dotNet5782_9171_7973

Bonus Review

General

New c# Features

- Record
- Switch Expression
- Tupples
- Init Only Setters
- Using Statement
- Range Operator

Dal

Structure

We have implemented the layers model in the second structure (The Bonus structure), So we have a config file which follows the given format:

```
<config>
    <dal>[chosen-dal]</dal>
    <dal-packages>
        <[package-1]>
            <class-name>[package-1-class-name]</class-name>
            <namespace>[package-1-namespace]</namespace>
        </[package-1]>
        ...
    </dal-packages>
</config>
```

This format allows to specify the namespace in addition to the class name.

Logic Deletion

The dal deletion is just a *Logic Deletion* rather than *Real Deletion*. All the `Dal` entities implement the interface `IDeletable` which consists of just one property as follows

```
interface IDeletable
{
    bool IsDeleted { get; set; }
}
```

From now, *deletion* is changing the entity's `IsDeleted` property to `true`. Only non-deleted entities are allowed to perform actions.

Extensive Use of Generic

In order to Avoid repetition according to the **DRY** principle, We implemented all our `Dal` methods as generic methods. So, Instead of having `AddDrone`, `AddParcel`, `AddBaseStation` and `AddCustomer` for example, We only have `AddItem<T>` method.

PL

Custom Window Layout - Docking

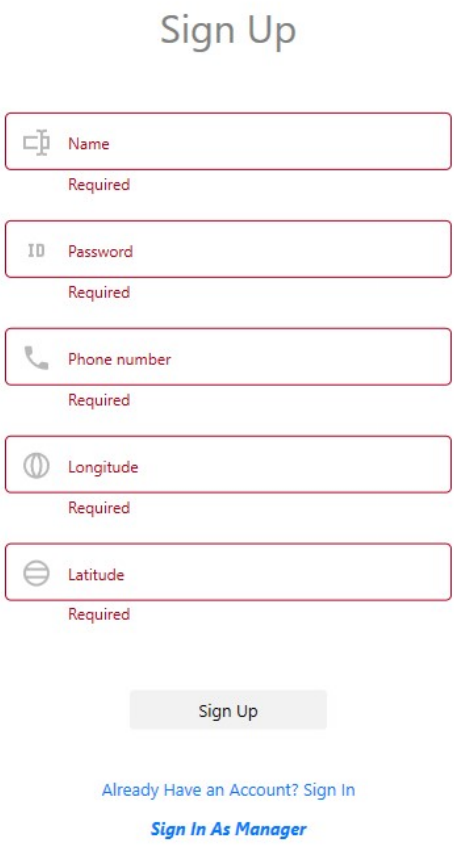
Layout is very flexible and easy to use.

Regular Expression

We used `Regex` for validity checking. [example]

User Interface

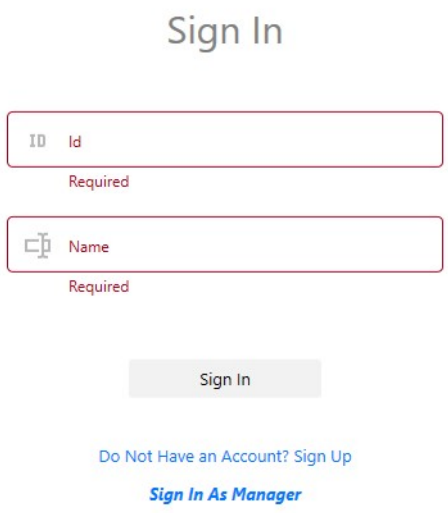
Our project supports two modes: customer mode and manager mode. When running the program the following screen shows up:



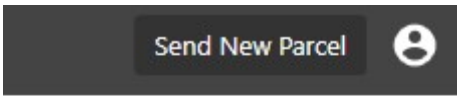
Then pressing the `Sign In As Managar` button enters the program in manager mode.

Signing up enters with a new customer account in customer mode.

Clicking on `Already Have an...` gives sign in page like this screen:



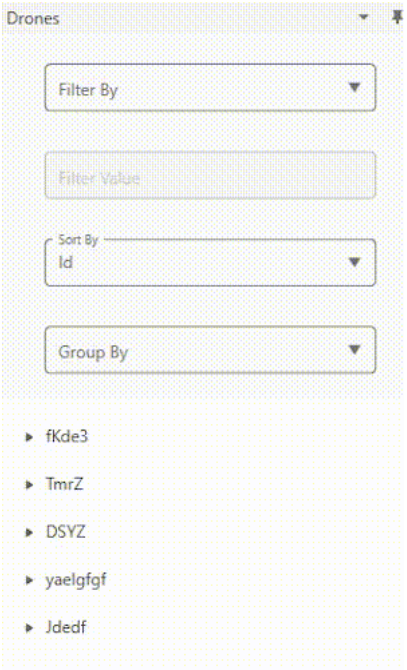
Clicking `Log Out` any time brings back to register window, where reconnecting is available again.



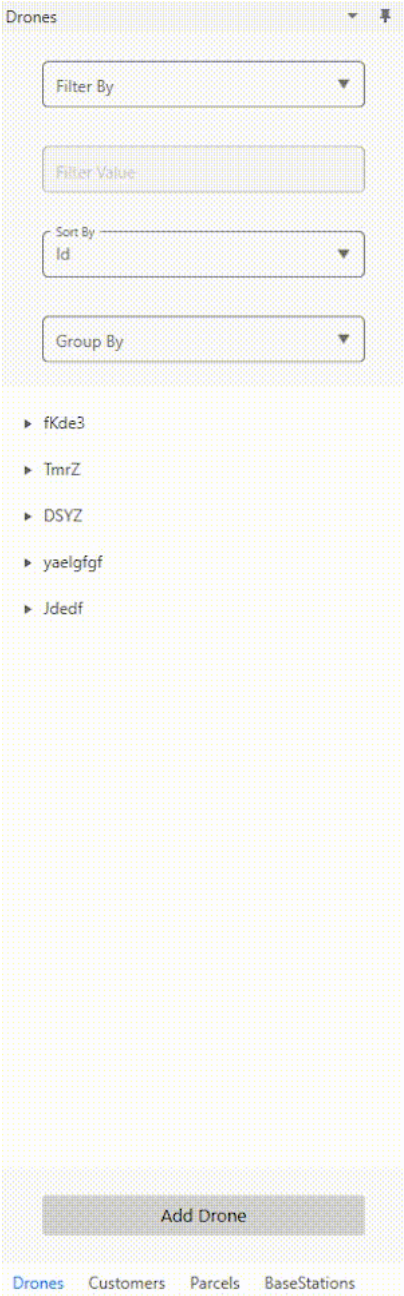
Full Support of All Data Queries

Very easy way to accsses accurate data. (uses reflection)

- Filter (notice the dynamic input)



- Sort and Group



MVVM

We used ***FULL*** MVVM, with full binding.

PO Entities

`PO` entities are mainly used as `Model`s in MVVM.

Custom `UserControl`

Extensive use in `UserControl` rather than `Window` control.

Makes the user experience better, and makes the application look better.

Miscellaneous

- Triggres
  - Event Trigger example
- Behaviors examples: definition use
- Converters example

- Commands (We implemented A `RelayCommand` class and used it as properties in our `PL` classes) example
- Data templates example
- `ObservableCollection` example
- `Collection View` example
- `Context Menu` example

External dictionary

We used dictionary for style definitions, outer `Data templates`, etc.

The application sends an email using `Smtp` object:

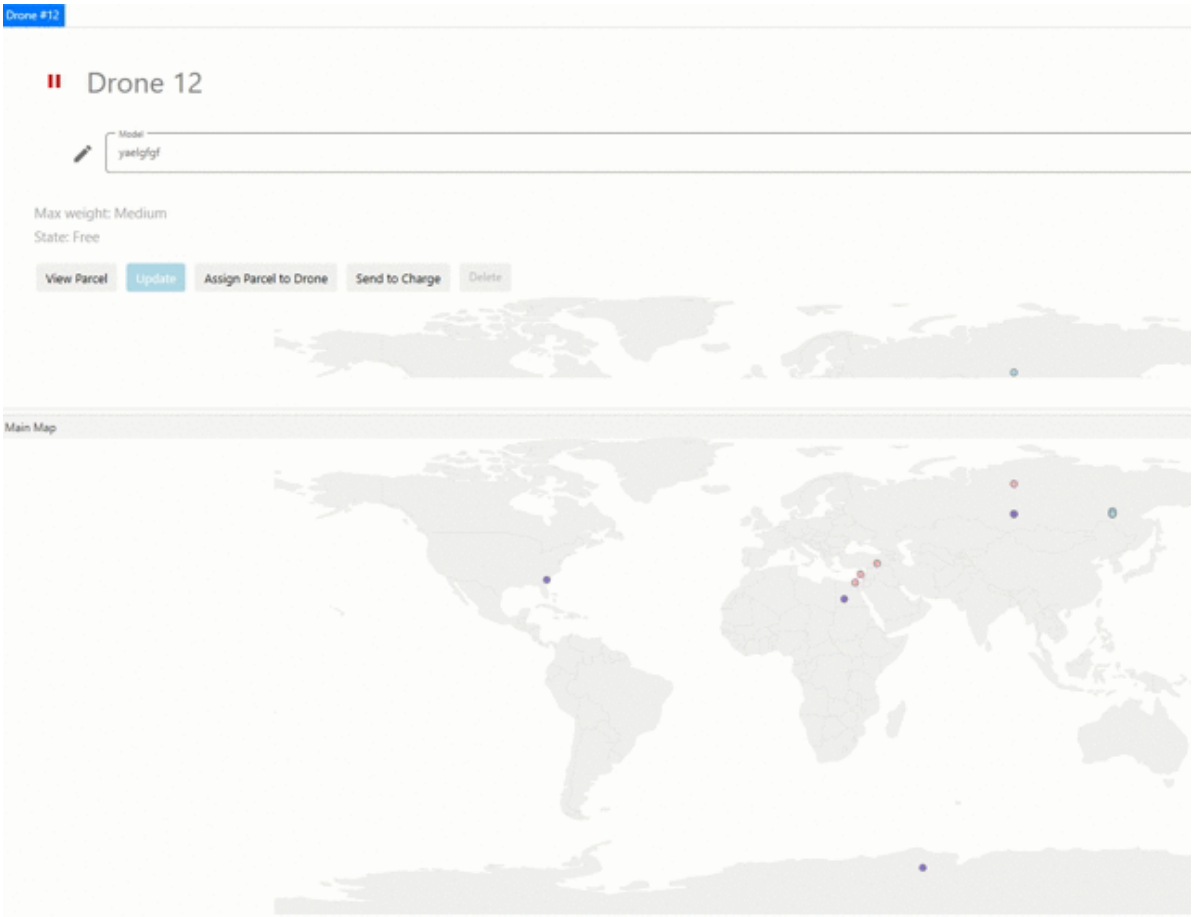
- To sender- when his parcel is sent
- To reciever- when he gets a parcel

see code

Simulator

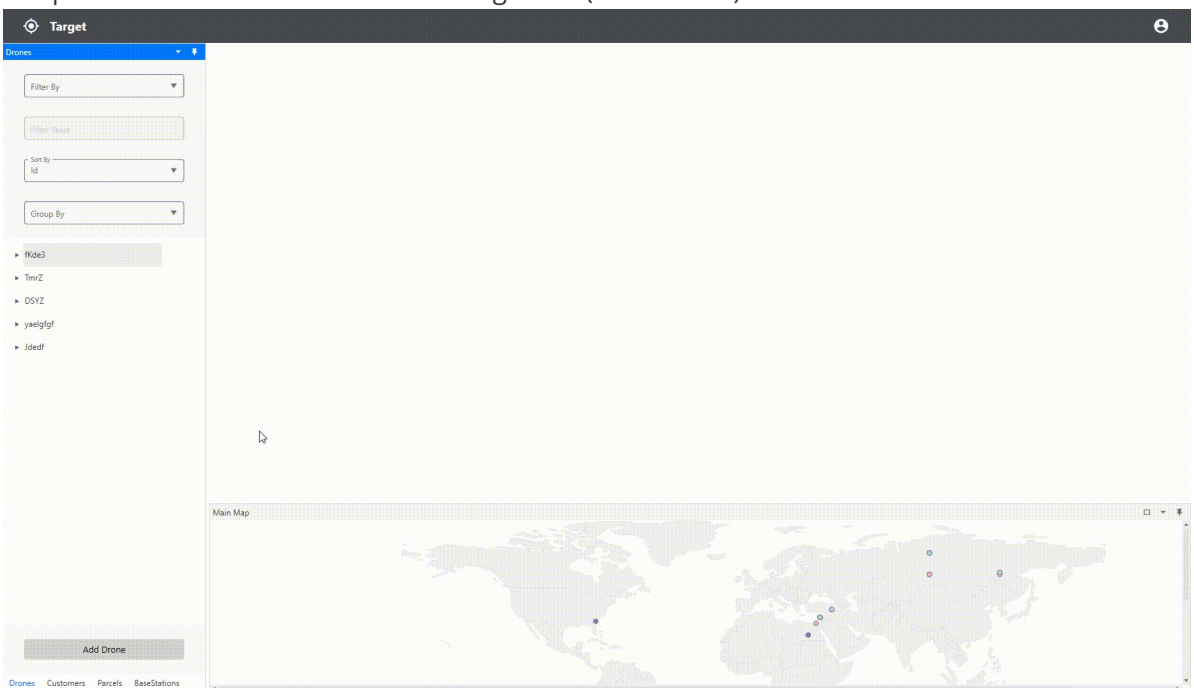
Location Update

Location updates in all related items while running.



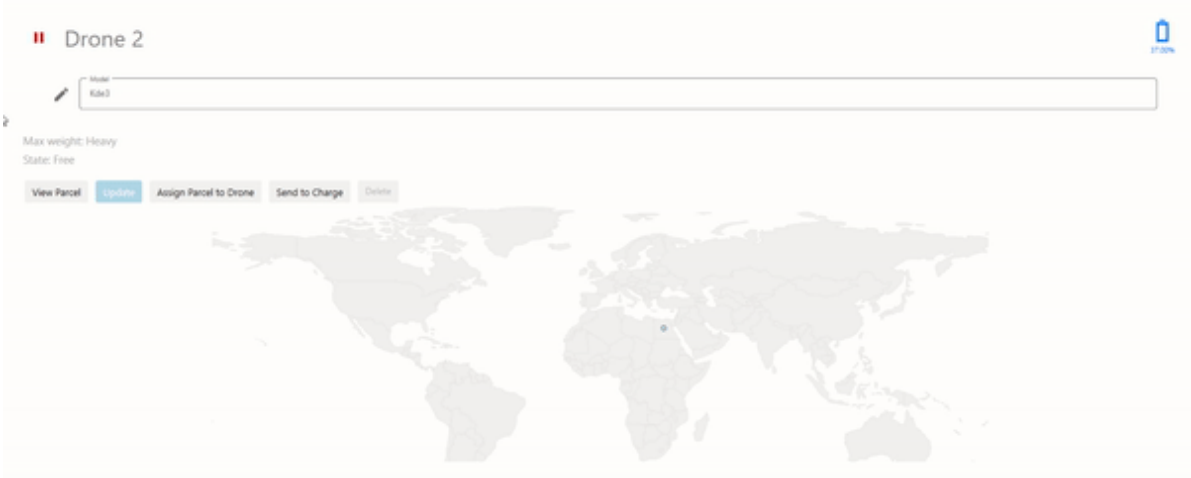
Parallel Activation

It is possible to run several simulators together. (not limitted)



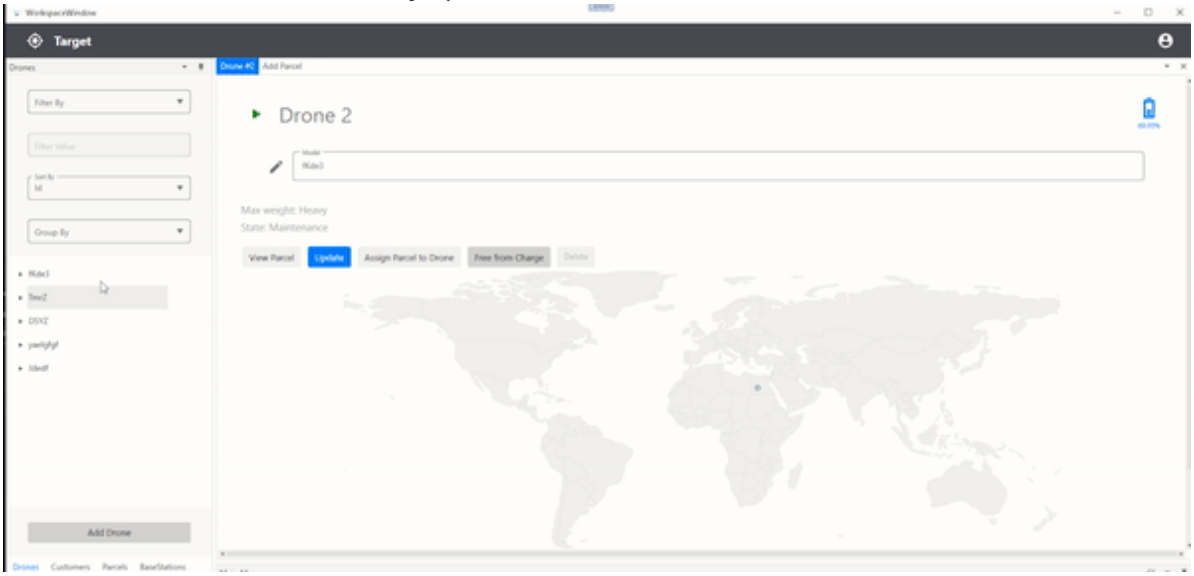
Busy Indicator

Busy indicator is on while simulator is about to stop its activity, after user's request.



Prevent Application Closing

The Application Prevent Closing As long as Simulators are On.
This is to make sure all data is fully updated.



Maps

Each entity has its map to represent its location, Besides, there is a `Main Map` for all the entities together.

Design patterns

Factory - Full structure

We used the full (bonus) structure for our `Factory`.
It finds the requested implementation of the servise contract and supplies its `Instance`.
To get it the following piece of code alone is necessary:

```
DalApi.IDal Dal { get; } = DalApi.DalFactory.GetDal();
```

Singleton

We implemented an abstract class `Singleton` which has lazy initialization and is thread-safe. The `Dal` and `BL` layers just inherit it.

Last But Not Least - Well Neat, Organized and Detailed `README`.