adapter will delegate if the application should be run on the GUI or CLI. This component uses the Strategy pattern to provide the functionality to select CLI or GUI operation.

#### MainApp

I.1, C.1, T.1

User interaction is handled via the MainApp. MainApp runs the basic event loop for our travel application. The methods used within the MainLoop class to run the application are start(), launch(), changeScence().

## User

The User object works on the Singleton design pattern and holds all the user data at each given time.

## Country

The Country class stores the country data and works closely with the controllers to display country information in the views.

## DestinationGenerator

C.2, C.3, C.4

The Destination Generator is a component in the NOMAD Travel app responsible for generating a list of destination options based on the user's preferences and criteria. It implements the Strategy Pattern such that it uses the user profile data like their nationality and generates a recommended list of destinations based on visa availability and travel regulation information. The DataGenerator class uses a User object, a CountryModel object and uses the methods generate(), order(), and getDestinations(). The generated list is then presented to the user, who can select a destination from the list to learn more about it or favorite it for future references.

## Model

C.2, C.4

The Model component has the CountryModel, UserModel, and DatabaseConnectionModel to store and retrieve data from the database. It works with the Controllers and processes data requests.

## View

The View component displays the information to the user and has interactivity for a better user experience.

## Controllers

The Controller component sends data requests to the Model and displays the data in the View component.

# Detail Design

## InputAdapter

The InputAdapter component contains the logic for generating output to the display and receiving input from the user. Based on the arguments provided when TAP is started, the GameMaker will instantiate the correct concrete InputAdapter and provide that adapter to the GameEngine.

## InputAdapter

ux.1, ux.2

The InputAdapter is an interface that is implemented by the TextInterface and GraphicInterface classes. The concrete TextInterface and GraphicInterface classes will be responsible for the details of interaction with the user.

## TextInterface

ux.1

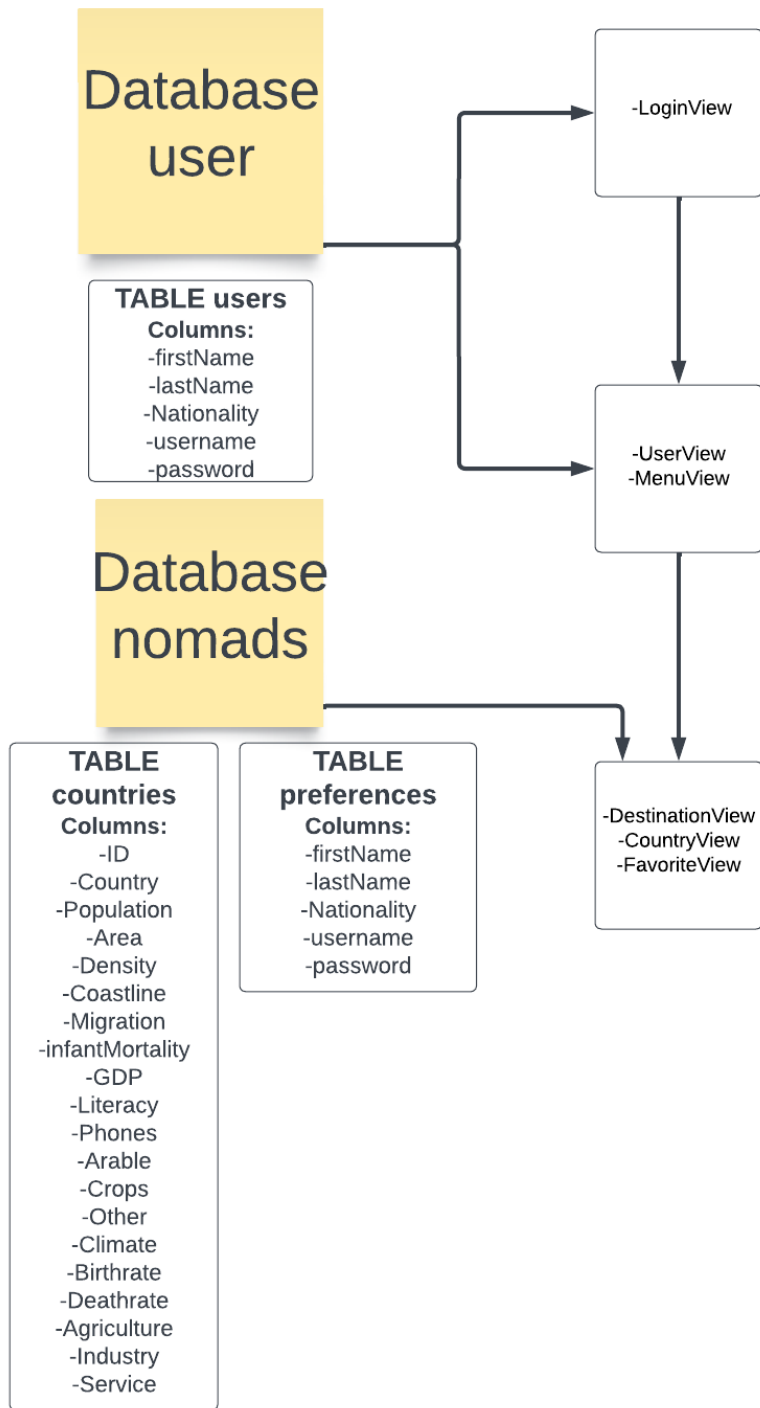
The TextInterface implements a strategy for CLI operation. CLI operation is fairly straightforward and can likely (but does not have to) be implemented in a single class.

NOTE: if multiple classes are used the SDD must be updated to include these additional classes.

## GraphicInterface

ux1, ux2, ux3, ux4, ux5

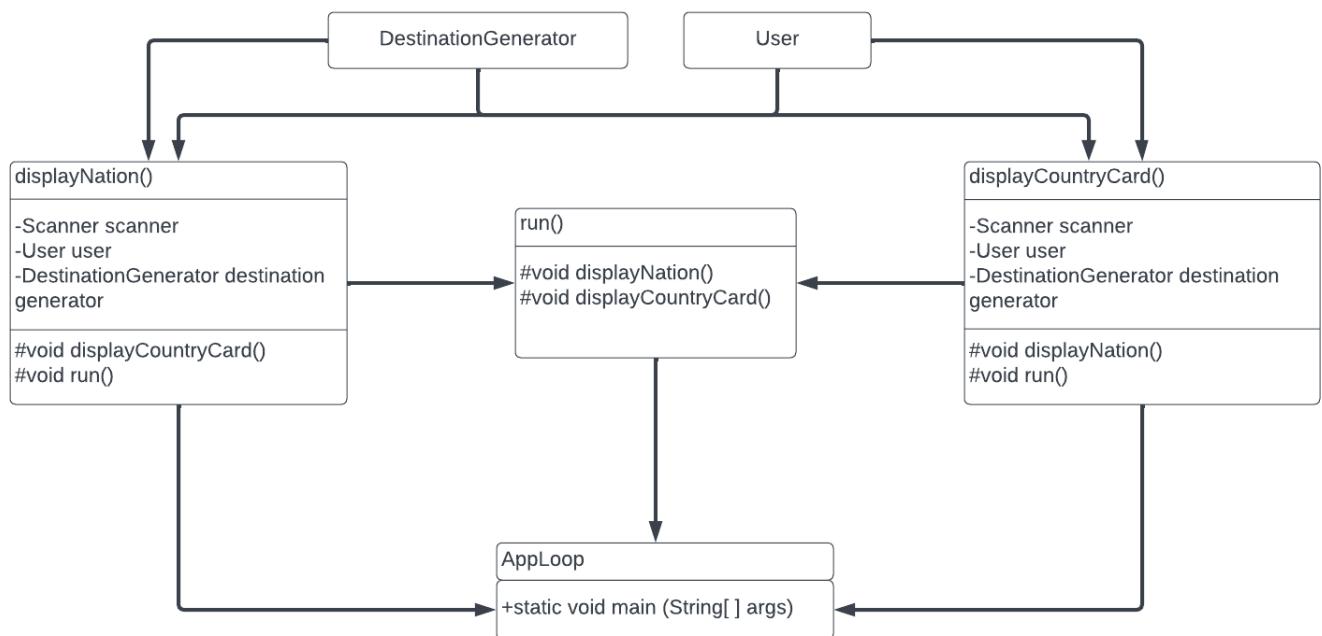
The GraphicInterface implements a strategy for GUI operation. It uses JavaFX for the GUI implementation. We will use xml files for the following Views: LoginView, MenuView, DestinationView, CountryView, FavoriteView, UserView. We will have separate controllers for each view.



## MainApp

The MainApp is the central component of NOMAD Travel that manages the basic event loop and handles user interactions. When the MainApp class starts, the main method runs the launch() method which starts the Login View and takes the user to the login screen. It keeps the whole application running. It has a property called 'countries' that holds the list of countries this application supports. It also has a method called changeScene which changes to the next desired View depending on the button clicked.

## MainApp



## User Interface

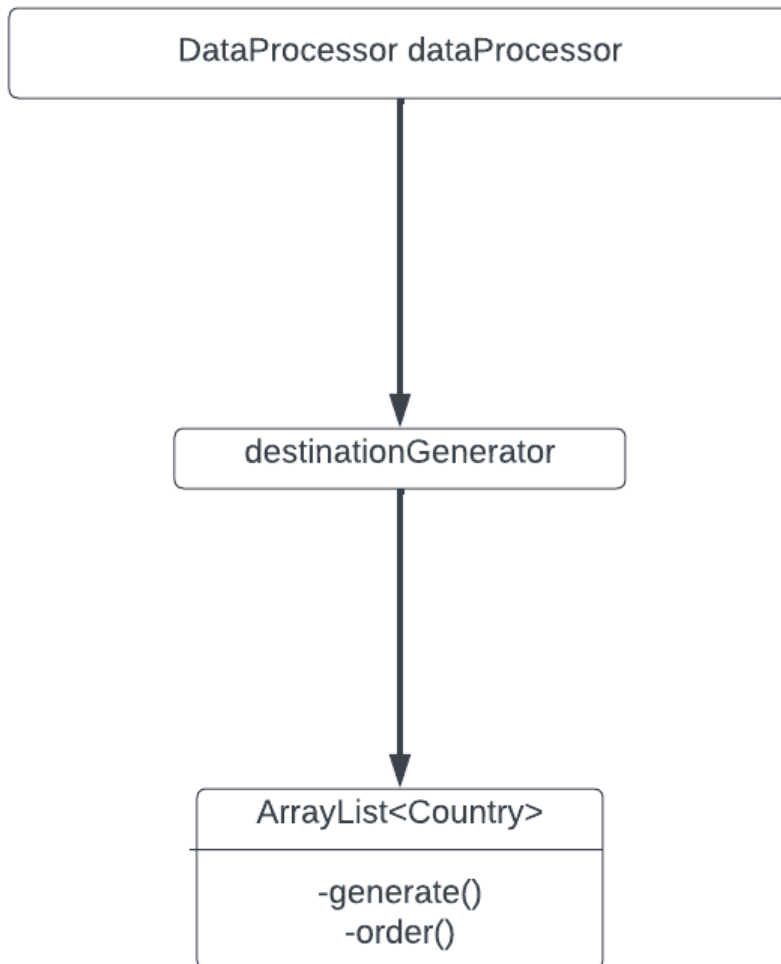
The user interface for NOMAD Travel is a Command Line Interface (CLI) that displays messages, prompts, and menus to the user and accepts input from the user. From the InputAdapter, they can also choose GUI and interact with the JavaFX application.

## DestinationGenerator

The Destination Generator component in the NOMAD Travel app is responsible for generating a list of destination options that match the user's preferences and criteria. It implements the Strategy Pattern, which allows different algorithms to be used for generating the list of destinations based on user input or other factors. DestinationGenerator uses a CountryModel

and User Object to run the methods generate(User user) to generate a list of destinations or country object items to display to the user based on a ranking algorithm.

## DestinationGenerator



## Generate

Generate is a method that generates a list of recommended travel destinations based on various factors, including the user's nationality, visa availability, and travel regulation information.

## Order

Order is a method that sorts the list of recommended destinations based on visa availability for the user.



## User

The User class represents a user of a travel application. This class has several properties such as firstName, lastName, nationality, username, password, and boolean values that indicate the user's preferences for outdoor activities, urban environments, cultural experiences, and food. The class also has an ArrayList of Country objects, which represent the user's favorite countries and another ArrayList of Country objects that represent the user's travel destinations. The User class is designed to be a singleton class, meaning only one instance of this class can be created throughout the application.

The getInstance() method returns the only instance of the User class because of the Singleton design pattern. The class has several methods that allow the user to add or remove favorite countries and set travel destinations.

## Country

The Country class represents a country object in the travel application that provides recommendations to users based on their preferences. The class is designed to store information about a country's name, region, population, area, and visa requirements. It also stores information about the user's preferences related to the country's outdoors, urban, cultural, and food experiences.

The Country class is designed to implement the Comparable interface, allowing it to be sorted based on its rank. The rank is calculated based on the country's visa requirement and the user's preferences. The compareTo method returns 1 if the current object's rank is greater than the argument object's rank, -1 if it is less, and 0 if they are equal.

The setVisaRank method takes a visa string as input and sets the visa attribute of the Country object to a human-readable value. It also returns a value based on the type of visa, which is used to calculate the rank. The setPreferenceRank method calculates the preference rank of the Country object based on the user's preferences.

The Country class has a constructor that takes a visa, name, region, population, and area as inputs. These inputs are used to set the corresponding attributes of the object. The rank is calculated using the setVisaRank and setPreferenceRank methods.

The Country class has an updatePreferences method that takes integers representing the user's preferences for outdoors, cultural, food, and urban experiences, and a description string. These inputs are used to update the outdoors, cultural, food, urban, and description attributes of the object.

## Model

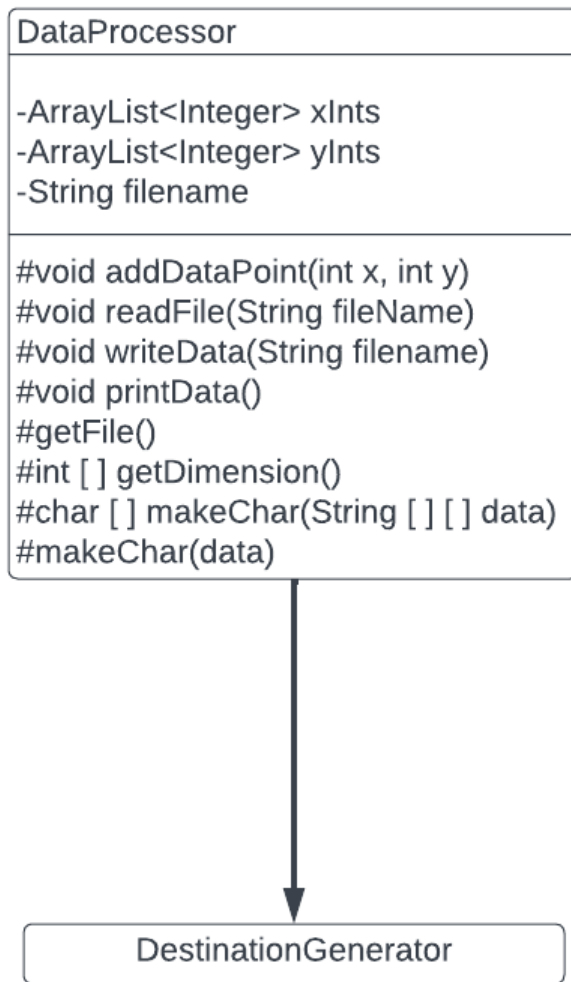
The Model component defines three models: UserModel, CountryModel, and ItineraryModel according to the MVC design pattern. These models are responsible for handling user authentication, visa and travel information retrieval, and itinerary storage, respectively.

## UserModel

Responsible for handling user login, registration, and data update. It uses the DatabaseConnectionModel class to establish a connection to the database and execute SQL statements. The validateLogin() method verifies the user's login credentials by querying the database. The updateUserAtLogin() method retrieves the user's data from the database and stores it in the User singleton instance. The updateUserAfterLogin() method updates the user's data in the database. The registerUser() method creates a new user account in the database.

## CountryModel

Retrieves visa and travel information from a CSV file and a database, respectively. The updateVisaInfo() method reads the CSV file and creates a hashmap that stores visa information for each country. The hashmap is then filtered to include only the countries in the target list. The updateGeneralInfo() method queries the database and retrieves general information about each country in the target list. The retrieved data is used to create an ArrayList of Country objects.



## View

The View component in the MVC design pattern is responsible for displaying the user interface and handling user interactions. It communicates with the Model component to retrieve and update data, and with the Controller component to handle user input and application logic.

### login-view

This view displays a login screen where the user enters their username and password. When the user submits their credentials, the view sends a request to the Controller to authenticate the user. If the user is authenticated, the Controller will update the Model with the user's information and redirect the user to the destination-generator-view.

### register-view

This view allows the user to register a new account by entering their first name, last name, username, password, nationality, and preferences. When the user submits their registration information, the view sends a request to the Controller to create a new user account in the Model. If the registration is successful, the Controller will redirect the user to the destination-generator-view.

### destination-generator-view

This view displays a list of personalized recommendations for the user based on their registration information and preferences. The view retrieves this information from the Model through the Controller. The user can click on a destination to see more information about it.

### favorites-view

This view displays a list of countries that the user has favorited. The view retrieves this information from the Model through the Controller. The user can remove countries from their favorites by clicking on a button.

### update-user-view

This view allows the user to update their user information, such as their password or preferences. When the user submits their updated information, the view sends a request to the Controller to update the Model with the new information.

### country-card-view

This view displays detailed information about a selected country, such as its flag, name, description, characteristics, area, region, and population. This view is accessed when the user clicks on a destination in the destination-generator-view.

## Controller

### CountryCardController

This CountryCardController functions as the controller for the country-card-view.fxml javafx and scene builder file used to create the Country card GUI for our project. The Controller implements Initializable, and user interactions to be able to dynamically change the GUI while simultaneously storing data generated by user interaction, such as adding destinations to favorites from the CountryCardController.

### DestinationGeneratorController

This DestinationGeneratorController functions as the controller for the destination-generator-view.fxml javafx and scene builder file used to create the Destination Generator, or search destinations view in GUI for our project. The User is taken to the

Destination Search View after logging in and setting preferences. This requires DestinationGenerator Controller to be able to store data and visually change in the GUI simultaneously. Action Listeners are placed onto Country list items within the ListView of the Destination Generator to be able to allow the user to interact with the country cards.

### FavoritesController

This FavoritesController functions as the controller for the favorites-view.fxml javafx and scene builder file used to create the Favorites Scene, or favorite destinations view in GUI for our project. The user must be able to interact with the favorite countries dynamically, just by adding and removing them, and this must also store to the user data.

### LoginController

The LoginController is the controller for the login-view.fxml for the login screen of our application. It implements a connection to our MySQL database of users. The LoginController utilizes a verifyuser() function to check user input in the GUI textfield to information in the userdatabase to verify the user's existence, and then move it into the application.

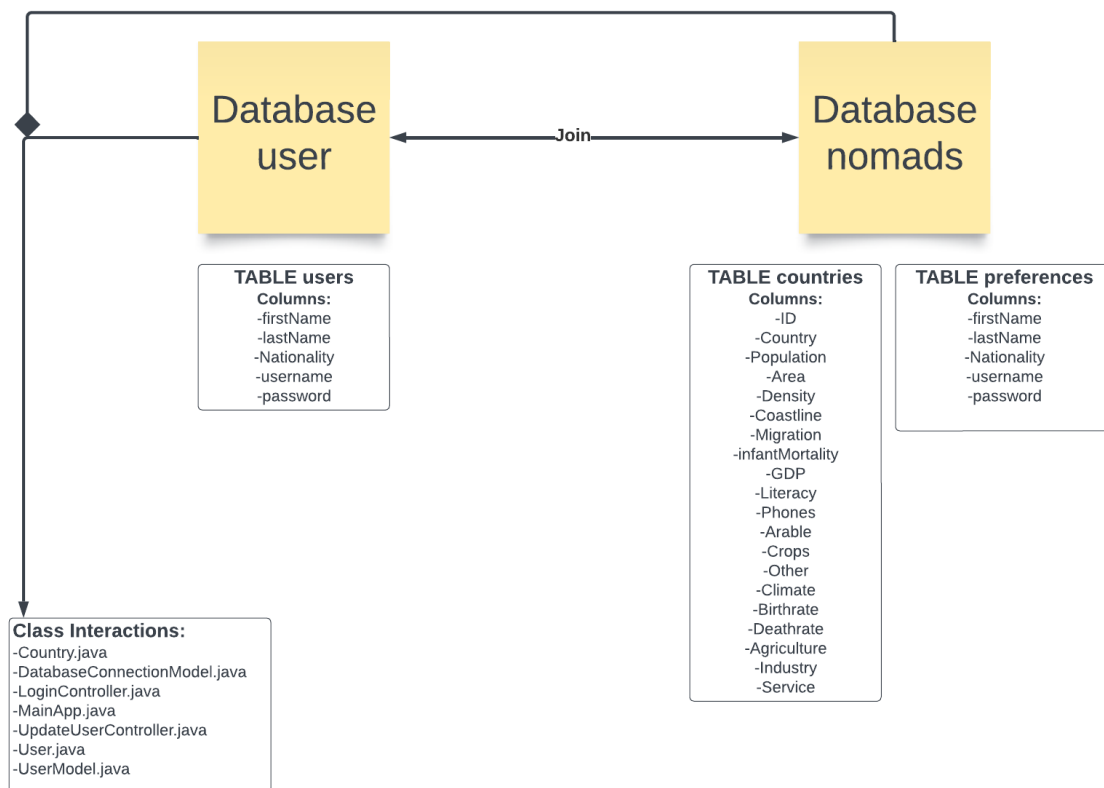
### RegisterController

The RegisterController is the controller for the register-view.fxml file that handles new user registration. The controller utilizes a textfield in the GUI to create a new User instance in the User database within MySQL.

### UpdateUserController

The UpdateUserController is the controller for the update-user-view.fxml file and utilizes a GUI user interaction from textfield and button clicks to allow the user to edit and update their information, for example, nationality and preferences. The UpdateUserController also updates the user information in our MySQL User database.

## Data Formats



## User Database

The user profile is a table in the database file that includes user information such as the user name, nationality, first name, last name, password, and preferences..

### Format

Username, Password, Nationality, F\_Name, L\_Name, Preferences

## Preferences Database

This database stores information about characteristics of countries that match with the preferences column of the user database.

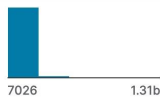

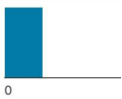
### Format

Country, Outdoors, Urban, Cultural, Food, Description

# Countries Database

The Country File is another table in the database that includes information stored in the ArrayList<Countries> Destination recommendations created by the Dataprocessor and DestinationGeneration classes. The format of the data Countries is found in the Database example below, and a table example is also provided below.

## Example Country Data

Country	Region	Population	Area (sq. mi.)	Pop. Density (per
227 unique values	SUB-SAHARAN AF... 22%			
	LATIN AMER. & C... 20%			
	Other (131) 58%			
Afghanistan	ASIA (EX. NEAR EAST)	31056997	647500	48,0
Albania	EASTERN EUROPE	3581655	28748	124,6
Algeria	NORTHERN AFRICA	32930091	2381740	13,8
American Samoa	OCEANIA	57794	199	290,4
Andorra	WESTERN EUROPE	71201	468	152,1
Angola	SUB-SAHARAN AFRICA	12127071	1246700	9,7
Anguilla	LATIN AMER. & CARIB	13477	102	132,1
Antigua & Barbuda	LATIN AMER. & CARIB	69108	443	156,0
Argentina	LATIN AMER. & CARIB	39921833	2766890	14,4
Armenia	C.W. OF IND. STATES	2976372	29800	99,9
Aruba	LATIN AMER. & CARIB	71891	193	372,5
Australia	OCEANIA	20264082	7686850	2,6