PokéOne WebApp - Implementation Documentation

Introduction

This document serves for noting down all considerations related to making detailed plans for how to implement the concept as specified in the master plan as well as for documenting all the actual implementation. This document is held in English in order to make it readable for the rest of the Unofficial PokéOne Guide team which will be owning the product and thus need to know how it was realized.

Terms related to PokéOne are not explained in this document, for that please refer to the appendix of the Master Plan / concept document. Technical terms and explanations are explained within the text, assuming the reader does know about the basics of object oriented programming and web applications.

Project Management

Version control

For version control GitHub is used, along with the GitFlow workflow. The repository can be found here: https://github.com/Finrod-Amandil/PokeOneWeb

Collaboration

During the first couple months, a team of four people is actively working on the realization of this project. In order to coordinate the work, the tool **ZenHub** is used. ZenHub is a browser extension for Google Chrome and Mozilla Firefox and adds a Tab containing KanBan board and progress reports right into the GitHub repository.

Continuous Integration

The project is continuously integrated: With every commit it is assured that the project can still be built. For that, the tool Travis CI is used.

The setup required some fiddling with the version of the SDK (Software Development Kit), until a matching combination with the chosen Runtime version was found. As of the start of the project the target SDK is .NET Core 2.1.500, matching to the .NET Core runtime 2.1.6. Automatic execution of unit tests was not yet implemented right away as none were available yet. This will be resumed later on.

Implementation Backend

Framework and programming languages

As Framework ASP.NET Core MVC 2.1 was chosen, as this is the framework with which the initiator and main developer of the project is most experienced and satisfied with. For the persistence, Entity Framework Core 2 is paired with an MSSQL Express Server.

Thus, the following programming languages are mainly involved in the development of this project:

- C#

- Razor / CSHTML
- CSS
- JavaScript

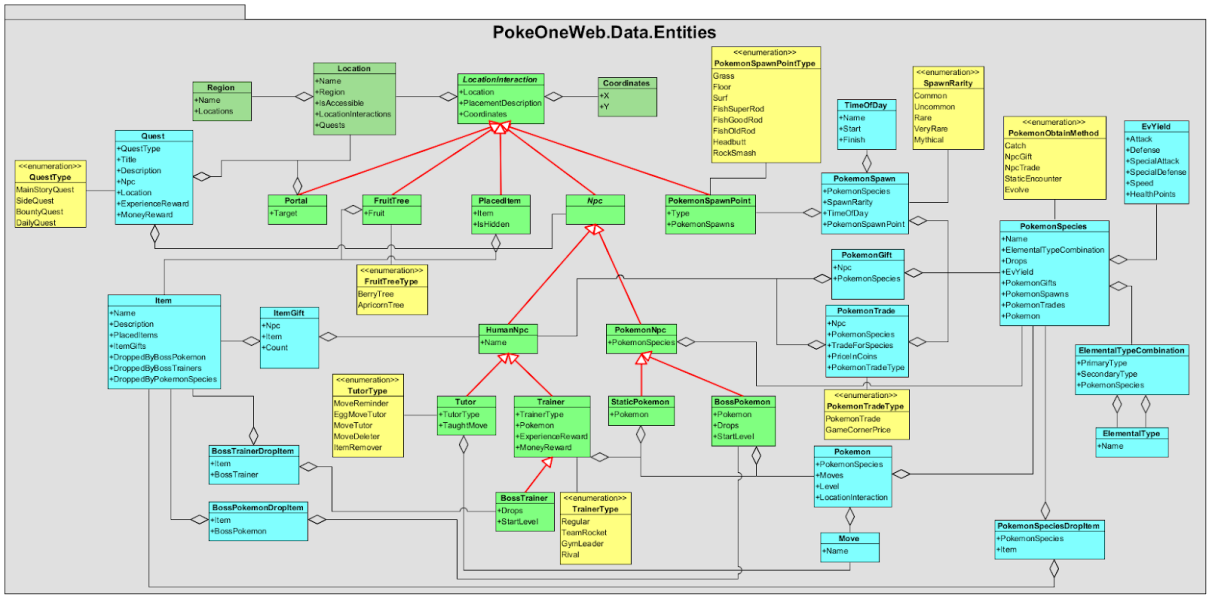
Business model and database.

The Pokémon Universe, which to some extent is the data baseline that is to be modelled by the database of this web application has grown over many years, and thus it can not really be avoided to deal with a rather large and complex data model.

As said above, Entity Framework Core (EF) is used for the persistence layer. EF is a so-called ORM, Object-Relational Mapper, which allows correlating a database structure to a set of C#-classes. With a technique called “EF Code First” it is possible to define the business model in C# and then generating a matching T-SQL database.

As porting the information of the existing Unofficial PokéOne Guide has the main priority, the first large part of the data model was designed to be able to take in all of the data presented in said Guide. Very early on the idea arose of partly using class inheritance to model some of the entities. These entities encompass all interactive objects that are placed within the PokéOne game map, such as NPC’s, items, and Pokémon spawn locations. All of these objects share certain properties, such as where they are placed (location and coordinates), and may want to be treated as one type of objects, while still having more specific properties. A quick research confirms, that EF is able to model inheritance to a relational database.

With this in mind, the following class structure was developed, with the inheritance tree highlighted in red:



Larger version: https://drive.google.com/file/d/1QUvgYYKyJtYtQ7IeB3MIr_Rs8dbBlfHY

As description of all the entities can be found further down the document.

After devising this first big part of the business model by creating a class diagram the respective classes were implemented. The next now is to let EF generate a database out of the class structure. As it was completely new to me to use inheritance with EF Code First I took the time to conduct some more profound research regarding how an inheritance tree can be translated to a table structure, as SQL does not know the concept of inheritance. Thus I figured that generally there are three principles

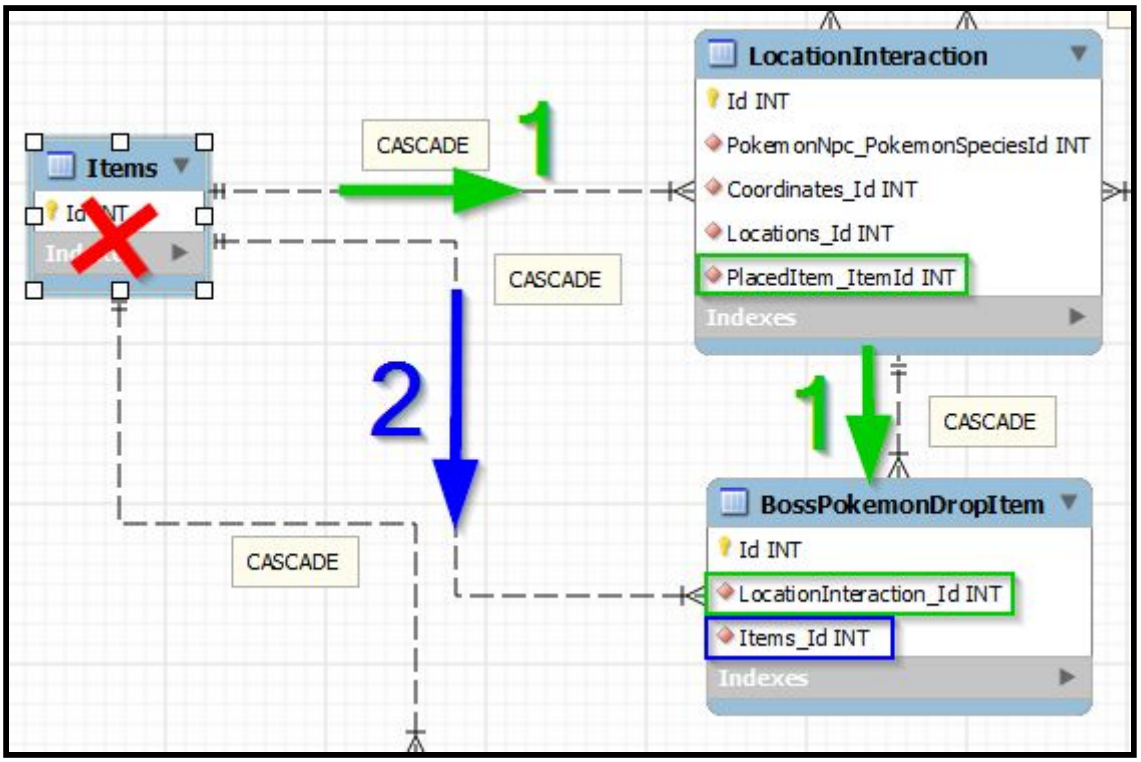
of modelling inheritance in a database: Table-per-Hierarchy (TPH), Table-per-Type (TPT) and Table-per-Concrete-Class (TPC). TPH collapses the entire inheritance hierarchy into one table with one column for every property appearing on any of the entities. An additional discriminator column is generated which allows determining of which type every entry actually is. With TPT, a separate table is created for every entity. That way the ERD looks very similar to the structure in the class diagram. With TPT, every entity is then spread across multiple tables, one entry for each level of the inheritance tree, and the tables are connected through 1:1 relations. The major drawback of TPT is performance. The last option, TPC, creates a table for each concrete (non-abstract) class. In each of these tables all inherited properties are included. Between these tables, no relations exist in the database. The drawback here is, that it is not possible to model, polymorphic structures, i.e. relating the base class with another entity, as no table for the abstract base class exists.

While the TPT would have initially been preferred due to it generating a more logical and solid database structure, TPH was chosen as this is the only modelling technique that was supported by EF Core. TPT would have been possible, but would have come with a decent amount of fiddling with the entity classes.

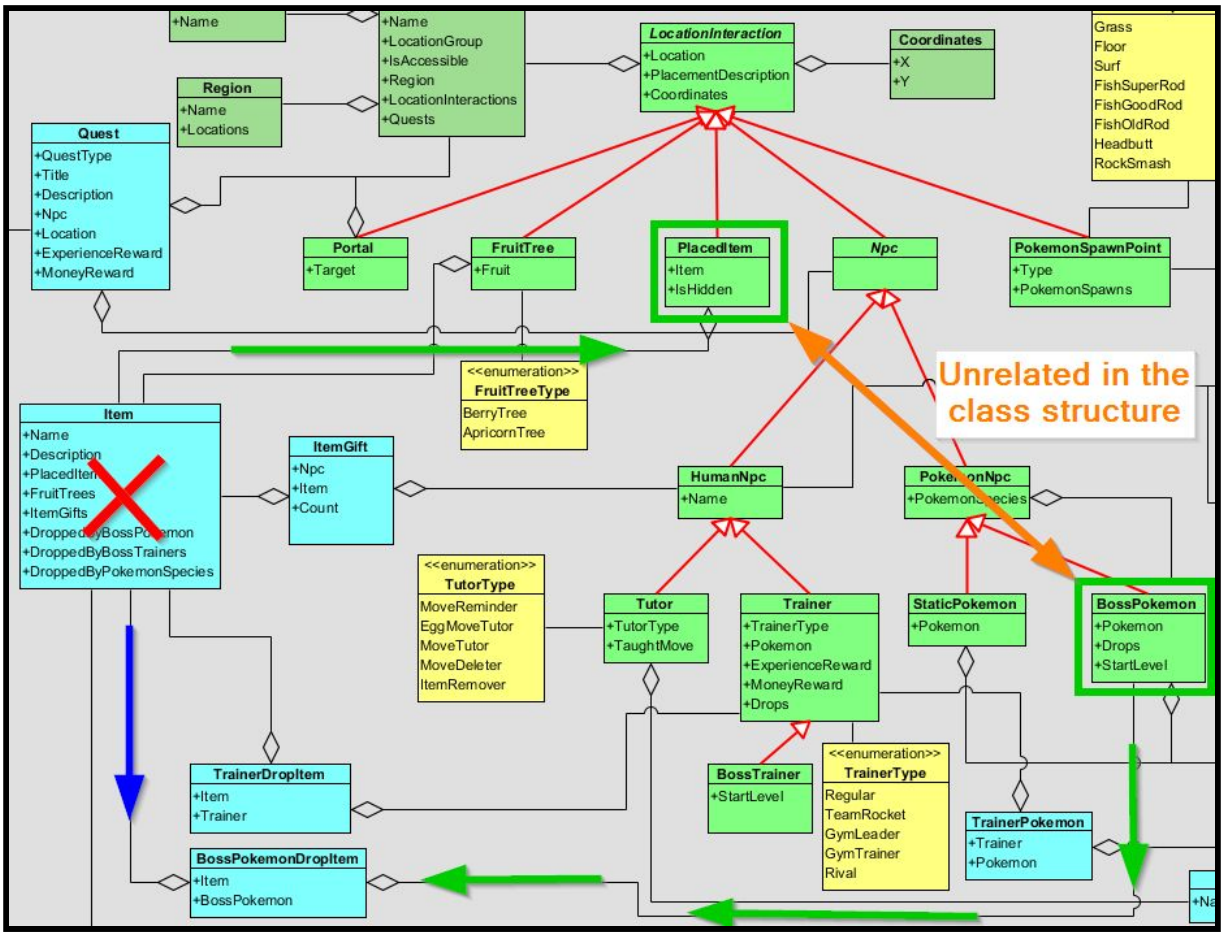
Unfortunately, generating the database kept returning errors:

```
fail: Microsoft.EntityFrameworkCore.Database.Command[20102]
      Failed executing DbCommand (50ms) [Parameters=[], CommandType='Text',
CommandTimeout='30']
      CREATE TABLE [BossPokemonDropItem] (
          [Id] int NOT NULL IDENTITY,
          [BossPokemonId] int NOT NULL,
          [ItemId] int NOT NULL,
          CONSTRAINT [PK_BossPokemonDropItem] PRIMARY KEY ([Id]),
          CONSTRAINT [FK_BossPokemonDropItem_LocationInteraction_BossPokemonId]
FOREIGN KEY ([BossPokemonId]) REFERENCES [LocationInteraction] ([Id]) ON DELETE
CASCADE,
          CONSTRAINT [FK_BossPokemonDropItem_Items_ItemId] FOREIGN KEY
([ItemId]) REFERENCES [Items] ([Id]) ON DELETE CASCADE
      );
System.Data.SqlClient.SqlException (0x80131904): Introducing FOREIGN KEY
constraint 'FK_BossPokemonDropItem_Items_ItemId' on table 'BossPokemonDropItem'
may cause cycles or multiple cascade paths. Specify ON DELETE NO ACTION or ON
UPDATE NO ACTION, or modify other FOREIGN KEY constraints.
```

Upon this, the class structure was once more examined in detail in order to ensure that no actual cyclic relations exist. As many classes are being reduced to one single table as TPH is applied, following the problem with the Cascade Paths as described in the error message was rather difficult. Thus an ERD of the database structure that was expected to be generated by EF was created, however excluding any fields that are not foreign keys. The tables of the ERD was created in the same order as EF was doing it, up until the point where the error occurred. After some considerations the cause of the error could be found: When deleting an entry in the “Items” table two cascade paths exist which both end at the “BossPokemonDropItem” table:



The reason that this error occurs is a direct consequence of applying TPH. Because, while it looks like the tables “Items” and “BossPokemonDropItem” are connected to the same kind of entity this is not actually the case: The entity `Item` in this case is related to `PlacedItem` (inheriting from the class `LocationInteraction`), while `BossPokemonDropItem` is related to an instance of `BossPokemon` (also inheriting from `LocationInteraction`). Between `BossPokemon` and `PlacedItem` there is no relation, but due to TPH, both are represented as an entry in the `LocationInteraction` table and thus, from a database perspective it would mean that deleting an “Items” entry with ON UPDATE CASCADE would result in deleting a “BossPokemonDropItem” entry as well as a “LocationInteraction” entry which then *may* result in yet another deletion of a “BossPokemonDropItem” entry, and thus when deleting an “Item” entry theres two cascade paths leading to the deletion of a “BossPokemonDropItem” entry which is not allowed to happen in a TSQL database.



After inspecting the logs generated by Entity Framework I notice that it seems to be the default of EF anyway to set relations to ON DELETE NO ACTION, only if the foreign key was specified explicitly (as was done on all entities of the inheritance tree) the relation type is set to ON DELETE CASCADE. Thus this problem is being solved by explicitly setting all relations between entities of the inheritance tree and other entities to ON DELETE NO ACTION. Reflecting on that, this may even be better than having the database automatically delete related entities as this may not be the intention in all cases. When deleting an Item for example, it may be better to keep the PlacedItem entity and just set the foreign key to null, signifying that the Placed Item still does / may exist, but it's currently unknown which item it holds.

```
modelBuilder.Entity<PlacedItem>()
    .HasOne(p => p.Item)
    .WithMany(i => i.PlacedItems)
    .HasForeignKey(p => p.PlacedItemId)
    .OnDelete(DeleteBehavior.Restrict);
```

The summary on the `DeleteBehaviour.Restrict` gives some more insight: "For entities being tracked by the `Microsoft.EntityFrameworkCore.DbContext`, the values of foreign key properties in dependent entities are not changed. This can result in an inconsistent graph of entities where the values of foreign key properties do not match the relationships in the graph. If a property remains in this state when `Microsoft.EntityFrameworkCore.DbContext.SaveChanges` is called, then an exception will be thrown." This is good to know. So as long as the database is only changed through EF, the database can not be corrupted by accident.

Import Google Spreadsheet

The currently used PokéOne Guide exists in form of a Google Spreadsheet. The amount of data contained within it is remarkably high. This results in some challenges:

- Adding the data of the spreadsheet into a database takes a long time if done manually.
- The Guide data is subject to change while the web application is being developed. These changes would need to be tracked and constantly reflected on the database.

It seems to be a good idea to solve these challenges with an automated solution. Thus it is decided to write an Import Service which automatically loads the contents of the Google Spreadsheet into the database. This also offers the possibility to already release the web application even with the option of editing the data through the web application still missing as it will be possible to just reload the Guide with this service.

The initial approach was to use OpenXML to import the Guide as Excel file. However this would require downloading the Guide as Excel file every time. Thus this idea was abandoned, when the new idea arose to check, whether Google offers an API to extract the spreadsheet directly off the internet. Sure enough, Google does have an API for this. That way it will be possible to place an admin-accessible button onto the webpage and simply by clicking that button it could be accomplished to update the database with the data held in the Google Spreadsheet.

While a lot of manual work may be avoided by the guide import service in the long run, some manual editing of the data is still required in order to make the spreadsheet structured enough to be able to export it programmatically. Due to it being possible that the spreadsheet will be continued to be used as primary data input, this normalized structure is specified explicitly below.

Specification of the normalized Unofficial PokéOne Guide Spreadsheet

The following specifications apply to the Unofficial Guide Google Spreadsheet and need to be followed in order to make it possible to export **all** the data from the relevant sheets. The specification is built in a way that makes the Spreadsheet still usable as data presentation for the time in which the application is being developed.

Exported sheets

- General: Not exported (only an overview of the sheets)
- Update Log: Not exported
- Credits: Not exported
- Kanto/Johto/Halloween Spawns and Items: Exported
- Kanto/Johto Rock Smashes: Not exported (Application can not parse images)
- Kanto/Johto Headbutts; Not exported (cf. above)
- Kanto/Johto Quests: Exported
- Kanto/Johto Bounties: Exported
- Pokémon List: Exported, but only Obtainable status
- Bosses: Not exported (Too few informations to justify automated export)
- EV Training: Not exported (cf. above)

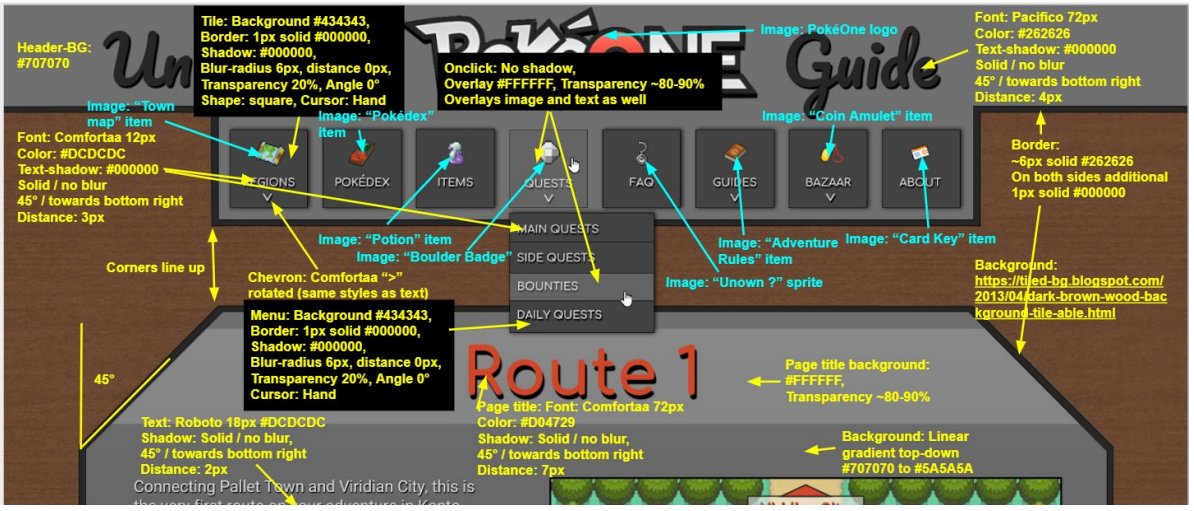
Description of Business Model Entities

Planning Frontend

Design Template

The main challenge when coming up with a matching design was settling for the color palette. An early idea was to generally go with a dark website design rather than one with a white background. As overall palette the colors of the PokéOne logo were chosen: Black, light gray and orange-red as accent color. As additional colors only various shades of gray were added. As background a reddish wooden texture was chosen, to counter the otherwise untextured and clean but bland gray backgrounds. As separator between the gray content backgrounds and the wooden page background a three-part thick border was chosen, consisting of a very dark gray and additional black framing around the border itself. All text has a black cast shadow for better legibility, to stick out from the background, add more depth and to resemble the PokéOne logo.

To make the implementation of the design easier and no longer require to figure out all the technical specifications of the layout, an annotated version of the layout was created:



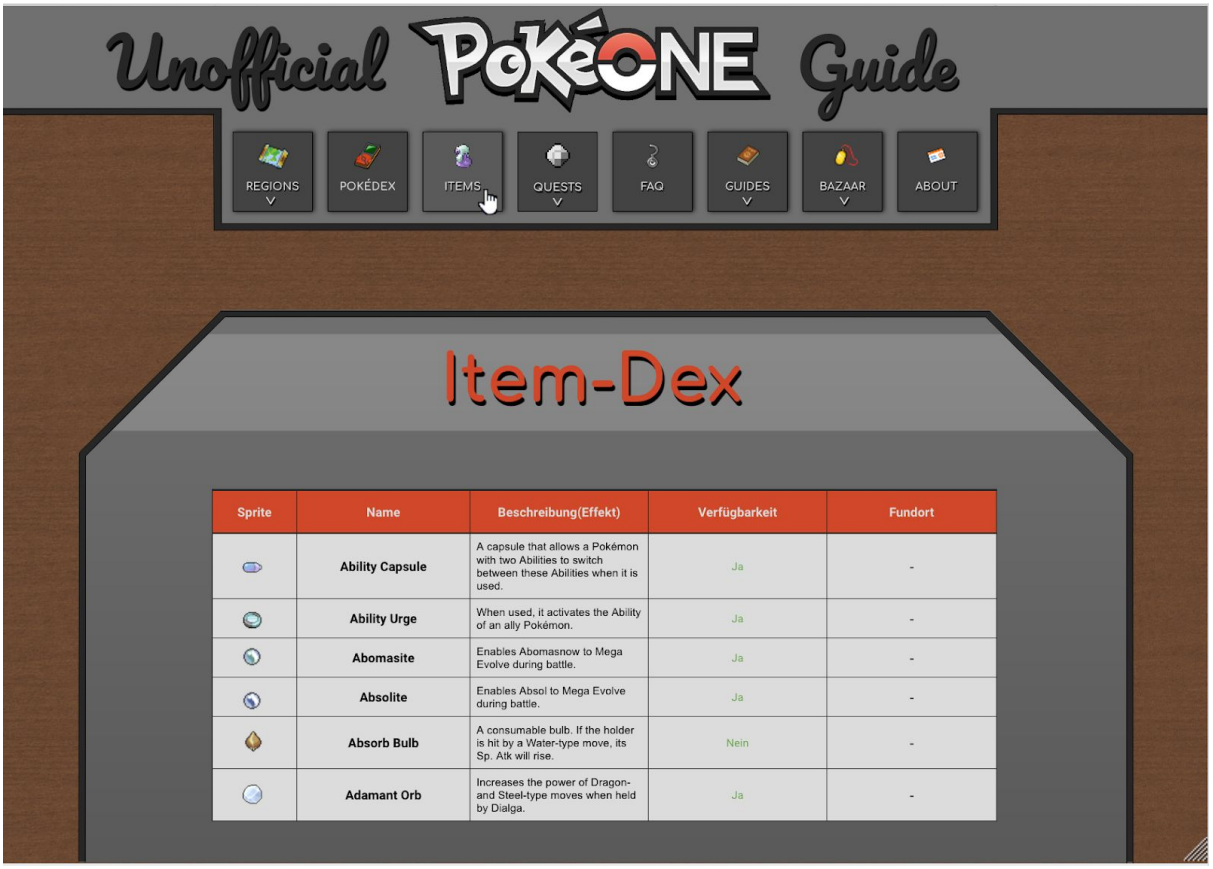
Mockup "Region"

MockUp “Pokédex”

#	Pokémon	Types	Ability I	Ability II	Hidden Ability	Obtainable	Base stats						
							Total	HP	ATK	SPA	DEF	SPD	SPE
001	Bulbasaur	GRASS POISON	Overgrow		Chlorophyll	YES	318	45	49	65	49	65	45
002	Ivysaur	GRASS POISON	Overgrow		Chlorophyll	NO	405	60	62	80	63	80	60
003	Venusaur	GRASS POISON	Overgrow		Chlorophyll	VISIBLE	525	80	82	100	83	100	80
004	Charmander	FIRE	Blaze		Solar Power	TRADE	309	39	52	60	43	50	65
005	Charmeleon	FIRE	Blaze		Solar Power	YES	405	58	64	80	58	65	80
006	Charizard	FIRE FLYING	Blaze		Solar Power	YES	534	78	84	109	78	85	100
007	Squirtle	WATER	Torrent		Rain Dish	YES	314	44	48	50	65	64	43
008	Wartortle	WATER	Torrent		Rain Dish	YES	405	59	63	65	80	80	58
009	Blastoise	WATER	Torrent		Rain Dish	YES	530	79	83	85	100	105	78

Mockup “Pokémon Detail”

Mockup “Items”



Implementation Frontend

xx