# PizzaFactoryLocator

This application is the finished product of the Pizza Delivered Quickly Factory Locator Subsystem. This module was written in Elixir and as such has been compiled to Erlang byte code. Provided below is the documentation on how to use this module.

Reference:
https://people.uwec.edu/sulzertj/Teaching/is455/Resources/PizzaDeliveryQuickly_Case_Study.pdf

## Command Line API Documentation

Run the following commands within the project directory's root using the current system's shell. If you are using an operating system other than linux then replace 'linux' in the command's path with your operating system before executing. Supported operating systems are:

- linux
- windows
- macos

Commands will require slight modifications for windows systems.

Windows Command Sample (Get Configuration):

```
_build\release\windows\rel\pizza_factory_locator\bin\pizza_factory_locator eval
"{:ok, config} = PizzaFactoryLocator.get_config(); {:ok, config} = config |>
Map.from_struct() |> Jason.encode(); IO.puts(config);";
```

### 1. Set Configuration

Updates the configuration. Enter the corresponding information of the mongo database server.

```
_build/release/linux/rel/pizza_factory_locator/bin/pizza_factory_locator eval
'''
PizzaFactoryLocator.set_config(
  mongo_database_address,
  mongo_database_name,
  mongo_database_username,
  mongo_database_password,
  mongo_database_port,
  orders_collection_name,
  factories_collection_name
);
''';
```

**Parameters**

- mongo_database_address - The IP address of the Mongo Database server
- mongo_database_name - The name of the Mongo Database
- mongo_database_username - The Auth Username of the Mongo Database
- mongo_database_password - The Auth Password of the Mongo Database
- mongo_database_port - The port number of the Mongo Database Server
- orders_collection_name - The name given to the collection used to store pizza orders

- factories_collection_name - The name given to the collection used to store factories

## 2. Get Configuration

This command reads the configuration from file and prints it to the console.

```
_build/release/linux/rel/pizza_factory_locator/bin/pizza_factory_locator eval
'''
{:ok, config} = PizzaFactoryLocator.get_config();
{:ok, config} = config |> Map.from_struct() |> Jason.encode();
IO.puts(config);
''';
```

## 3. Save Order

Saves the coordinates of an order to the database

```
_build/release/linux/rel/pizza_factory_locator/bin/pizza_factory_locator eval
'''
Application.ensure_all_started(:mongodb);
{:ok, _} = Database.connect();
{:ok, coordinates} = Coordinates.constructor(
  x_coordinate,
  y_coordinate
);
{:ok, order} = Order.constructor(
  coordinates
);
{:ok, _} = Database.save_order(order);
''';
```

**Parameters**

- x_coordinate - The x coordinate of the order
- y_coordinate - The y coordinate of the order

## 4. Save Factory

Saves a Factory to the database.

```
_build/release/linux/rel/pizza_factory_locator/bin/pizza_factory_locator eval
'''
Application.ensure_all_started(:mongodb);
{:ok, _} = Database.connect();
{:ok, coordinates} = Coordinates.constructor(
  x_coordinate,
  y_coordinate
);
{:ok, factory} = Factory.constructor(
  factory_name,
  coordinates,
  phone_number
);
{:ok, _} = Database.save_factory(factory);
''';
```

**Parameters**

- x_coordinate - The x coordinate of the factory
- y_coordinate - The y coordinate of the factory
- factory_name - The name of the factory
- phone_number - The phone number of the factory

## 5. Determine New Pizza Factory Location

Reads all the Pizza Orders from the database and, using the coordinates for each order, calculates the ideal location to place a new pizza factory.

```
_build/release/linux/rel/pizza_factory_locator/bin/pizza_factory_locator eval
'''
Application.ensure_all_started(:mongodb);
{:ok, _} = Database.connect();
# UNCOMMENT THE LINES BELOW TO USE OPTIONAL PARAMS
# {:ok, boundary_start} = Coordinates.constructor(
#   boundary_start_x,
#   boundary_start_y
# );
# {:ok, boundary_stop} = Coordinates.constructor(
#   boundary_stop_x,
#   boundary_stop_y
# );
##################################################
# REMOVE THE LINES BELOW TO USE OPTIONAL PARAMS
boundary_start = nil;
boundary_stop = nil;
##################################################
new_location = PizzaFactoryLocator.determine_new_factory_location(
  boundary_start,
  boundary_stop
);
{:ok, new_location} = Jason.encode(new_location);
IO.puts(new_location);
''';
```

**Parameters**

- boundary_start_x - (Optional) The start x coordinate of a new factory area
- boundary_start_x - (Optional) The start y coordinate of a new factory area
- boundary_stop_x - (Optional) The stop x coordinate of a new factory area
- boundary_stop_y - (Optional) The stop y coordinate of a new factory area

## 6. Get Closest Factory

Gets the nearest factory to supplied coordinates. Calculations are based on the distance between coordinates and not the length of the route to the factory.

```
_build/release/linux/rel/pizza_factory_locator/bin/pizza_factory_locator eval
'''
Application.ensure_all_started(:mongodb);
{:ok, _} = Database.connect();
{:ok, coordinates} = Coordinates.constructor(
  x_coordinate,
  y_coordinate
```

```
);
result = PizzaFactoryLocator.get_closest_factory(coordinates);
(length(result) > 0 &&
    (
      {:ok, result} =
        result
        |> Enum.at(0)
        |> Map.from_struct()
        |> Map.merge(%{:distance => Enum.at(result, 1)})
        |> Jason.encode();
      IO.puts(result);
    )) || IO.puts("null");
''';
```

**Parameters**

- x_coordinate - The x coordinate of the origin
- y_coordinate - The y coordinate of the origin

## 7. Get Factories

Gets a slice of factories from the database

```
_build/release/linux/rel/pizza_factory_locator/bin/pizza_factory_locator eval
'''
Application.ensure_all_started(:mongodb);
{:ok, _} = Database.connect();
# UNCOMMENT THE LINES BELOW TO USE OPTIONAL PARAMS
# {:ok, boundary_start} = Coordinates.constructor(
#   boundary_start_x,
#   boundary_start_y
# );
# {:ok, boundary_stop} = Coordinates.constructor(
#   boundary_stop_x,
#   boundary_stop_y
# );
################################################
# REMOVE LINES BELOW OUT TO USE OPTIONAL PARAMS
boundary_start = nil;
boundary_stop = nil;
################################################
{:ok, factories} = Database.get_factories(
  start_index,
  stop_index ,
  boundary_start,
  boundary_stop
);
factories = Enum.map(factories, fn factory ->
  try do
    Map.from_struct(factory);
  rescue
    _ -> nil;
  end
end);
{ok, factory_json} = Jason.encode(factories);
IO.puts(factory_json);
''';
```

**Parameters**

- start_index - An integer as the start index of factory slice from database

- stop_index - An integer as the stop index of factory slice from database
- boundary_start_x - (Optional) The start x coordinate of the area to get factories from
- boundary_start_x - (Optional) The start y coordinate of the area to get factories from
- boundary_stop_x - (Optional) The stop x coordinate of the area to get factories from
- boundary_stop_y - (Optional) The stop y coordinate of the area to get factories from