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 extensively 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.

Data considerations

Data is the foundation of this application. Its main purpose is the presentation of data which has been carefully gathered and maintained. Thus in this section, the following topics are discussed:

- What data is going to be displayed?
- How is the data stored?
- How is the data obtained?
- How is the data maintained?

Data range

Based on the Master Plan, the following data needs to be able to be displayed by the application. For explanations of the various game-related please refer to the appendix of the Master Plan

- Regions / Event areas
    - Region name
    - Region's locations
- Locations
    - Region / event
    - Sub-Locations (i.e. Mt. Moon → Mt. Moon B1F / B2F / 2F…)
    - Screenshot of entire location
        - Coordinate root of screenshot
    - Location's Pokémon
        - Catchable Pokémon
            - Species
            - Method
                - Grass
                - Surf
                - Fish
                    - Which rod type
                - Floor / Cave
                - Headbutt
                    - Exact placement
                - Rock Smash
                    - Exact placement
            - Time of day
                - Morning
                    - Start time
                    - End time
                - Day
                - Evening
                - Night
            - Rarity
                - Common
                - Uncommon
                - Rare
                - Very Rare
                - Mythical
                - ? / Unknown
            - Notes
        - Gift Pokémon

            - Species
            - NPC
            - Notes
        - Trade Pokémon
            - Species
            - What to trade
                - Pokémon Species
                - Pokédollars
                - Game Corner Coins
            - Notes
        - Static Encounters
            - Species
            - Level
            - Notes
            - Exact placement
        - Bug Contest Pokémon
            - Species
            - Rarity
            - Notes
    - Location's items
        - Placed items
            - Item type
            - Hidden?
            - Exact placement
            - Notes / placement description
        - Gift items
            - Item type
            - NPC
            - Notes
        - Reward items
            - Item type
            - Trainer
            - Notes
        - Quest items
            - Item type
            - Quest
            - Notes
    - Location's Trainers
        - Trainer Type
            - (normal)
            - Gym Trainer
            - Gym Leader
            - Team Rocket
            - Team Plasma
            - Rival
            - Top 4
            - Champ
        - Name (includes trainer class like "Bugcatcher")
        - Pokémon
        - XP reward
        - Money reward

            - item reward
            - Notes
            - Exact placement
        - Location's Tutors
            - Tutor Type
                - Move Tutor
                    - Move
                - Move Reminder
                - Move Deleter
                - Egg move tutor
            - Name
            - Notes
            - Exact placement
        - Location's Other NPC's
            - Name
            - Notes
            - Exact placement
        - Locations's Fruit Trees
            - Berry or Apricorn
            - Berry / Apricorn type
            - Exact placement
        - Portals to other locations
            - Target location
            - Portal label
            - Exact placement
        - Significant Pokémon species
        - Whether the location is "discoverable", i.e. triggers the "Discovered <location name>" message and adds to the correlating achievement.
- Headbutt tree loot
    - Item type
- Rock Smash loot
    - Item type
- Pokémon Mart items
    - Item type
- Quests
    - Title
    - Type
        - Main quest
        - Side quest
        - Daily quest
        - Bounty
    - Quest steps
        - Description
        - From / Trigger
            - NPC
            - Location
            - Quest
        - Location (target)
        - XP reward
        - Money reward
        - Item reward

        - Difficulty
    - Next / previous quest
- Pokémon species
    - National dex number
    - Species name
    - Species names in other languages
    - Availability in PokéOne
        - Obtainable
        - Unavailable
        - Trade only
        - Visible only
    - Obtain methods
    - EV yield
    - Sprite / artwork in various sizes
    - Shiny artwork
    - Elemental type(s)
    - Primary ability
        - Name
        - Description / effect
    - Secondary ability
    - Hidden ability
    - Base stats
        - Attack
        - Special Attack
        - Defense
        - Special Defense
        - Speed
        - HP
    - Evolution chain
        - Evolutions
            - Unevolved species
            - Evolved species
            - Description of evolution trigger
            - Evolution possible in PokéOne?
    - Learnset / available moves
        - Move
        - Obtaining type
            - By Level-Up
            - By TM/HM
            - By Breeding (egg-move tutor)
            - By Move tutor Tutor
        - Available in PokéOne?
    - Popular configurations
        - Ability
        - Nature
    - Builds
        - Moves
        - Item
        - Ability
        - Nature
        - EV's

                - Attack
                - Special Attack
                - Defense
                - Special Defense
                - Speed
                - HP
        - Wild Pokémon item drops
            - Item type
        - Different Forms of a Pokémon
- Elemental Types
    - Name
    - Color
    - Effectiveness against other types
        - defending elemental type
        - damage factor
- Natures
    - boosted stat
    - reduced stat
- Items
    - Name
    - Item category
        - General
        - Pokéball
        - Medicine
        - TM/HM
        - Berries
        - Hold
    - Description / effect
    - Available in PokéOne
    - Obtain methods
        - Placed Items
        - Gifts
        - Rewards (from battle)
        - Trades (Game Corner)
        - Buy
        - Boss loot
        - Wild Pokémon drops
        - Rock Smash
        - Headbutt trees
        - Quest rewards
- Boss Pokémon
    - Interval (daily, every second day, unlimited)
    - Pokémon
        - Species
        - Level (Start level)
        - Moves
            - Type
            - Move type
                - Physical
                - Special
                - Status

                - Power
                - Notes / description / effect
        - Level increase?
        - Max. level
        - Location
        - Exact placement
        - Notes / requirement
        - Item drops
        - Money reward
        - XP reward
- Boss Trainer
        - Interval (daily, every second day, unlimited)
        - Pokémon (multiple)
            - Start Level
            - Max. Level
        - Location
        - exact placement
        - Notes / requirement
        - Item drops
        - Money reward
        - XP reward
- EV Training spots
        - Location
        - Pokémon species
            - EV Yield
            - Method
- User feedback
        - User / user name
        - Feedback text
        - Date
        - Fixed?
- Frequently asked questions (FAQ)
        - Question
        - Answer
- Guides
        - Title
        - Guide text (with markup)
        - Tags
- User
        - user name
        - email
        - password
        - discord username
        - PokéOne accounts
            - Account username
        - Timezone
        - IsOnline
- Basar offers
        - WTS/WTB
        - Pokémon
            - Predicate

                - PVP-ready
                - Shiny
                - HA
                - Any
            - Species
            - Ability
            - Nature
            - Shiny yes/no
            - IV color
            - IV's (ATK/SPA/DEF/SPD/SPE/HP)
            - EV's (ATK/SPA/DEF/SPD/SPE/HP)
            - Moves
            - Gender
            - Price
            - Original Trainer
        - Item
            - Item type
            - Count
            - Price per item
        - Offerer
        - Creation date
        - Last refreshed
        - Active/Inactive
- Contributors
    - Name
    - Tagline

Entity model

The Pokémon universe is very intricate and features a very rich gameplay. Accordingly, the underlying information is anything but simple, and definitely not tailored to be easily stored in a relational database. It is a very difficult task to determine the grade of granularity in which the data should be persisted, and many kinds of information need to be considered in detail. This is done in this section of the documentation. As a result, an entity model is created which is able to store the information in a way which allows the presentation of the data to be as detailed as anticipated.

Locations

The game world PokéOne features is the "container" of pretty much all of the data which is contained within the Unofficial PokéOne Guide. Questions beginning with "where" are by far the most common ones answered by the Guide. It has taken many days of profoundly studying this world and raising, arranging and re-arranging the location's informations in the Guide to develop a data model which will be capable of taking in the data the world consists of.

The easiest part of describing the game world is the sectioning into so-called "regions". At the time the development of this web application is starting, two regions, named "Kanto" and "Johto" are available. A third one, "Unova" is to be released in early 2019.

A region again consists of entities which are commonly referred to as "locations". However, a concise definition of this entity required prolonged considerations and constant refining in order to achieve a granularity which is both granular enough for a precise technical representation but still not too

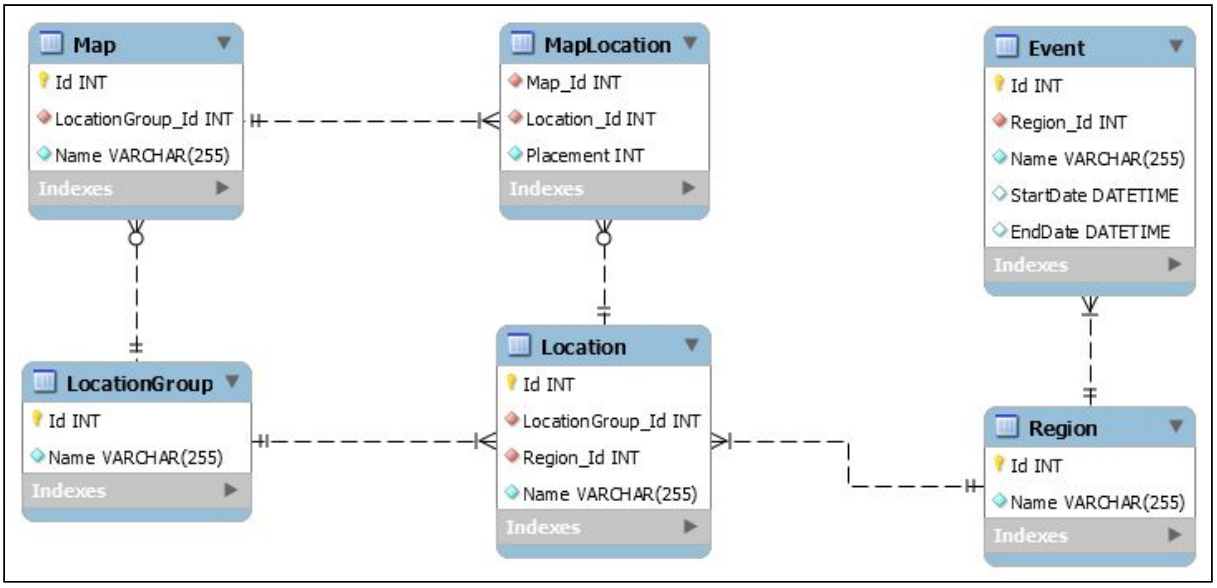
granular for a reasonable presentation to the viewer. Thus the additional terms of “location groups” and “maps” were crafted:

- **Location**: A location is the smallest (virtual) spatial entity which is commonly perceived by the player. This described most easily by looking at which points in time a player changes *location* (as per this definition):
    - Whenever the screen turns black and a new map is loaded. I.e. by using a door, going up a stair or ladder or going through a cave entrance etc., a new location is being entered.
    - In the overworld (outside) whenever the name of the location in the bottom right of the screen changes, a new location is being entered.

    - In the overworld (outside) in some cases when location changes are required to access various parts of an area which are labelled with the same name by the game, each of these area parts may be considered a separate location. The most prominent example of this kind is Route 10 in the Kanto region which is clearly split into a northern and southern part which require walking through a cave to get from one to the other part.
- **Location group**: As described in the concept, there will be pages which each display the details to one specific *location*. However, as many locations in the technical sense, as described above, are very small (i.e. the top floor of a single house is an own location) it would not be user-friendly to have entire pages for such small parts of the game world. Thus, location group entities allow to group multiple locations together and then show one location group per page instead of single locations. Thus i.e. a city with all its houses (each being an own location) is displayed on one page. Every location belongs to exactly one location group, and every location group consists of one or multiple locations.
- **Map**: The location pages may eventually feature images of the ingame map along with the other information about the locations. As one page may contain multiple locations the question arises whether it makes more sense to display a separate map for each location, or one map which contains the images of all locations of a location group combined. As has been found, neither is optimal. Thus yet another entity is introduced to specify which locations are grouped into the same map. This is the map entity. This makes it possible to i.e. group all rooms of a floor of a building into one map, but have separate maps for each floor, while the entire page is about the entire building.
  Every location group can have none or multiple maps. Every map has one or multiple locations, which are however only linked indirectly using “map location” entities.
- **Map location**: A problem that arises when grouping multiple locations on the same map is the requirement that interactive markers can be placed at specific spots on the map. As there is no spatial logic within the placement of the various locations on the map, it is necessary to define where on the map each location’s representation is placed. This information can be held in this entity which stands between the “map” and “location” entity. Each map location has exactly one location and belongs to one map.

In the ERD, this looks as follows:



A couple notes:

- Locations are connected to regions, while LocationGroups are not. This is because a location group does not necessarily represent any logical grouping related to the game world; location groups are solely required for presentation and do not model any aspect of the game world. Theoretically it would be possible to have a location group containing locations from multiple regions, although this is not planned on being used.
- Locations always belong to exactly one region. This may leave a few locations with the tricky question to which region they belong, i.e. the Routes 27 and 26 which connect Kanto and Johto, or the S.S. Aqua ship which travels back and forth between regions. However, for simplicity, this kind of relation is kept, as it already is in the Guide currently. As the location detail pages are not grouped by region this is in all currently known edge cases a negligible drawback.
- As the names of the location groups are used to build page URLs, they need to be unique. Location names need to be unique per region to avoid confusion. Maps have additional names which are used to label the said map.
- The technical representation of the placement of a map location is yet to be determined at this point.
- Occasionally, events are organized. So far events encompassed multiple locations which were only accessible during the event. Events are represented by an entity of the same name and it's locations are grouped under an event-specific region. Event-Locations are also attributed to the the event region even if they were embedded into a regular location, such as the Ecruteak Graveyard during the Halloween Event 2018.
  The Start- and EndDate represent the dates when a particular event started resp. ended. Either of these may be NULL if the date is unknown yet.

Location interactions

While it is crucial to model the locations themselves in a sensible way, the location entities themselves do not contain much information which is of interest to the Guide users. What really can answer all the "Where" questions is information about which "things" are contained within a location. As the listing above shows, the variety of these things is rather large. But for a location feature to be of interest, it needs to fulfill one condition: It needs to be able to be interacted with. Thus the term "location

interaction" is created: Any object within a location which can be interacted with, is a location interaction.

Having a lot of entities which differ in many ways, but all share some aspects and behaviour in a object-oriented environment strongly indicates that these entities could be represented using inheritance. However, the goal is to get a database entity model, which in this case does NOT support inheritance. Still it is possible, as a so-called ORM (Object-Relational mapper) is used which is capable of transforming an inheritance tree into a relational entity structure. Thus at this point all that matters is that it is possible to use inheritance within the entity model. Considerations regarding the transformation are discussed further below.

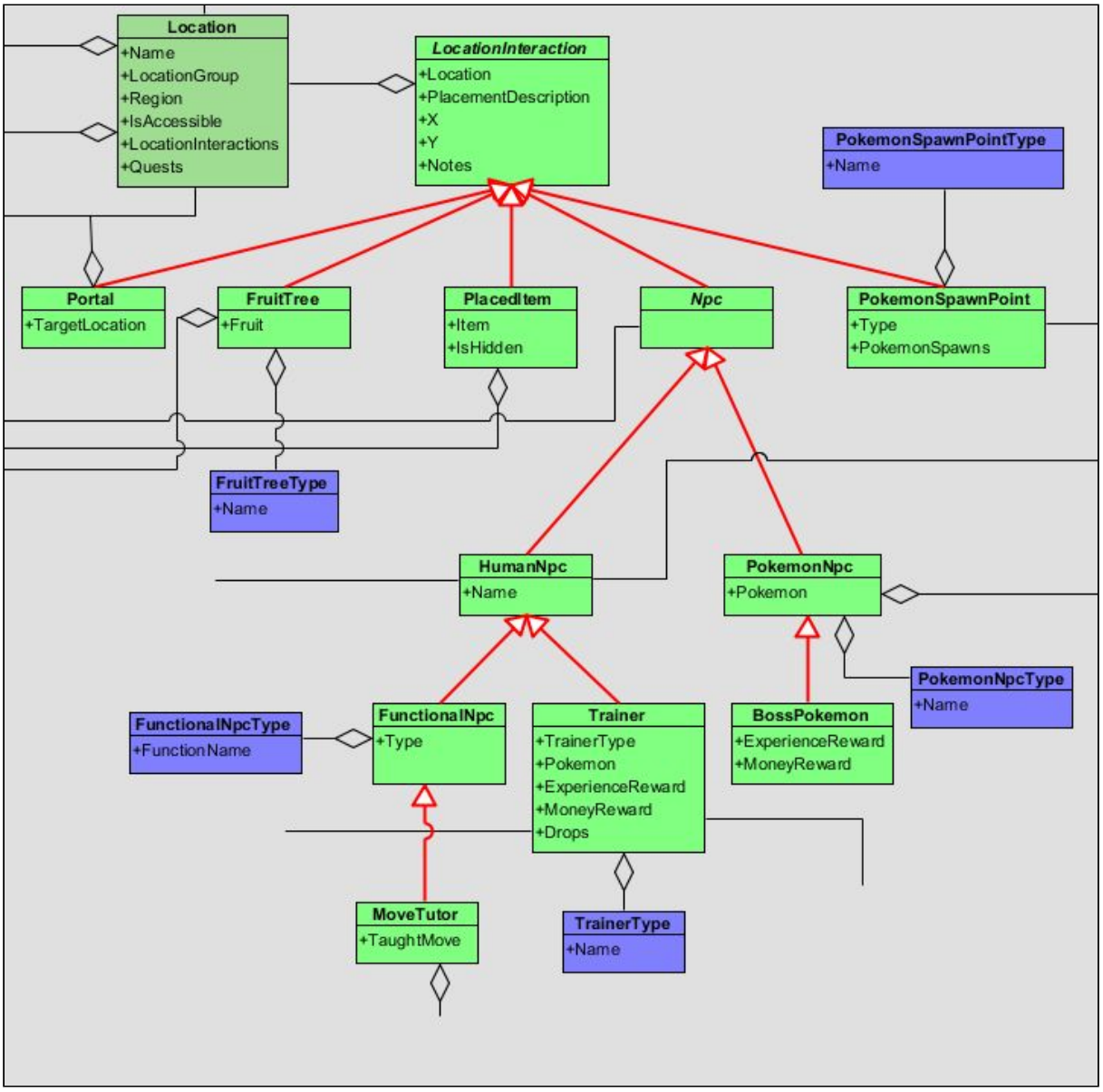
The following location interactions are currently distinguished:

- **NPC**: Non-player-character. This will be an abstract class which is being subclassed to further specify the kind of NPC:
    - **HumanNPCs**: Direct instances are NPCs which can be talked with but don't provide any other functionality. Have a name.
        - **FunctionalNPCs**: Human NPCs with additional functions. These can be Move Reminders, Fossil Resurrection NPCs, Item collectors or similar.
            - **MoveTutor**: Function NPCs which teach one specific move. Have the additional information of which move they teach.
        - Trainer: Human NPCs which can be battled against. Hold information about which Pokémon they use in battle, what type they are (Regular, Boss Trainer, Rival, Team Rocket…) and what rewards (Experience, Money, Items) they yield.
    - **PokemonNPC**: Any Pokémon, which directly appear as interactive NPCs on the map. Can be neutral Pokémon without action, Pokémon which can be fought and caught, or Boss Pokémon. Instead of a name, these entities refer to an entity of type "Pokémon" which holds all information about the Pokémon such as its species. See further below for further elaboration.
        - BossPokemon: Additional information about the rewards when beating a Boss Pokémon (Experience and money). The repetition interval indicates, how often this Boss can be battled. This is usually daily, but other intervals have been observed as well, such as every second day.
- **PokemonSpawnPoint**: Any object which can invoke spawning of a wild Pokémon. This can be grass patches, headbutt trees and similar. Whenever a location has an accessible water area it also has the four spawnpoints relating to Surfing, Fishing with old rod, good rod and super rod. A route usually only has one spawnpoint each with types grass, surf, and fishing rods, but may have multiple headbutt spots and rock smash rocks. This is for two reasons:
    - Although still unconfirmed, it is suspected that different headbutt spots in the same location can have different sets of spawns.
    - Each location interaction has one set of coordinates. However, each Headbutt spot (and Rock Smash rock) should be placeable individually. The other spawnpoint types such as water and grass areas span over a larger area and will most likely not receive exact placement information as this would mostly be useless information, given that these areas are barely ever hard to find.
- **PlacedItem**: An item which is placed on the map. Can be hidden, i.e. it's not displayed with the regular sprite for placed items.
- **FruitTree**: Trees which yield specific items when interacting.
- **Portal**: Points on the map at which the Location is changed. Compare the above section about locations as to when a location is changed. Note that each portal exists twice, once on either of the locations being connected. This is necessary as the placement of the portal is

always respective to one location and are thus, while logically the same “thing”, two separate entities.

The base class “LocationInteraction” allows to specify both a textual description of where in the location the interaction is located (i.e. “north, near NPC Camper Harry”), but also the possibility to specify exact coordinates. The latter however only gets relevant as soon as a screenshot of the location has been created. As that is a lot of work, many locations won’t have a screenshot map at the start, or even never at all.

This hierarchical structure thus looks as following. Note that this is not an ERD but a class diagram. Each table gets an additional Id column which has been omitted for brevity.



The many type classes were initially concepted as static enums. However, as it may very well be that the number of types increases in the future, these were changed to regular classes so that the types can be stored to the database instead of the source code.

The entities related to the LocationInteraction entities are discussed further below.

Pokémon

As one might guess, the central entity of the Pokémon universe are the Pokémon. And it is the Pokémon which most likely cause the most headaches when trying to make them fit for a database.

First of all, the term “Pokémon” needs to be defined explicitly for the context of this entity model, as there are multiple ways to interpret it.

- **Pokémon**: This term is used solely, when one specific Pokémon is meant. This is only the case when talking about Location Interactions, i.e. Boss Pokémon (see above), or the Pokémon a trainer uses. Pokémon entities have properties which can be different between various Pokémon of the same species such as Moves or abilities. One property of a Pokémon is its Pokémon Species.
- **Pokémon species**: Whenever it is not about one specific instance of a Pokémon, but about the species as a whole, the term Pokémon species is used. Pokémon species, respectively it’s varieties (cf. below) are what is listed in the Pokédex and the Spawn lists on the location pages.

Pokémon species have a multitude of properties. For each of them it needs to be determined whether it is relevant for this web application and thus needs to be modelled.

The one property of a Pokémon species which causes the most headaches for this entity model is the concept of forms. Certain species’ Pokémon can appear as one of multiple variants. These variants always differ visually, but may additionally differ in almost any aspect - typing, available abilities, learnset, base stats. As all of these properties are very important for PVP scenarios, and the PokeOneWeb app aims to offer the best information possible for PVP players, this is information that can not be neglected.

As the number of species which have more than one variant is relatively large, such alternative forms should not be considered some kind of exceptional cases, even though the majority of species does only have one variant. An overview of Pokémon species with various forms can be found here: https://bulbapedia.bulbagarden.net/wiki/List_of_Pok%C3%A9mon_with_form_differences.
Additionally, Alola-Forms and Mega-Evolutions exist, which add yet more relevant forms to the game. Most forms have some unique trait to them, as to which properties are unique to this form, or which circumstances influence which form a Pokémon takes.

As way to model these variants to the database, the structure as elaborated by the PokéAPI is adapted. Compare https://pokeapi.co/docs/v2.html#pok%C3%A9mon-species for the documentation and https://github.com/PokeAPI/pokeapi/issues/401 for elaboration on the entities.

Shortly, two different layers of variants are distinguished:

- Varieties: Variants of a Pokémon species which differ in anything else than just visuals, i.e. available abilities, types, base stats or others.
- Forms: Variants of a Variety which solely differ visually.

Thus, every Pokémon Species has one or multiple Varieties, and each Variety has one or multiple Forms. Each Pokémon species has exactly one variety which is considered the default Variety, and each Variety has exactly one form which is considered the default Form. That way it is possible to speak about a Pokémon species as a while and be able to i.e. represent this species with a specific Form.

As said, this entity structure is inspired by the one of PokéAPI, however, different entity names are used to ensure the least confusion about the uses of these entities:

- The Pokémon Species are called “PokemonSpecies” (“pokemon-species” in the PokéAPI)
- The varieties are called “PokemonSpeciesVariety”, signifying that they are varieties of aan entire species, not of a specific Pokémon (compared to “PokémonVariety”). In the PokéAPI this entity is called “pokemon” which is rather vague, and especially confusing given that this data model also has a Pokemon entity which is used for something else,
- The forms are called “PokemonSpeciesVarietyForm”, pretty bulky but as concise as possible, which seems appropriate, given how much time and thought was required to construct the data model in the first place. The PokéAPI has “pokemon-form”. This is also important as often the terms “form”, “variant” and “variety” are used interchangeably.

Regarding what fields and references are contained in which table, it can generally be said that most fields are found on the varieties rather than the species, as the range of properties in which two varieties of the same species may differ is very large.

The properties that remain on the Pokémon Species are:

- The Pokédex number
- A name for the species as a whole, in multiple languages
- The availability in PokéOne of the species as a whole. This will most likely not be persisted but much rather determined on runtime, based on whether any of a species’ varieties is available.
- The varieties of this species (always at least one).
- Which variety is the default one.

The varieties contain the following properties:

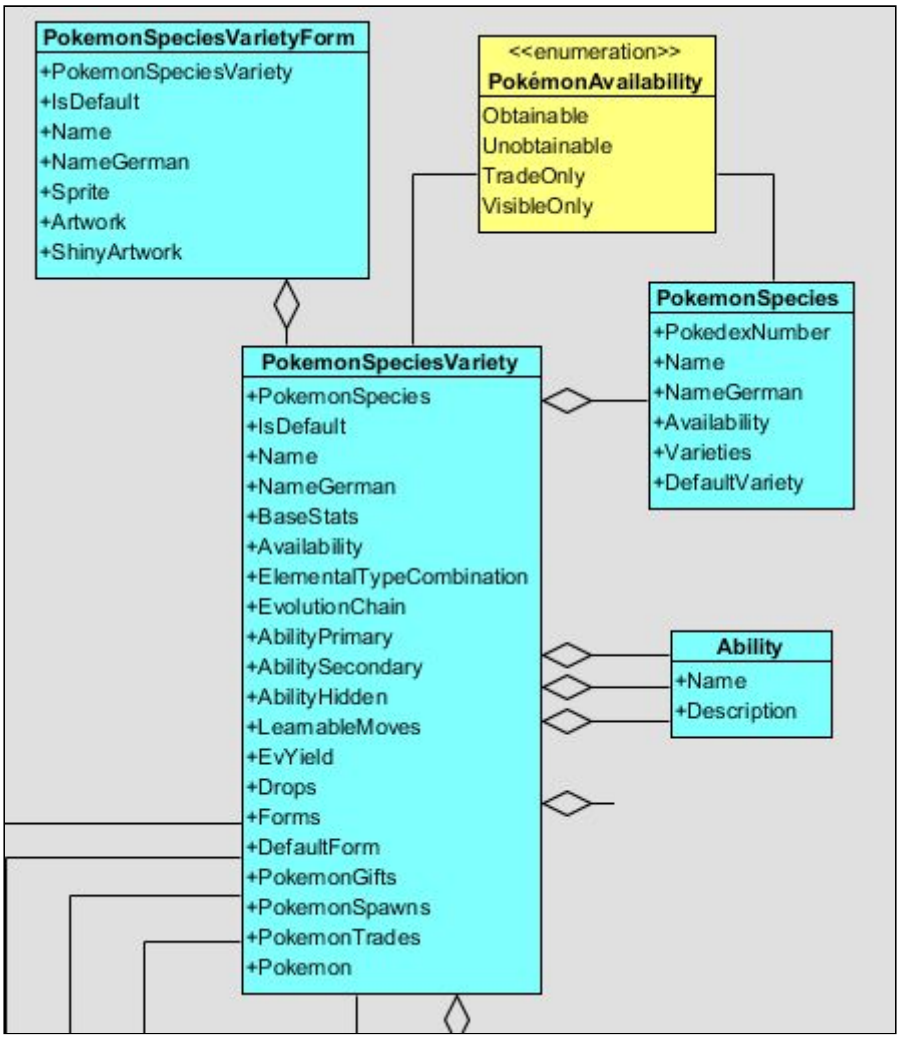
- To which species it belongs (always exactly one)
- Whether it is the default variety
- A name for the variety which usually includes the species name, i.e. “Castform (Rainy form)”
- The base stats (6 values for Attack, Defense, Special Attack, Special Defense, Speed and HP).
- The elemental types (see below)
- The evolution chain the Pokémon appears in. Pokémon without evolutions or pre-evolutions may have this field as NULL.
- Primary, Secondary and Hidden Ability. Only a primary ability is always present on any variety. Each has a reference to an “Ability” entity which gives the ability a name and a description of its effects.
- The learnset / learnable moves (see below)
- The EV yield (how many effort value points defeating this variety yields). 6 values for each stat.
- Which items this variety may drop when being defeated.
- The varieties’ forms. Must be at least one.
- The default form.
- Navigation properties to various obtain methods for this variety (see below)

The forms contain the following properties:

- To which variety they belong.
- Whether it is the default form.
- A name for the form.

- Various sprites and artworks of this form (to be determined how many different ones)

This part of the class diagram looks currently like this:



Evolutions

An Evolution in the Pokémon universe is the process of a Pokémon changing into another species. Most species can evolve into certain other species, or evolve from others. Some species even have either multiple consecutive or branching evolutions.

All evolutions of a species are called an “Evolution Chain”. The great difficulty with modeling these evolutions is the incredibly large amount of possible conditions that can be applied to an evolution. While most species can evolve when reaching a specific level, others have much more specific conditions to fulfill, such as that they need to be traded while holding a specific item, need to have a certain happiness and perform a level-up during a specific time of day and so on.

For now, the decision was made to not fully model this complexity. Instead, the evolution trigger and conditions are purely stored as textual description, potentially with the option to include markup in order to i.e. link to items as has been shown in the Mockup of the Pokémon Detail (i.e. “Trade while holding Prism Scale”).

How multiple evolutions are chained together to form an evolution chain is yet to be determined. The approach of the PokéAPI to nest Evolutions inside each other seems rather counter-intuitive though.

Moves

Moves appear on the following occasions in the data model:

- Moves be learned by certain Pokémon Species Varieties (“Learnset”)
- A move can be part of the set of moves a concrete Pokémon actually knows at a certain point in the game (“Moveset”)
- A move may be taught by a Move Tutor
- A move may need to be known in order to trigger certain evolutions, but as the evolution triggers and conditions are simplified, there is no link to the evolutions in the database.

The properties a move has are discussed in detail in the appendix of the Master plan. Most of them are thus pretty straight:

- Name of the move.
- Damage class. As this is not expected to change, this is solved using an enum.
- Elemental type of the move.
- Attack strength.
- Accuracy (Probability to be successful)
- Power Points (more commonly known as PP, how often a move can be used during battle)

The associations to the Pokemon and Move Tutor entities are solved using intermediary tables, as Pokemon can have more than one move and Move Tutors may be able to teach more than one move.

Similarily the learnset is solved, however in the intermediary table LearnableMove between the entities Move and PokemonSpeciesVariety two additional informations are stored:

- How the move can be learned, which is a fixed set of options, thus solved with an enum.
- Whether the move is available in PokéOne. The move “Stealth Rock” for example can be learned by certain species through a Move Tutor only, and as that specific tutor is not available yet, the move is not actually available in PokéOne. This is an important information for PVP.

Pokémon

The entity named “Pokemon” is used to model concrete Pokémon, i.e. members of a species which have been caught and appear somewhere on the map as NPC or are being owned by a Trainer. Pokémon entities are used to hold the battle-relevant details about an NPC Pokémon (i.e. Bosses and Legendaries) or NPC-owned Pokémon (regular trainers and Boss trainers). They have the following specifications:

- The Pokémon Species Variety they are an instance of.
- The moveset (the moves they can currently use, minimum 1, maximum 4), using a transformation table between the Pokémon and Move entities (a Pokémon may have multiple moves and a move can be had by multiple Pokémon)
- Their ability (as varieties can multiple abilities available)
- Their level
- Some Pokémon, namely Boss Pokémon and the Pokémon of Boss Trainers may have the property to raise in level every time they are defeated. For these a boolean DoesLevelIncrease and MaxLevel fields are provided.

- What items they may drop upon being defeated (the rewards from defeating a trainer are however directly associated with the trainer, not its Pokémon), using a matching transformation table between the Pokémon and Item entities.
- Whether the Pokémon is catchable or not.

Obtaining Pokémon

There are a couple additional entities missing which link the game world in which the player moves around with the location-unbound entities describing the various Pokémon Species, resp. their varieties. Or in other words: Entities describing different ways of obtaining Pokémon.

Generally, it's always the PokemonSpeciesVariety entity which is being linked, not the PokemonSpecies entity. As for all that is known in PokéOne at this point, either variant would work, as there are no situations yet in which a non-default variety of any species can be gotten. However, peeking at how the situation looks in the official games, it is safer to link the varieties instead of the species, given i.e. the following facts:

- In the 5th Generation games "Black" and "White" the two varieties of the Pokémon Basculin can be obtained through different means depending on the version.
- In the 7th Generation games "Sun" and "Moon" various so-called Alola-Forms can be caught which are non-default varieties of certain species.
- Also in the 7th Generation games, the Pokémon Oricorio has four varieties which can be caught in different locations.
- In the newest games, "Pokémon Let's Go Evoli" and "Pokémon Let's Go Pikachu" it is possible to trade Alola Forms against their default variety with NPC's.

The cases in which it is important that it is possible to link varieties to spawns is very small, but it exists. And the chance that it may eventually get relevant for PokéOne is very real.

One way to obtain a Pokémon is catching a PokemonNPC. This is done through the Pokemon entity described above.

The "classic" way of obtaining Pokémon is catching them. This is resolved with the PokemonSpawn entity. A "spawn" in this context describes the event of a wild / unowned Pokémon appearing when interacting with a location feature which can invoke the appearance of Pokémon, i.e. walking through tall grass. The PokemonSpawnPoint has been described further up under Location Interactions. The PokemonSpawn entity links this PokemonSpawnPoint entity (which describes where a Spawn can happen) with the PokemonSpeciesVariety entity. A PokemonSpawn further describes how often and under what circumstances a specific variety of a Pokémon species may spawn. A Spawn point may have multiple Spawns, one for each species variety spawning.

The properties of a Spawn include:

- Which species variety spawns.
- Where the spawn can occur, as a PokemonSpawnPoint, i.e. Fishing with Super Rod on Route 47.
- The rarity, descriptive as no exact numerical values are available, i.e. "Common" or "Rare".
- At which ingame time the spawn can occur. Most are at all times, but some Pokémon may only spawn at night. An ingame day is broken up into four phases, "Morning", "Day", "Evening" and "Night", which each have a specific ingame time at which they start and end. The time periods are linked through a transformation table as a Pokémon may spawn during multiple times of the day.

Another way of obtaining a Pokémon is by getting it from an NPC. There are two occasions this is possible:

- Getting the Pokémon as a Gift
- Getting the Pokémon through Trading with the NPC

The first is very simple to model, a simple entity PokemonGift between the HumanNPC (because Pokémon can not gift other Pokémon or themselves) and the PokemonSpeciesVariety entity is all that is needed. So far, no additional information needs to be known about a Gift.

When trading with an NPC the player has to give something in return for the Pokémon. Usually this is any Pokémon of another specific species. However, two other possibilities exist currently: Trading against ingame currency, and trading against GameCorner coins. The GameCorner is a building where luck-based minigames can be played in order to obtain coins which can be exchanged for Pokémon or Items.The type of Trade is modeled with an Enum, and matching properties for any of the "goods to trade for" exist.

Items

Items are inanimate objects with various effects. There's not a lot of information to be stored on an item itself:

- Name of the item
- Description of its usage and/or effects.
- The bag category, which signifies into which tab in the ingame inventory the item is sorted into.

Beside that, it is all about how to obtain items which is of real interest for the users of the web application:

- Items may be found lying around in the game world → PlacedItem entity.
- Items may be dropped. This means that an item is gained after defeating an enemy. Enemies can in this case be trainers (one may talk of a "reward" instead of a "drop" in this case), specific Pokémon (especially Boss Pokémon), or Pokémon Species Varieties. Regarding the latter: Any Pokémon of certain species (resp. varieties for the most flexibility) may drop certain items after defeating them when the appear as wild Pokémon.
    - → TrainerDropItem entity
    - → PokemonDropItem entity
    - → PokemonSpeciesVarietyDropItem entity
- Items may be given to the player as a gift by an NPC → ItemGift entity.
- Items may be obtained infinitely by buying them from a store. There are basically two kinds of stores in the Pokémon world: Pokémon Marts which are found in almost every town and department stores. All Pokémon Marts have the same items offered, however the offer gets larger as the player progresses through the story. Department stores are large item vendors, usually one per region. As these obtain method should not be duplicated for every store, an entity "GloballyObtainableItems" is created which can be used to model location-unbound ways of obtaining items. The price is not stored seperately, as that would be barely relevant. Instead, a descriptive string like "Buy from any Pokémon Mart for 500 Pokédollars" is stored and displayed.
- Items may be found by using Headbutt (there is a chance that a Pokémon appears, an item is obtained, or nothing happens), or similarily by breaking Rock Smash rocks. Either method may yield one out of a fixed set of items. As both Headbutt spots and Rock Smash rocks

appear in multiple locations, and the items gotten are the same everywhere, the GloballyObtainableItems entity is used for this kind of item acquisition as well.
- Items may be gotten as a Quest reward. As a quest may reward the player with more than one item, an appropriate transformation table is used → QuestItemReward entity.

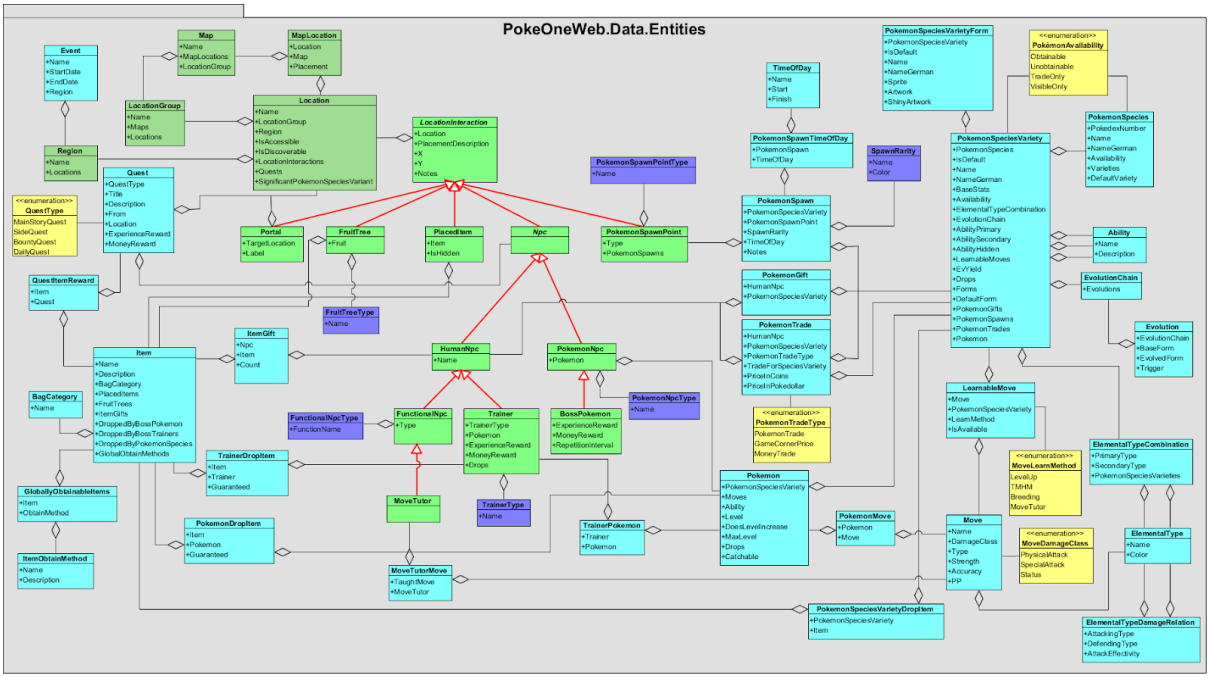
Quests

Quests are small tasks which build up the storyline and have the following properties:

- Which type of quest it is - Main story quest, side quest, daily quest or a bounty (location-bound daily quest), as Enum.
- The name or title of the quest (not unique)
- The description of the quest, what the player has to do to fulfill the quest.
- From who or where the quest is obtained (can be the name of an NPC or a Location). Whether actually making a relation here is of any use is to be determined.
- The location the quest is associated with, usually where it has to be executed.
- How many trainer experience points one gets after completing the quest.
- How much ingame money one gets for completing the quests.
- Item rewards (as intermediary table).

Full business model diagram

At this point the entity model looks like this:



Larger version:
https://drive.google.com/file/d/1A0mdxSkdZS2IM1XkVGxGB6rDHOIHDh4P/view?usp=sharing

Implementation backend

Framework and programming languages

As base for this project, the Framework ASP.NET Core MVC 2.1 was chosen, because 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

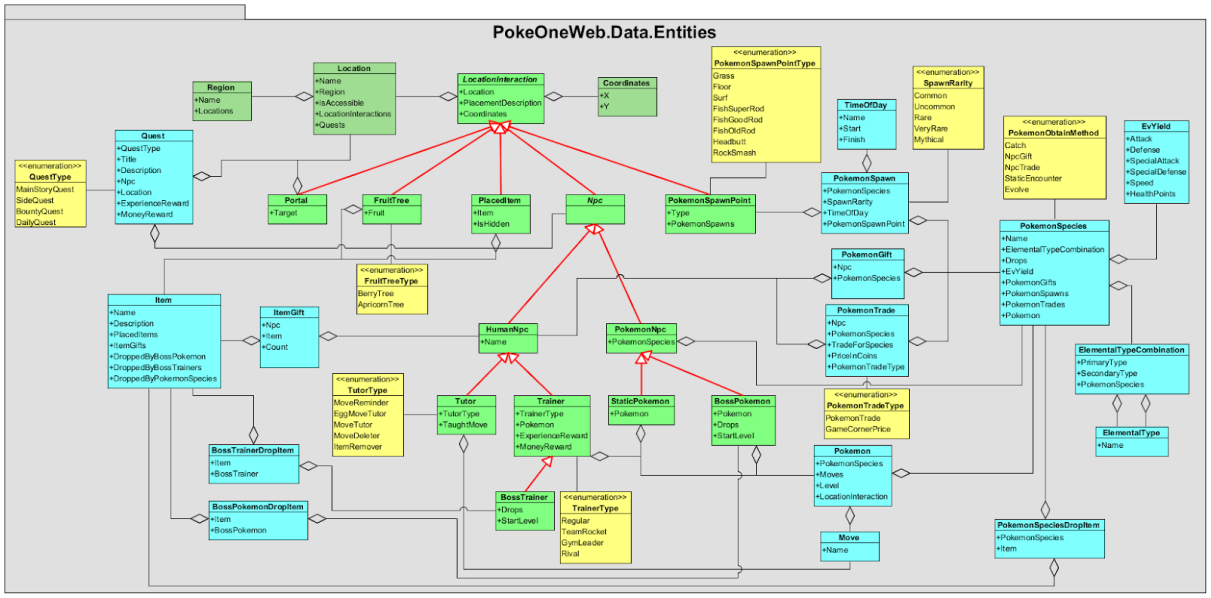
Business model and database.

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.

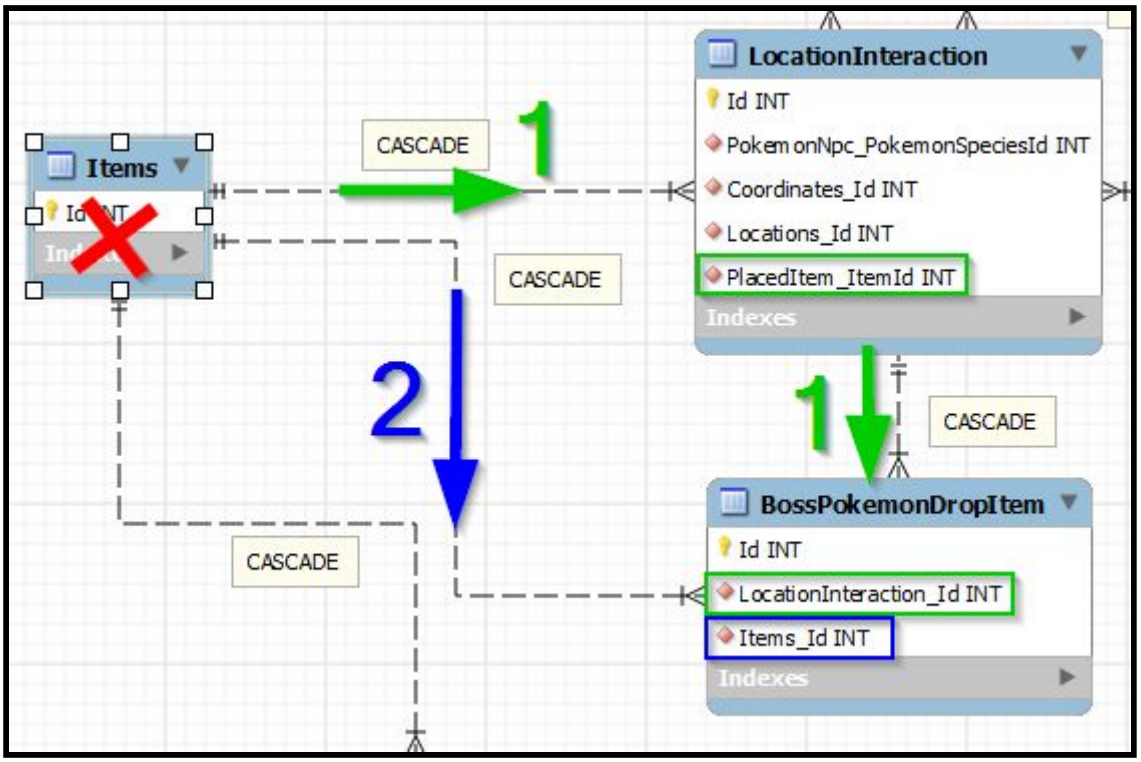
After devising this first big part of the business model by creating a class diagram the respective classes were implemented. The next now is to let EF generate a database out of the class structure. As it was completely new to me to use inheritance with EF Code First I took the time to conduct some 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.

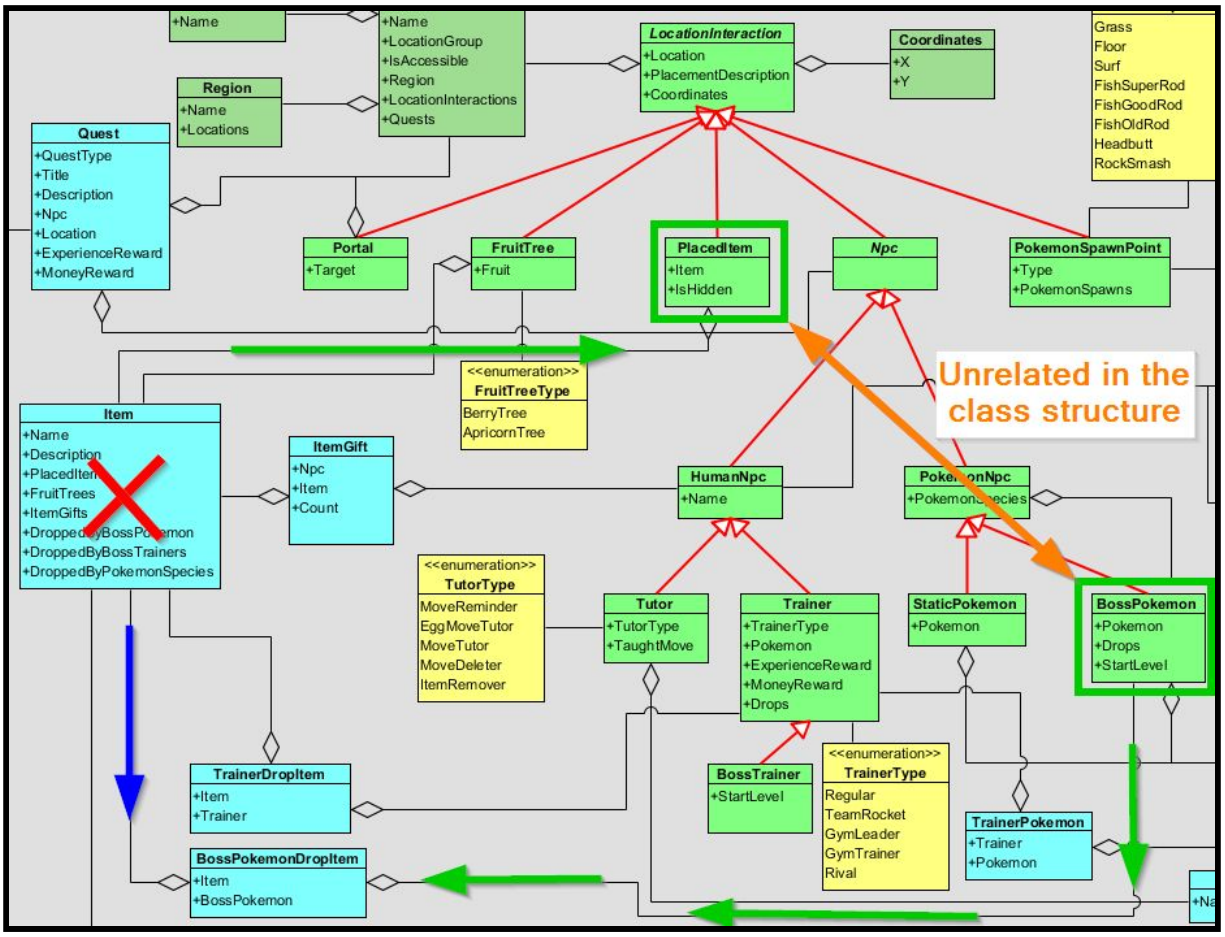
Unfortunately, generating the database kept returning errors:

```
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

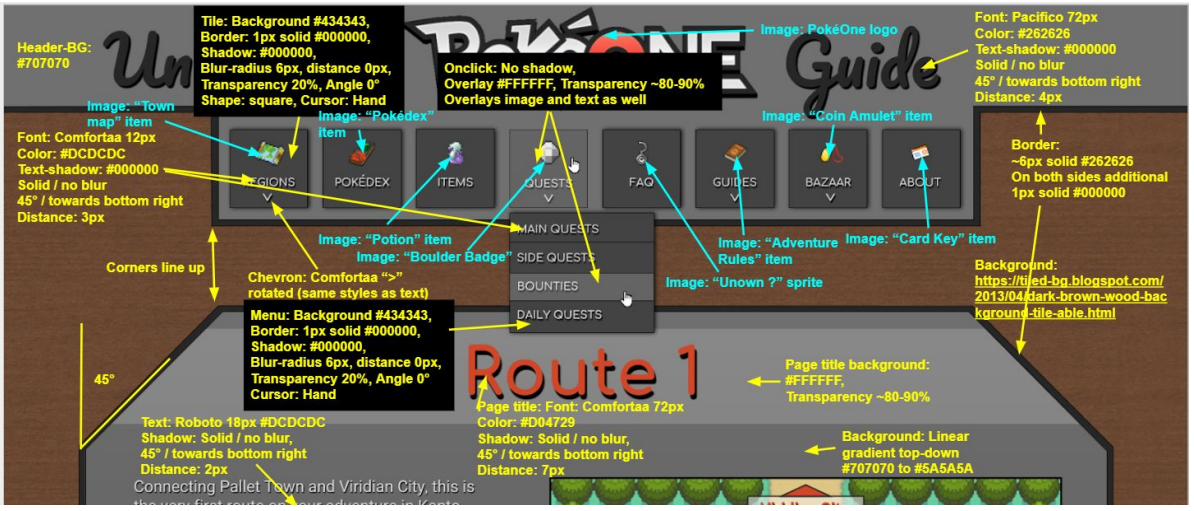
- General: Not exported (only an overview of the sheets)
- Update Log: Not exported
- Credits: Not exported
- Kanto/Johto/Halloween Spawns and Items: Exported
- Kanto/Johto Rock Smashes: Not exported (Application can not parse images)
- Kanto/Johto Headbutts; Not exported (cf. above)
- Kanto/Johto Quests: 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)

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:



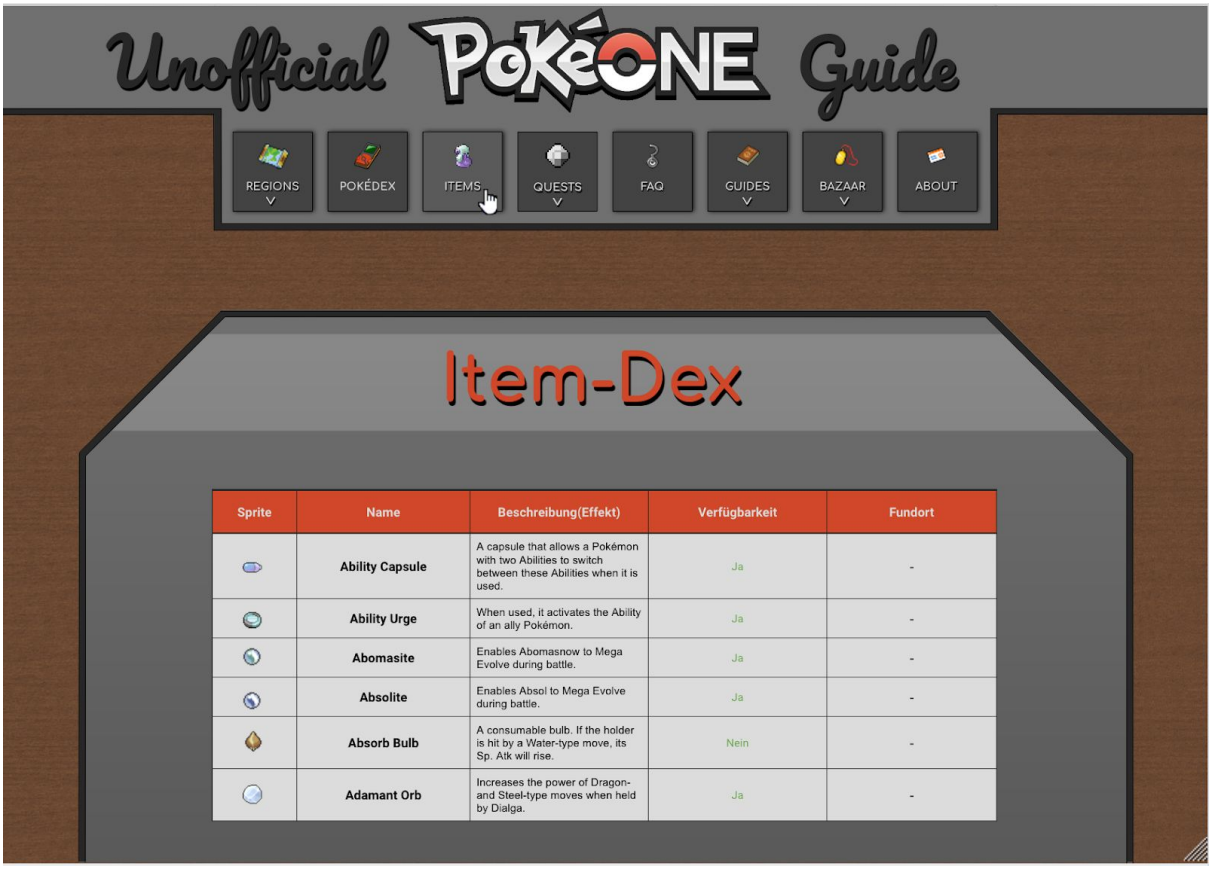
Mockup "Region"

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 Main Layout

First of all, a main layout in HTML/CSS needed to be created. It should not contain any specific content data, just sample texts. The layout was divided into four main sections:

- Header section - wide box: contains the words “Unofficial”, “Guide” and the logo
- Header section - large box: contains all navigation items
- Content section - title: contains site title and the special corners
- Content section - main: contains all site information according to the site

Generally, implementing the layout according to the layout notes (fonts and colors) is not that difficult. However, there were some issues that made the full implementation longer than expected.

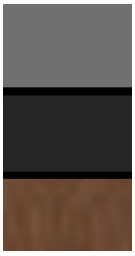
Navigation bar (dropdown)
The mockup required the dropdown menu of a navigation item to have a small gap of 5 pixel between them. The main issue was whenever the mouse left the square of the navigation item, the dropdown menu disappeared before you could over over any of the items within the dropdown. To solve that problem, the usage of a JavaScript that delays the disappearing of the dropdown items was required.

```
<!-- JavaScript for dropdown menu -->
<script type="text/javascript">
    //Dropdown menu opens when hovered over.
    $('.nav-item').hover(
        function () { //when mouse is over element
            $(this).find('.dropdown-content').stop(true, true).delay(200).fadeIn(500);
        },
        function () { //when mouse is no longer over element
            $(this).find('.dropdown-content').stop(true, true).delay(200).fadeOut(500);
        });
</script>
```

Triangle corner of content header

An important element of the layout is the 45° shape of the border around the main content area which leads the view of the user from the narrower navigation bar to the wider content area. Anything non-orthogonal is however hard to realise in HTML which strongly works with orthonormal divisions. The solution was to use CSS3's clip-path feature, which allows flexible masking of div's with polygons or other shapes.

```
-webkit-clip-path: polygon(-1px 150px,1282px 150px,1131px 0px,150px 0px);
```



Three-parted container borders

It might be a bit hard to see, but the container border actually has a small 1px border around itself as well. In order to achieve that on the website, it was required to create three divisions layered behind the original container. The first layer overlaps the original container by exactly one pixel, the second layer will be 11 pixel bigger than the original one (1px of the inner black border and 10px of the gray border), whereas the third layer overlaps the original container by 12 pixel.

```
<!-- content section: borders of main container-->
<div class="border-black border-content-container-layer3">
</div>
<div class="border-gray border-content-container-layer2">
</div>
<div class="border-black border-content-container-layer1">
</div>
```