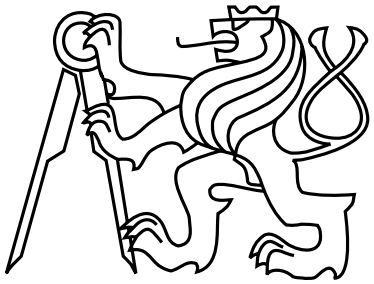
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Rozšíření aplikace pro hodnocení vín

Michal Kašpar

Vedoucí práce: Ing. Ondřej Macek, Ph.D.

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

26. května 2016

Poděkování

V první řadě bych chtěl poděkovat Ing. Ondřeji Mackovi, Ph.D. za vedení mé bakalářské práce, čas obětovaný pravidelným konzultacím a cenné rady po celou dobu realizace a psaní práce. Dále bych chtěl poděkovat mé přítelkyni Zuzaně Chovancové za její trpělivost a korekturu textu v celé práci. V poslední řadě bych chtěl poděkovat mojí rodině za jejich podporu po celou dobu dosavadního studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Ve Zlíně dne 18. 5. 2016 .....

Abstract

The aim of this Bachelor thesis is building on previous works to expand the existing system for wine ratings. The requirement is to create a climate map showing users by selected climatic conditions, wine searching according to selected parameters and also wine detail containing all of the information about selected wine. Another aim is to optimize the database model of wine, which is currently not in the rules for the effective design of the database schema. The motivation for this work is to enable all users of the system to retrieve information on large variety of wines or to compare wines from different parts of the world depending on the climatic conditions of cultivation.

Abstrakt

Cílem práce je v návaznosti na předchozí projekty rozšířit stávající systém pro hodnocení vín. Požadavkem je vytvořit klimatickou mapu zobrazující uživatele podle vybraných podnebných podmínek, vyhledávání vín podle zvolených parametrů a také detail vína obsahující veškeré informace o vybraném víně. Dalším cílem práce je optimalizace databázového modelu vína, který v současnosti neodpovídá pravidlům pro efektivní návrh databázového schématu. Motivací práce je umožnit všem uživatelům systému vyhledávat informace o velké škále vín nebo porovnávat vína z různých částí světa podle klimatických podmínek jejich pěstování.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Tato práce se zabývá rozšířením stávajícího systému pro hodnocení vín a současně navazuje na bakalářské práce [51] a [48], jejichž obsahem jsou v případě [51] realizace jádra systému v podobě hodnocení vín na soutěžích a v případě [48] design uživatelského prostředí. Původní verze systému sloužila pouze pro pořádání vinařských soutěží. Systém tedy umožňoval pouze import vín do soutěže ze souboru, organizaci soutěží organizátory (vytváření soutěží, vytváření komisí, přidělování komisařů do komisí, otevírání komisí apod.) a hodnocení vzorků vín komisaři. Systém byl doposud vyvíjen ve frameworku Django [9], proto i veškerá implementace související s touto prací bude realizována v tomto frameworku.

Předpokladem pro realizaci všech rozšíření obsažených v této práci je optimalizace databázového modelu vína. Tento model není ve stávajícím systému správně navrhnut a nesplňuje databázové normální formy. Problematikou převodu tohoto modelu na model nový, splňující databázové normální formy se zabývá kapitola 2.

Hlavní motivací a účelem této bakalářské práce je rozšíření stávajícího systému o nové funkcionality, které umožní využívat systém také široké veřejnosti. Pro účely vyhledávání vín budou sloužit filtry, jejichž realizace je popsána v kapitole 4. Pro zobrazení informací o vyhledaných vínech, jako jsou například účasti a výsledky na vinařských soutěžích nebo třeba obsah jednotlivých látek ve víně a další, bude k dispozici detail vína, který je obsahem kapitoly 5. Dále bude vytvořena klimatická mapa podle dat z [46], která umožní vyhledávat vinařské organizace, samotné vinaře a ostatní uživatele systému podle zemí nebo klimatických oblastí, ve kterých se nacházejí. To vše v období mezi roky 1900 až 2100. Realizace klimatické mapy je obsahem kapitoly 3.

Kapitola 2

Fáze 1 - Optimalizace databázového modelu vína

Databázový model vína, dále jen DMV, v současné verzi systému slouží pouze pro ukládání základních informací o víně. Hlavní funkcionalitu systému zajišťují modely vzorků, které reprezentují víno na vinařských soutěžích. Z toho důvodu neprobíhá téměř žádná přímá interakce uživatele se systémem, která by měla na DMV vliv. S příchodem požadavků na nové funkcionality od externího zadavatele viz. 1 se však tato skutečnost mění. Na základě těchto požadavků bude DMV tvořit jádro celého systému a většina nových funkcionalit bude na tomto modelu stavět. Dříve než došlo k jejich realizaci jsem tudíž analyzoval původní model vína a na základě této analýzy zjistil, že některé požadavky není možné realizovat efektivně a některé požadavků není možné dokonce realizovat vůbec. Musely tedy následovat potřebné optimalizace a rozšíření původního DMV, které jsou obsahem této kapitoly.

2.1 Požadavky na databázový model vína

Požadavky dále uvedené přicházely od externího zadavatele během vývoje a souvisejí s mým rozšířením DMV. Jejich postupná realizace je zachycena v části 2.4.

2.1.1 Funkční požadavky

Funkční požadavky [57] zde uvedené jsou požadavky na funkcionality spojené s mým rozšířením DMV. Jednotlivé požadavky jsou očíslovány a popisují právě jednu měřitelnou funkci systému. Splnění požadavků je samostatně otestováno v kapitole 6.

- F1: Přidání uživatelovy moštové odrůdy:
  Systém umožní přidat uživatelovu moštovou odrůdu na základě země a vybrané moštové odrůdy.
- F2: Přidání informací o výrobě vína:
  Systém umožní přidat informace o výrobě vína.
- F3: Přidání informací o balení vína:
  Systém umožní přidat informace o balení vína.

- F4: Zadání odrůd, ze kterých je víno složeno:
  Systém umožní vybrat až osm uživatelem dříve přidaných moštových odrůd, jejich sklizní a podíl v procentech, ze kterých je víno složeno.
- F5. Přidání informace o sklizni:
  Systém umožní k uživatelově moštové odrůdě přiřadit informace o jejích jednotlivých sklizních.
- F6: Filtrování vín:
  Systém musí umožnit vyhledávat vína podle zadaných atributů.
- F7: Lokalizace atributů vína:
  Systém umožní překlad všech atributů vína do všech systémem podporovaných světových jazyků již dříve zmíněných.
- F8. Lokalizace názvu moštové odrůdy:
  Systém umožní uživateli zobrazit název moštové odrůdy přeložený do jazyku dle jeho země, pokud se v ní odrůda pěstuje.
- F9: Kopírování moštové odrůdy:
  Systém umožní uživateli kopírovat jeho již existující moštovou odrůdu jako základ pro vytvoření odrůdy nové.

2.1.2 Obecné požadavky

Obecné požadavky [57] zde definují technologické a kvalitativní omezení DMV. Všechny obecné požadavky v této části se týkají normalizace původního DMV. Jedná se o proces změn v databázové struktuře vedoucí k dosažení databázových normálních forem [49]. Pro jednoduchost a lepší srozumitelnost zde uvádím definice z [50].

- O1: DMV musí splňovat první normální formu (1NF):
  První normální forma říká, že každý atribut v relačním schématu musí být jednoduchého datového typu. Jinými slovy, každý atribut ve dvou-rozměrné tabulce musí zaujímat právě jedno místo. Nesmí tedy být typu pole, strom, podtabulka nebo jiná struktura. Tato normální forma obecně definuje strukturu relačních databází.
- O2: DMV musí splňovat druhou normální formu (2NF).
  Pokud je relační schéma v druhé normální formě, pak neexistuje částečná závislost neklíčového atributu na klíčovém.
- O3: DMV musí splňovat třetí normální formu (3NF):
  Podle třetí normální formy neklíčové atributy nejsou tranzitivně závislé na klíči. Jedná se tedy o prohloubení druhé normální formy kde neklíčový atribut nesměl být částečně závislý přímo na klíči. Zde nesmí být neklíčový atribut částečně závislý na klíči ani skrze další neklíčový atribut. Pokud je relační schéma ve třetí normální formě, tak se již dá pokládat za kvalitně navržené. Schémata řady systémů jsou právě v této podobě.
- O4: DMV musí splňovat Boyce Coddovu normální formu (BCNF):
  Třetí normální formu ještě více omezuje Boyce Coddova normální forma kde každý atribut je přímo, ne tranzitivně, závislý na klíči.

2.2 Nedostatky původního DMV

Tato část obsahuje popis nalezených nedostatků původního modelu vína na základě analýzy z úvodu této kapitoly. Původní model vína je na obrázku 2.1

2.2.1 Výkonnostní problémy

Model vína obsahoval velké množství atributů, a tedy i tabulka vín v databázi byla velmi rozsáhlá. Při dotazování se pak pracuje s větším objemem dat, což má negativní dopad na rychlost dotazování. Rozdělení jedné velké tabulky do více malých tabulek propojených cizími klíči jednak zabraňuje vzniku redundancí a taky se pracuje vždy jen s menší částí dat, přičemž zbylá data jsou připojena až v případě potřeby.

2.2.2 Nedostatečná podpora pro požadavky F6 a F7

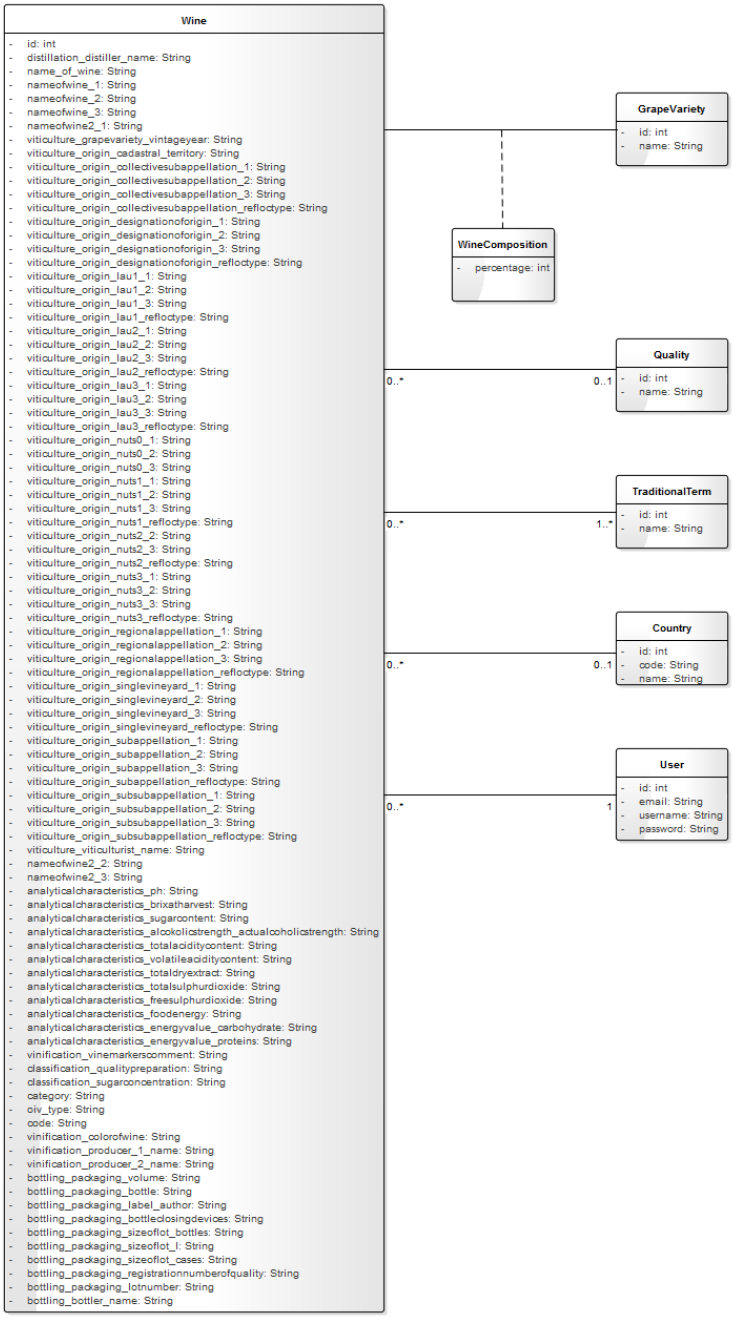
Společné atributy vína jako vinification_colorofwine, quality, classification_sugarconcentration byly přímo obsaženy v modelu vína viz. 2.1 Pro požadavek F6 (Filtrování vín) není taková implementace vhodná z hlediska pozdějšího použití knihovny django-filters 4.2.1.2, stavby indexů a velkého výskytu duplicit.

Kód je také méně kvalitní z hlediska údržby, neboť pro zajištění výskytu dané hodnoty atributu v databázi pouze v jednom formátu je třeba tento formát specifikovat ve zdrojovém kódu viz. 2.2.3. V případě finálního nasazení systému a jeho údržby je mnohem jednodušší změna na úrovni atributu v databázi.

Pro požadavek F7 by díky knihovně django-modeltranslations [32] v tabulce reprezentující víno musel vzniknout pro každý z atributů stejný počet sloupců s překlady jako je počet podporovaných jazyků v systému. Jelikož jednotlivé překlady atributů do různých jazyků jsou vždy totožné, jednalo by se o porušení 3NF.

2.2.3 Výskyt duplicitních hodnot v tabulce vína

Systém umožňuje importovat a následně vytvořit celou soutěž ze souboru ve formátu CSV pomocí skriptu na straně serveru. Tento skript při přidávání nových vín postrádal kontrolu, zdali se již atributy nových vín v databázi nenacházejí. Došlo tedy k tomu, že tabulka vína obsahovala několik mírně rozdílných hodnot vyjadřujících stejnou informaci. Například pro všechna bílá vína je třeba zajistit, aby atribut barvy měl vždy hodnotu “bílé”. V původní verzi systému nebyla žádná kontrola formátu vstupních dat, což vedlo k vytvoření velkého počtu duplicit. Například pro již zmíněný atribut barvy vína se jednalo o duplicity jako “bílé”, “biele”, “Bile”, “Bíle” apod. Tento stav zcela znemožňoval pozdější implementaci požadavku F6, viz. 2.2.2.



Obrázek 2.1: Model tříd původního DMV

2.2.4 Nedostatečná podpora pro původ vína

DMV obsahoval informace ohledně původu vína formou atributů, jejichž jméno začínalo na "viticulture_origin", viz 2.1. Každý z těchto atributů vyjadřoval určitý typ lokality. Jednalo se o atributy typu CharField [8]. Nebylo tedy možné provázat jednotlivé lokality mezi sebou tak, aby omezily uživatelův výběr původu vína při vytváření vína ve formuláři. Řešením bylo převedení atributů původu vína na samostatné modely s cizími klíči na všechny své nadoblasti. Formulář přidávání vína ani převod atributů původu vína na samostatné modely nejsou obsahem této bakalářské práce, nové modely pro původ vína nejsou obsaženy v modelu 2.2 a ani jejich řešení nebude součástí práce.

2.2.5 Špatný návrh způsobu překládání atributů

Ve verzi systému s původním modelem vína byl špatně navržen způsob překládání atributů. Pro překlad atributu byl přímo v DMV vytvořen atribut další, který sloužil pro přeloženou hodnotu. Jedná se o stejnou situaci jako v bodě 2.2.2, tedy porušení 3NF.

2.2.6 Balení vína jako sada atributů modelu vína

V původním DMV sloužila pro informaci o balení vína sada atributů začínajících prefixem "bottling_packaging" obsažená přímo v modelu vína. V tu chvíli nám nebylo přesně od zadavatele zadáno, jak proces balení vína probíhá. Později bylo zjištěno, že některé atributy mohou být společné pro více vín najednou a mohly by proto v budoucnu tvořit v původním modelu vína duplicity.

2.3 Validace původního DMV vůči databázovým normálním formám

- Validace vůči 1NF:
  Všechny atributy v původním DMV jsou deklarovány jako potomek abstraktní třídy Field [19] ve frameworku django, která obecně reprezentuje sloupec tabulky v databázi. Tato knihovna zaručuje, že všechny atributy z ní deklarované splňují 1NF. Proto je i celý původní DMV v 1NF.

- Validace vůči 2NF:
  Porušení 2NF může z její definice nastat pouze u modelů/tabulek, kde je klíč tvořen více atributy. Takový model se nachází v původním DMV pouze jeden a to WineComposition, který slouží jako vazební tabulka mezi modelem vína a modelem moštové odrůdy. Klíč je zde tvořen cizím klíčem vína a cizím klíčem moštové odrůdy. Neklíčovým atributem je zde atribut percentage vyjadřující procentuální podíl dané moštové odrůdy ve víně. Aby byla 2NF porušena, musel by být atribut percentage závislý buď na víně, nebo na moštové odrůdě. Je však zřejmé, že podíl určité odrůdy ve víně nezávisí ani na zvolené moštové odrůdě, ani na vybraném víně. Tento model tedy 2NF neporušuje. Zbylé modely původního DMV obsahují jako klíč pouze atribut id jako jednoznačný identifikátor instance modelu v databázi (řádku v tabulce). 2NF proto neporušují. Původní DMV je i v 2NF

- Validace vůči 3NF:
  Původní DMV obsahoval celou řadu redundantích atributů sloužících pro překlad jejich originálních hodnot (viz. bod 2.2.5). Originální hodnota atributu, například názvu vína, je závislá na primárním klíči vína, tedy id. Hodnota překladů je pro určitý název vína vždy stejná a je proto závislá na jeho originální hodnotě. Hodnota překladu je tedy postupně (tranzitivně) závislá na klíči. Tento stav 3NF zakazuje. Původní DMV nebyl validní vůči 3NF.

- Validace vůči BCNF:
  Vzhledem k předchozímu zjištění, že původní DMV nebyl validní vůči 3NF, nemá v tuto chvíli dále význam zkoumat, zdali byl validní vůči BCNF. Nutnou podmínkou platnosti BCNF je, že model musí být validní vůči 3NF, což v tomto případě neplatí.

2.4 Postupná migrace DMV

2.4.1 Migrace ve frameworku django

Migrace [31] jsou způsob, jakým django propaguje změny v modelech do databáze. Spuštěním příkazu makemigrations z konzole django najde změny v modelech oproti databázi, pokud nějaké existují. Na základě těchto změn vytvoří ve vlastním jazyce migrační skripty pro modely v jejich mudulech. Nakonec jsou příkazem migrate nové změny propagovány do databáze.

2.4.2 Nedostatky migrací pro přechod na nový DMV

Migrace umožňují pouze ekvivalenty k operacím v jazyce SQL pro definici dat (CREATE, ALTER, DROP, ...). Umožňují tedy pouze definovat a měnit strukturu dat v databázi (vytvářet nové tabulky, měnit existující tabulky, mazat tabulky, ...). V mém případě ovšem databáze již obsahovala reálná data a pro operace manipulující s daty (přesun atributu a jeho dat do jiné tabulky, nahrazení určitého modelu za jiný, ...) bylo třeba zajistit zachování všech původních dat. Django migrace nemají přímou podporu ve svém jazyce pro manipulaci s daty (SELECT, INSERT, DELETE, ...). Pro tento problém se nabízela následující řešení:

2.4.2.1 Funkce RunPython přidaná do migračního skriptu

Funkce RunPython [37] umožňuje připojit k existující migraci vlastní kód v jazyce python, který provede potřebnou manipulaci s daty. Po konzultaci s vedoucím práce byla tato možnost řešení zamítnuta z důvodu pozdějších komplikací při vývoji. Důvodem je udržování jednoho konzistentního stavu databáze pro celý tým. Veškeré změny provedené za určité období vývoje jsou poté hromadně migracemi propagovány do databáze a nová verze databáze je sdílena mezi vývojáři. Manipulaci s daty je tedy potřeba provést pouze v tuto chvíli, z toho důvodu není nutné ji distribuovat společně s migracemi.

2.4.2.2 Vlastní skripty v jazyce python

Druhou možností bylo vytvářet vlastní skripty v jazyce python pro manipulace s daty. Výhodou je možnost spuštění skriptu pouze jednou nezávisle na migraci. Nevýhodou je horší údržba, neboť na jednu změnu v modelech připadá migrace a vlastní skript, ale vzhledem k potřebě spustit tento skript pouze jednou je tato nevýhoda přijatelná. Toto řešení jsem použil ve všech případech zmíněných v 2.2.5 v průběhu vytváření nového DMV.

2.4.3 Realizace přechodu na nový DMV

Jak již bylo řečeno v části 2.1, Požadavky na DMV přicházely od externího zadavatele postupně. Přechod byl tedy realizován v několika krocích. Pro názornost jsou k některým krokům přiloženy modely tříd, které obsahují pouze modely (třídy), kterých se změna týká.

- Smazání redundantních atributů názvu a výrobce vína:
  Prvním krokem byla eliminace přebytečných atributů v DMV na základě nové analýzy modelu vína. Pro informaci o výrobci vína byly v původním modelu 2.1 dva atributy. Bylo zjištěno, že druhý z atributů je zcela nepotřebný, proto byl zachován pouze první z nich a druhý byl smazán. Podobná situace nastala u názvu vína, pro který model obsahoval dokonce šest atributů. Tyto redundantní atributy měly v této verzi aplikace sloužit pro překlady prvního z atributů viz. 2.2.5 Všechny redundantní atributy pro název vína byly smazány.

- Smazání zbytečného atributu category:
  Analýzou DMV bylo zjištěno, že atribut category logicky nesouvisí s vínem, ale s jeho vzorky, neboť vyjadřuje kategorii, do které vzorek spadá na soutěži. Tento atribut nabýval hodnoty určené soutěží a nacházel se i v modelu Vzorku (Sample). Nebyl tedy důvod ho dále zachovávat v modelu vína a byl proto smazán.

- Vytvoření možností pro cukernatost a barvu vína:
  Pro problém z duplicitními hodnotami z bodu 2.2.3 byly pro atributy vinification_colorofwine reprezentující barvu vína a classification_sugarconcentration pro obsah cukru přidány možnosti, jakých mohl atribut nabývat viz. B.1. Databáze tímto krokem zamezila do pole v tabulce uložit hodnotu jinou, než některou z nabízených možností.

- Vytvoření modelu pro balení vína:
  Analýzou DMV bylo dále zjištěno, že informace o balení vína, do té doby obsažená v DMV, může být pro některá vína společná. Všechny atributy spojené s balením vína byly přesunuty do samostatného modelu a navázány z vína pomocí ForeignKey [14]. Situaci zachycuje model B.2.

- Vytvoření samostatných modelů pro cukernatost a barvu vína:
  S příchodem požadavku F7 a novým způsobem překladu pomocí knihovny Django-modeltranslation [12] začal být nedostatečný i omezený výběr možností pro atributy z předchozího kroku a to kvůli problémům zmíněným 2.2.2. Pro barvu vína a obsah cukru byly vytvořeny samostatné modely Color a SugarConcentration viz. B.3 a tyto modely byly knihovnou django-modeltranslation registrovány pro překlad. Tímto krokem byl dokončen požadavek F7.

- Vytvoření lokalizace názvu moštové odrůdy pomocí synonym:
  Externí zadavatel dodal první požadavky na model moštové odrůdy. Ten byl v tomto kroku rozšířen o cizí klíč na model Barvy, model Státu a byla přidána nová vazba typu ManyToMany na jinou moštovou odrůdu, která měla vyjadřovat spojení stejných odrůd různě pojmenovaných v jednotlivých zemích. Tento model byl počátečním návrhem implementace požadavků F1 a F8. Model zachycuje B.4.

- Vytvoření samostatného modelu pro oiv_type
  Z důvodů z 2.2.3 a 2.2.2 byl také pro atribut oiv_type v DMV vytvořen vlastní model OivType viz. B.5.

- Přesun vazeb na původ vína do modelu moštové odrůdy:
  Na základě dokumentu poskytnutého zadavatelem, který definoval zobrazení detailu vína pro budoucí uživatele systému, bylo zjištěno, že veškeré dosavadní vazby na modely původu vína již dříve vytvořené viz. 2.2.4 se vážou k moštové odrůdě a nikoli k vínu. Všechny tyto vazby byly přesunuty z DMV do modelu moštové odrůdy. Situace je znázorněna v modelu B.6.

- Vytvoření nových atributů pro energetické hodnoty v modelu vína:
  Dokument obsahoval také informaci ohledně nových atributů pro energetické hodnoty vína. Oproti DMV z předchozího kroku bylo přidáno několik nových atributů do sekce s prefixem "analytical_characteristics" pro obsah sodíku, vitamínů, cholesterolu a dalších viz. B.7.

- Změna modelu barvy vína na model barvy hroznů a vytvoření nového modelu pro barvu vína:
  Ve verzi systému odpovídající modelu B.6 obsahoval DMV a model moštové odrůdy cizí klíč na model barvy. Ukázalo se, že model barvy je modelem barvy hroznů. Musel proto být vytvořen nový model reprezentující barvu vína a cizí klíč z DMV na barvu byl nahrazen cizím klíčem právě na nový model barvy vína. Výsledek je zachycen v B.8.

- Přesun většiny atributů z modelu moštové odrůdy do vazební tabulky mezi vínem a moštovou odrůdou:
  Dosavadní model moštové odrůdy jsem identifikoval jako špatně navržený, neboť kvůli přítomnosti atributů o původu odrůdy přímo v modelu moštové odrůdy by docházelo ke vzniku velkého množství záznamů v tabulce moštové odrůdy, neboť jedna odrůda může být vypěstována na mnoha místech po celém světě. Navíc současný model moštové odrůdy porušuje BCNF. Je to z toho důvodu, že neklíčový atribut názvu odrůdy definuje klíčový atribut synonyma v tabulce synonym. Při daném návrhu by také časem docházelo k velkým výkonnostním problémům, neboť při vzniku nového vína by musely znovu vzniknout všechny moštové odrůdy, ze kterých je víno složeno. Snahou tedy bylo, aby záznam o určité moštové odrůdě existoval v databázi pouze jednou. Veškeré atributy týkající se původu hroznů kromě země byly přesunuty do vazebního modelu mezi DMV a modelem moštové odrůdy obsahujícího informaci o poměru zastoupení dané odrůdy ve víně viz B.9.

- Vytvoření modelu pro lokalizaci názvu moštové odrůdy:
  Na základě analýzy bylo zjištěno, že stejná odrůda se může v různých zemích lišit barvou a informací, zda se jedná o odrůdu aromatickou. Z těchto důvodů přestala být lokalizace pomocí synonym dostačující. Došlo k vytvoření nového modelu přímo pro lokalizaci moštové odrůdy, čímž byl kompletně realizován požadavek F8. Model je zobrazen v modelu B.10.

- Vytvoření samostatného modelu pro uživatelovu odrůdu:
  Podle oficiální dokumentace k vazební tabulce ve frameworku django [9] by tato tabulka měla obsahovat pouze dva cizí klíče na modely, mezi kterými realizuje vazbu. Toto doporučení nesplňuje vazební model mezi DMV a modelem moštové odrůdy kvůli přítomnosti dalších cizích klíčů na označení původu. Byl tedy vytvořen nový model reprezentující uživatelovu odrůdu s cizím klíčem na uživatele. Tímto krokem došlo ke splnění požadavků F1 a F9. Do vazebního modelu mezi DMV a modelem moštové odrůdy přibyl cizí klíč na nový model uživatelovy odrůdy, čímž se splnil požadavek F4. Nový model je součástí modelu B.11.

- Přidání vazby na sklizeň do vazební tabulky mezi vínem a uživatelovou odrůdou:
  Na základě požadavku F5 byla do vazební tabulky mezi vínem a uživatelovou odrůdou přidána nepovinná vazba na model sklizně vytvořený kolegou v týmu viz. B.12.

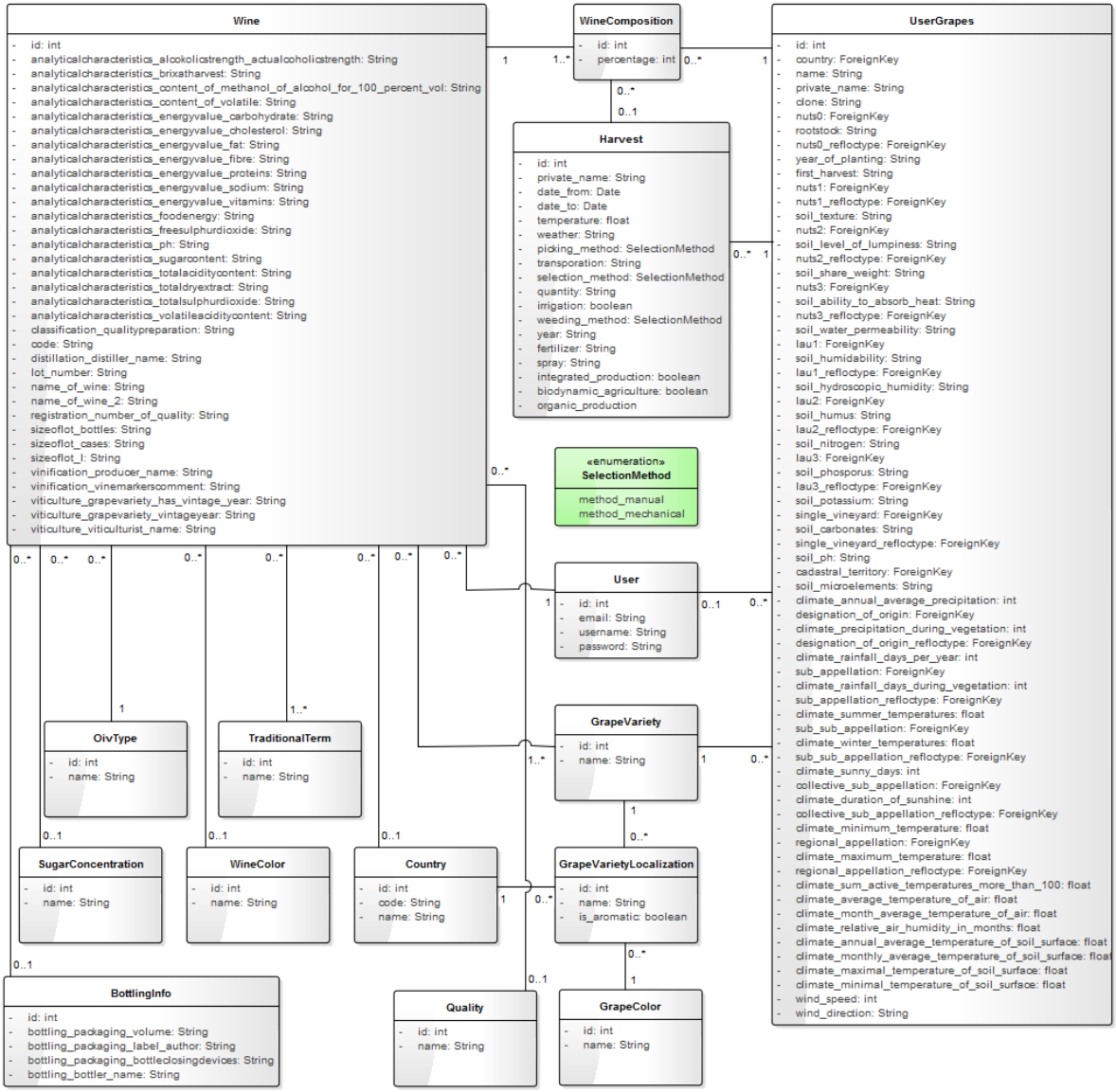
2.4.4 Model tříd nového DMV

V modelu schází část týkající se výroby vína a jeho původu, která byla realizována jinými členy týmu. Původ vína byl navíc přesunut do jiného balíčku. Vazby na modely Původu z DMV jsou proto v modelu zobrazeny jako atributy datového typu ForeignKey. Oproti části 2.4 je zde navíc část atributů v modelu Uživatelovy odrůdy, která byla realizována kolegou z vývojového týmu. Model tříd nového DMV je na obrázku 2.2

2.4.5 Porovnání kvality původního a stávajícího modelu vína z hlediska databázových normálních forem

Původní model vína porušoval ve velké míře 3NF kvůli špatnému systému překladu atributů a přítomnosti překládaných atributů přímo v modelu vína, což mělo velký dopad na výkon, přehlednost a udržovatelnost systému viz. 2.2.5. V průběhu migrací docházelo k porušení 3NF a BCNF, problém byl však vždy vyřešen. Nový model vína je oproti původnímu výrazně rozsáhlejší viz. 2.1, 2.2, výkonnější a je také mnohem lépe a snadněji rozšiřitelný, nežli model původní. Díky spojení atributů spolu souvisejících je model i přehlednější.

DMV je validní vzhledem k databázovým normálním formám až na dvě výjimky. Přímo v modelu vína se nacházejí atributy name_of_wine pro název vína a name_of_wine_2 pro řadu vína. Mimo to se zde nacházejí i atributy jejich překladů vygenerované knihovnou django-modeltranslation [12]. Jednalo by se o stejnou situaci jako v bodě 2.2.5, tedy o porušení 3NF. Avšak na základě konzultace s vedoucím práce a faktu, že překlady těchto atributů se chovají na základě nastaveného prostředí jako atribut samotný a také faktu, že stále není od externího zadavatele znám význam těchto atributů je tato skutečnost přijatelná.



Obrázek 2.2: Model tříd nového DMV

Kapitola 3

Fáze 2 - Klimatická mapa

Obsahem této kapitoly je zařazení klimatické mapy světa, dále jen KM, do kontextu vinařských soutěží. Jejím hlavním účelem má být možnost zobrazování existujících uživatelů v systému a dále možnost vyhledávat podobné uživatele z hlediska klimatické oblasti, ve které se nacházejí. Tyto oblasti se můžou napříč jednotlivými časovými obdobími lišit viz. 3.1 a je tedy do vyhledávání nutné zahrnout i tento fakt. Požadavky na kompletní funkčnost KM jsou v části 3.2.1, případy užití jsou v části 3.2.2. Implementace KM se nachází v části 3.3.

3.1 Klimatické rozdělení světa

Na základě [52] je svět rozdělen do pěti hlavních klimatických oblastí, kterými jsou:

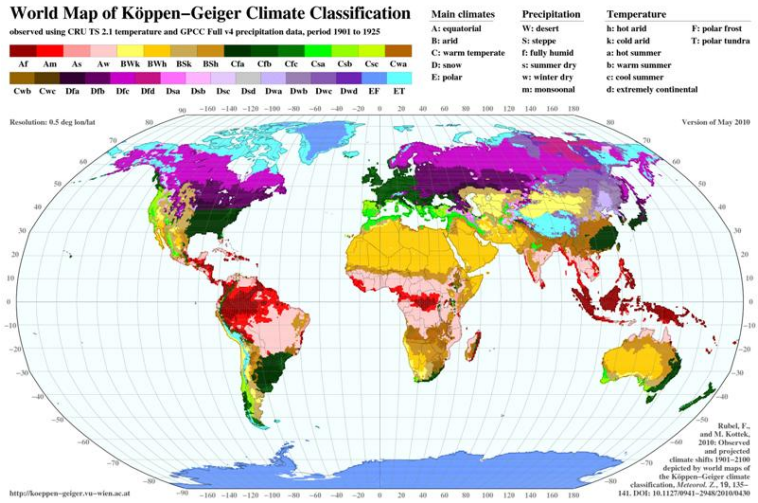
A: Equitorial climates,

C: Warm temperate climates,

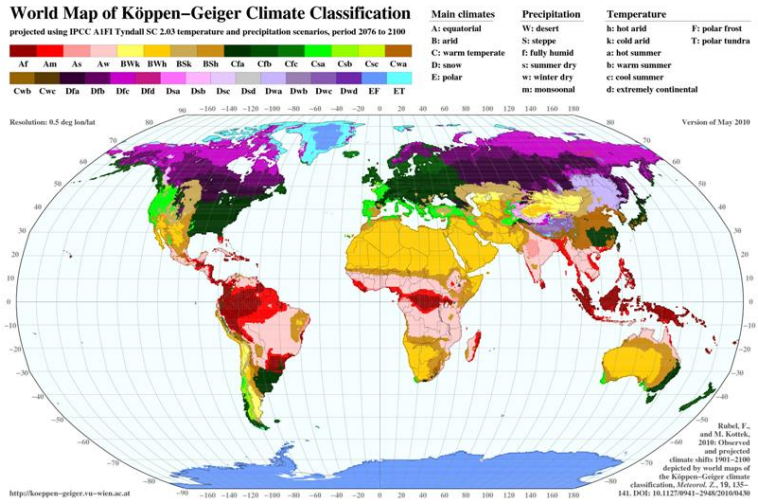
D: Snow climates,

E: Polar climates.

Tyto oblasti se dále dělí dle průměrných teplot ovzduší a průměrného množství srážek viz legenda v obrázku 3.1. Hranice jednotlivých oblastí se liší také podle sledovaného období. Zde se jedná o období 1900-2100. Jak moc se hranice mohou lišit je zachyceno na obrázcích 3.1 a 3.2 kde na prvním z obrázků je svět mezi lety 1900 a 1925 a na obrázku druhém je očekávaný stav mezi lety 2075 a 2100. Aby byl rozdíl mezi obrázky dobře viditelný, zvolil jsem právě obrázky ze začátku sledovaného období, respektive z konce sledovaného období.



Obrázek 3.1: Klimatické rozdělení světa v letech 1900-1925



Obrázek 3.2: Očekávané klimatické rozdělení světa v letech 2075-2100

3.2 Analýza

3.2.1 Požadavky

Požadavky na KM přicházely od externího zadavatele postupně. Z počátku se jednalo především o požadavek na vytvoření dema pro samotné zobrazení jednotlivých uživatelů na mapě s klimatickými oblastmi odpovídajícími dnešní době, tedy období mezi roky 2000 až 2025. Demo mělo také k vybranému uživateli umožnit zobrazit jiné uživatele, jejichž adresa spadala do stejné klimatické oblasti. Později přibyly požadavky na filtrování jednotlivých uživatelů podle zemí, klimatických oblastí a nakonec požadavek na zobrazení a filtrování podle celého sledovaného období tzn. 1900-2100 s výběrem jednotlivých období. Níže je kompletní finální seznam funkčních a obecných požadavků na modul klimatické mapy v systému.

3.2.1.1 Funkční požadavky

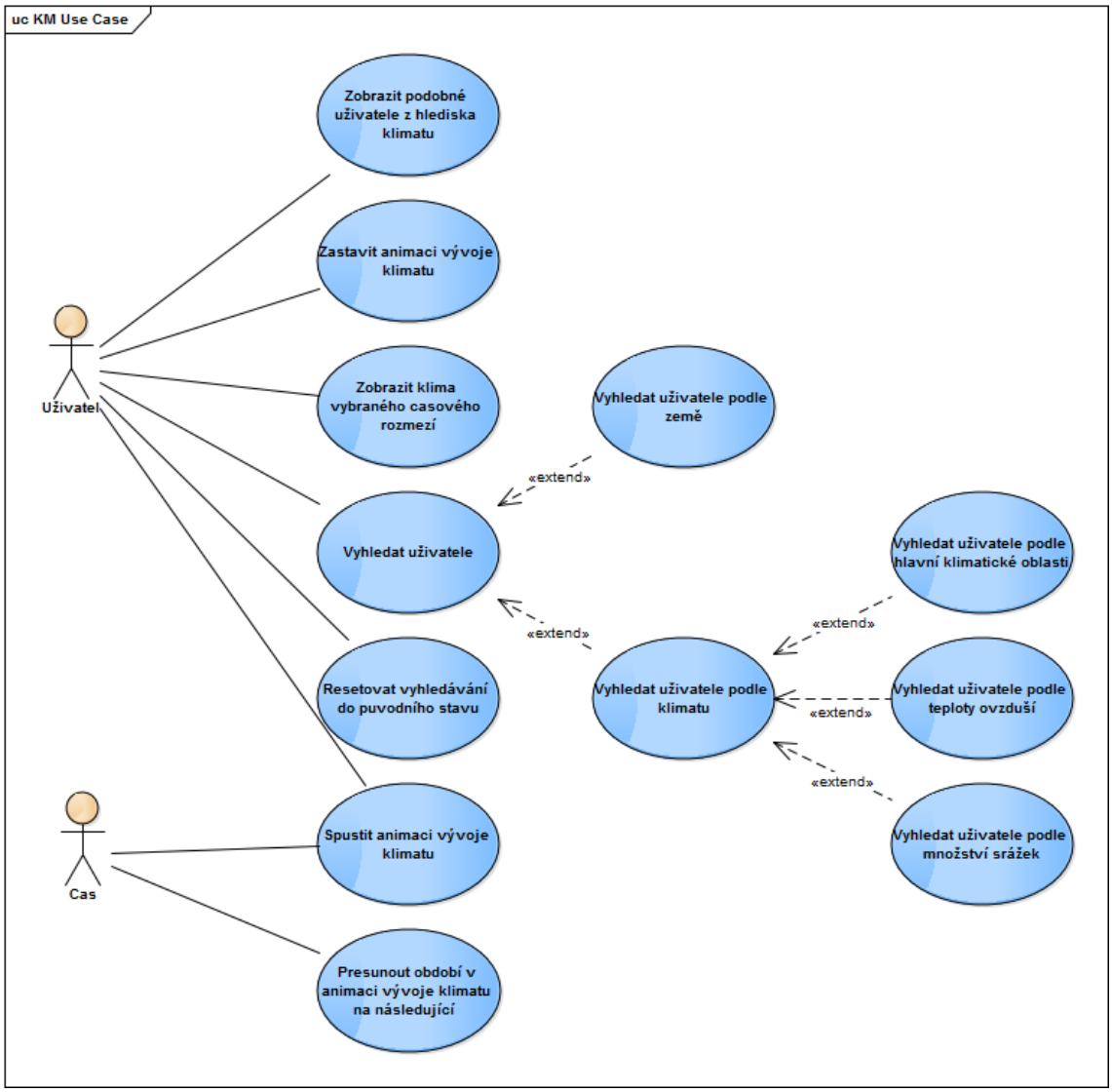
- FK1: Zobrazení všech uživatelů v KM:
  KM umožní zobrazit všechny uživatele v systému
- FK2: Vyhledání podobných uživatelů z hlediska klimatu:
  KM umožní zobrazit uživatele, kteří se nachází ve stejné klimatické oblasti jako zvolený uživatel.
- FK3: Vyhledání všech uživatelů dle vybraných parametrů:
  KM umožní vyhledat všechny uživatele, kteří jsou ve stejné zemi nebo klimatické oblasti (hlavní oblasti, teploty, srážek), nebo jejich kombinaci.
- FK4: Zobrazení stavu klimatu v určitém období:
  KM umožní zobrazit klimatické oblasti odpovídající vybranému časovému rozpětí.
- FK5: Zobrazení animace vývoje klimatu ve sledovaném období:
  KM umožní zobrazit animaci, která bude v pravidelných intervalech měnit klimatické oblasti postupně podle proudu času ve sledovaném období.
- FK6: Vyhledávání pouze podle platných klimatických oblastí:
  Výběr klimatických oblastí pro filtrování bude možný pouze z možností pro hlavní klimatickou oblast, průměrnou teplotu ovzduší a průměrné množství srážek. Vybrat bude navíc možné pouze platnou kombinaci těchto parametrů.
- FK7: Interaktivní vyhledávání uživatelů podle zemí:
  Vyhledávání uživatelů podle zemí bude fungovat jako našeptávač a zároveň bude možné zvolit více zemí najednou.

3.2.1.2 Obecné požadavky

- OK1: Zdroj dat:
  Zdrojem dat pro KM bude World maps of Köppen-Geiger climate classification [46]

3.2.2 Případy užití

Modelování případů užití je způsob, jak nalézt hranice systému, aktéry v systému a případy užití systému [47]. V mém případě jsou hranice systému možnosti funkcionality celého modulu KM. Aktérů v systému může být velké množství. Mohou to být například majitelé vinařství, běžní uživatelé, komisaři na vinařských soutěžích, organizátoři soutěží a další. Pro všechny však platí, že mají přístup ke stejným funkcionalitám modulu KM. V modelu 3.3 je proto pro jednoduchost zobrazen pouze jeden abstraktní aktér Uživatel a aktér Čas. Jednotlivé případy užití, tedy činnosti, které mohou aktéři vykonávat [47] jsou taktéž zachyceny v modelu 3.3. Popis samotných případů užití je dále níže.



Obrázek 3.3: Případy užití pro modul klimatické mapy

- Zobrazit podobné uživatele z hlediska klimatu:
  Akce zobrazí v KM uživatele, kteří se nachází ve stejné hlavní klimatické oblasti jako zvolený uživatel.

- Vyhledat uživatele:
  Součástí akce jsou další dva rozšiřující případy užití, které slouží k vyhledávání uživatelů podle určité země, respektive klimatu. Tyto případy užití jsou navzájem nezávislé, mohou tedy nastat oba najednou, jeden nebo žádný. Mezi těmito případy je konjunkce [27], tudíž jsou akcí vyhledáni uživatelé, kteří splňují oba parametry (pokud jsou zvoleny).

- Vyhledat uživatele podle země:
  V rámci akce může být zvoleno více zemí najednou a mezi jednotlivými zeměmi je disjunkce [1]. V případě konjunkce by totiž akce sloužila v případě zvolení více zemí pro vyhledání uživatelů se sídlem ve více zemích najednou.

- Vyhledat uživatele podle klimatu:
  Součástí akce jsou další tři rozšiřující případy užití podobně jako v případě vyhledávání uživatelů. Jedná se o vyhledávání uživatelů podle hlavní klimatické oblasti, teploty a srážek. Avšak na rozdíl od vyhledávání uživatelů na sobě nejsou jednotlivé případy užití plně nezávislé. Lze zvolit pouze takové kombinace jednotlivých případů, které tvoří existující klimatické oblasti. Lze například hledat všechny uživatele v oblasti, jejíž průměrná teplota ovzduší je typu "hot arid". Není ale možné hledat uživatele, kde hlavní klimatická oblast, ve které se nacházejí je typu "polar" a průměrné množství srážek je typu "monsoonal", neboť žádná taková oblast neexistuje.

- Vyhledat uživatele podle hlavní klimatické oblasti:
  V rámci akce lze vyhledat uživatele spadající do skupin s hlavní klimatickou oblastí z následujícího seznamu:

  - equatorial,
  - arid,
  - warm temperate,
  - snow,
  - polar.

- Vyhledat uživatele podle teploty ovzduší:
  V rámci akce lze vyhledat uživatele spadající do oblasti s průměrnou teplotou ovzduší z následujícího seznamu:

  - hot arid,
  - cold arid,
  - hot summer,
  - warm summer,
  - cool summer,
  - extremely continental,

  - polar frost,
  - polar tundra.

- Vyhledat uživatele podle množství srážek:
  V rámci akce lze vyhledat uživatele spadající do oblasti s průměrným ročním množstvím srážek z následujícího seznamu:
  - desert,
  - steppe,
  - fully humid,
  - summer dry,
  - winter dry,
  - monsoonal.

- Resetovat vyhledávání do původního stavu:
  Součástí akce je vymazání všech atributů vyhledávání obsažených v případu užití Vyhledat uživatele podle klimatu. Akce v KM zobrazí všechny uživatele v systému stejně jako v počátečním stavu.

- Zobrazit klima vybraného časového období:
  V rámci akce je zobrazena KM odpovídající nově zvolenému časovému rozpětí. Všichni uživatelé zobrazení v původním časovém úseku zůstanou zobrazeni i v období novém.

- Spustit animaci vývoje klimatu:
  V rámci akce je opět spuštěna animace vývoje klimatu po předchozím uživatelově zastavení.

- Zastavit animaci vývoje klimatu:
  V rámci akce je zastavena animace vývoje klimatu. Součástí akce je dále nastavení časovače na 600 vteřin, kdy při jeho vypršení nastane případ užití Spustit animaci vývoje klimatu.

- Přesunout období v animaci vývoje klimatu na následující:
  V pravidelných pětisekundových intervalech je v rámci animace vývoje klimatu zobrazena KM odpovídající jednomu z časových období v rozpětí let 1900-2100. Animace začíná hodnotou 1900-1925 a každým dalším intervalem je posunuta o hodnotu 25 dále. Pokud je aktuálním obdobím zobrazeným v KM v rámci animace období 2075-2100, pak je jako další období zobrazeno období první, tedy období 1900-1925.

3.3 Implementace

3.3.1 Implementace podkladové mapy

Jako podkladovou mapu pro KM jsem zvolil Google Maps [18], dále jen GM. Důvodem byla její veliká oblíbenost, rozšířenost na internetu a také prověřená kvalita jejího API [20]. Data pro následující vykreslení klimatických oblastí do GM byla navíc ve formátu KML

[24] viz 3.3.2.1, což je formát pro zobrazování dat v aplikaci Google Earth [17] přímo od společnosti Google. Tudíž je zde garantovaná podpora pro vykreslení tohoto formátu dat i do Google Maps.

3.3.1.1 Ukázka inicializace GM v JS

Ukázka JS kódu níže zachycuje, jak vypadá inicializace Google mapy v systému, takto inicializovaná mapa je vykreslena ve standardním formátu Google mapy, tedy bez vykreslených klimatických oblastí a uživatelů.

```
function initMap() {

    var mapOptions = {
        center: new google.maps.LatLng(44.689763, -12.124659),
        zoom: 2,
        mapTypeId: google.maps.MapTypeId.TERRAIN
    };

    map = new google.maps.Map(document.getElementById('map'), mapOptions);

}
```

Z ukázky inicializace mapy je patrné, že lze nastavit několik parametrů ovlivňujících vzhled mapy. Jejich kompletní výčet lze nalézt v oficiální dokumentaci API [20]. Zde stačily atributy pro typ mapy (v tomto případě terénní), dále střed mapy, zde souřadnice (44.689763, -12.124659) a zoom nastaven na hodnotu dva, což je druhá nejnižší povolená hodnota. Poslední dva parametry byly zvoleny tak, aby na mapě při inicializaci byl viditelný téměř celý svět, tudíž i většina uživatelů (při hodnotě zoom: 1 by už oddálení bylo příliš veliké a svět by byl na mapě vidět zhruba 1,5 krát). Ukázka vzhledu mapy po inicializaci je na obrázku 3.4.

Obrázek 3.4: Ukázka podkladové mapy

3.3.2 Příprava dat pro vykreslování klimatických oblastí

3.3.2.1 Dostupná data

Dle požadavku OK1, musí být data použitá pro vykreslování klimatických oblastí do KM z World maps of Köppen-Geiger climate classification [56]. Zde jsou data rozdělena do tří kategorií podle formátu:

- ASCII text:
  Nejprimitivnější typ dat. Data jsou tvořena dvojicemi souřadnic bodu s informací, ke které klimatické oblasti bod patří. Při spojení všech bodů patřících k dané klimatické oblasti vznikne polygon reprezentující celou oblast. Data však neobsahují žádnou informaci nebo podporu pro vzhled těchto polygonů a také neexistuje parser, který by byl schopen data v tomto formátu do GM vykreslit. Pro můj problém je tento typ dat tudíž nepoužitelný.

- Data pro GIS software:
  GIS software je na počítačích založený informační systém pro získávání, ukládání, analýzu a vizualizaci dat, která mají prostorový vztah k povrchu Země [55]. Tato data obsahují spoustu informací navíc pro geografické účely, například informaci o geometrii, topologii, dynamice a další. Pro mé účely jsou data nevhodná svou přílišnou složitostí.

- KML data:
  Jak již bylo řečeno v sekci 3.3.1, jedná se o formát dat přímo od společnosti Google pro použití v aplikaci Google Earth. Data jsou validní XML soubory obsahující část popisující jednotlivé klimatické oblasti jako Polygony [26] a část obsahující styly (barvu, šířku okrajů) připojené referencemi k těmto polygonům. Tento formát je pro vykreslení klimatických oblastí do KM nejvhodnější, zvolil jsem jej proto jako zdroj dat pro KM.

Ukázka části KML dokumentu pro definici stylů polygonů:

```
<Style id="34">
        <BalloonStyle>
        </BalloonStyle>
        <LineStyle>
                <color>ffc8d0d4</color>
                <width>0.4</width>
        </LineStyle>
        <PolyStyle>
                <color>ff00ff00</color>
                <outline>0</outline>
        </PolyStyle>
</Style>
```

Ukázka části KML dokumentu pro definici samotných polygonů:

```
<Placemark>
        <name>34</name>
        <Snippet maxLines="0"></Snippet>
        <styleUrl>#34</styleUrl>
        <Polygon>
```

```
                <outerBoundaryIs>
                        <LinearRing>
                                <coordinates>
                                        -120.5,36,0
                                        -120.5,36.5,0
                                        -121,36.5,0
                                        -121,36,0
                                        -120.5,36,0
                                </coordinates>
                        </LinearRing>
                </outerBoundaryIs>
        </Polygon>
</Placemark>
```

3.3.2.2 Dostupné scénáře

Dostupná KML data zvolená z předchozí části se dále dělí podle časového období stejně jako u animace vývoje klimatu z části 3.2.2. Tato období jsou dále dostupná v pěti různých scénářích:

- Observed,
- A1F1,
- A2,
- B1,
- B2.

Scénář Observed je jediný dostupný scénář pro data v rozmezí let 1900-2000, byl tedy jedinou možností pro použití. Pro zbylá období bylo možné zvolit jeden ze čtyř zbylých scénářů. Scénář A1F1 zobrazuje největší posuny a scénář B1 nejmenší posuny mezi hlavními klimatickými oblastmi. Hodnoty scénářů A2 a B2 jsou v měřítku mezi nimi [56].

Vzhledem k faktu, že jedním z požadavků je vytvořit animaci pro vývoj klimatu za sledované období, zvolil jsem scénář B1, tedy scénář s nejmenšími posuny mezi hlavními klimatickými oblastmi. Důvodem je zajistit co možná největší plynulost výsledné animace, kde by s velkými rozdíly mezi oblastmi docházelo ke skokům.

3.3.2.3 Úprava dat

Data obsahují v části pro styl polygonu xml element pro reprezentaci barvy polygonu sloužící pro barevné rozlišení jednotlivých klimatických oblastí viz ukázka výše. U všech barev polygonů v původních datech byl alfa kanál [2] nastaven na hodnotu ff, což znamená úplná neprůhlednost. Při zobrazení takových dat do GM by klimatické oblasti zcela zakryly podkladovou mapu, což je nežádoucí. Pomocí skriptu využívajícího knihovnu lxml [28] jsem nastavil alfa kanál u všech barev na hodnotu 64, která odpovídá průhlednosti asi ze dvou třetin. Tuto hodnotu jsem zvolil pomocí metody pokus omyl.

3.3.3 KML parsery

- KML Layer:
  KML Layer [25] je parser přímo obsažený v GM API. Z tohoto hlediska se jedná o standardní nástroj pro parsování KML dokumentů do GM. Dokáže však v GM zobrazit pouze jediný KML dokument z nastaveného URL [43] viz. ukázka použití v dokumentaci. Vzhledem k požadavku F5 pro vytvoření animace je však potřeba parsovat všechny dokumenty s daty najednou a poté pouze volit dokument, který bude zobrazen v GM kvůli dosažení potřebné rychlosti. Tuto možnost KML Layer neumožňuje, nemohl být tedy použit.

- Geoxml3:
  Geoxml3 [16] je předchůdcem KML Layeru a je taktéž vyvíjen společností Google. Podpora KML dat vznikla až v GM Api verzi 3, do té doby byl Geoxml3 jedinou možností jak v GM s KML daty pracovat. Síla tohoto parseru je především v modularitě. Jeho chování se tedy dá nastavit podle potřeb programátora. Navíc tento projekt obsahuje verzi zaměřenou na práci s polygony. Geoxml3 parser dokáže parsovat několik dokumentů naráz a poté jednotlivé dokumenty zobrazovat a skrývat v GM, což byla funkcionalita, jak jsem již uvedl v předchozím bodě, kterou jsem hledal. Tento parser jsem proto zvolil jako nástroj pro zobrazení a práci s KML daty v GM.

3.3.4 Úprava parseru

KML data dále obsahovala v sekci pro styl polygonů, viz. 3.3.2.1 , element pro zobrazení ikony polygonu s číslem reprezentujícím klimatickou oblast. Tato ikona se vždy zobrazila při kliknutí na kterýkoli polygon v KM, což bylo nežádoucí. Geoxml3 parser je naštěstí, jak jsem již uvedl v části 3.3.3, značně modulární. Proto nebyl problém vytváření těchto ikon u polygonů ve zdrojovém kódu parseru potlačit.

3.3.5 Vykreslování KML dat do podkladové mapy

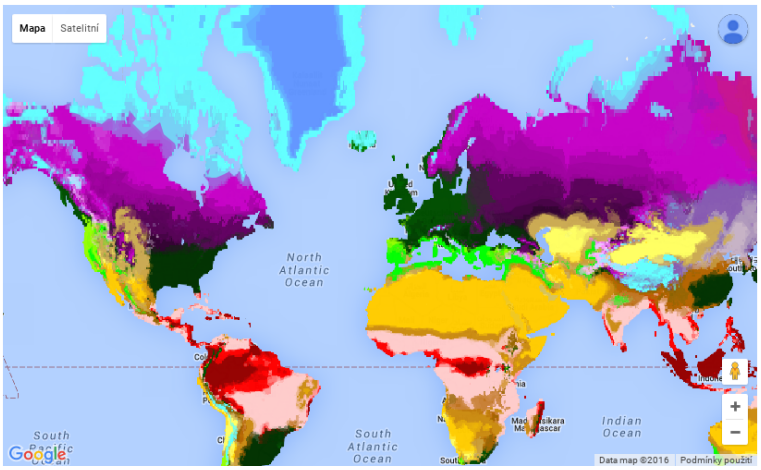
Části 3.3.2.1 a 3.3.3 obsahovaly rozbor výběru a přípravy dat, respektive parseru. Obsahem této části je způsob vykreslení vybraných dat do GM pomocí zvoleného parseru.

Data je třeba do GM vykreslit ihned po načtení stránky, aby uživatel neměl možnost vidět pouze samotnou podkladovou mapu. Inicializace parseru a následné parsování (vykreslení dat) je proto přítomné v inicializační funkci podkladové mapy. Ukázka funkce a parsování:

```
function initMap() {

    var mapOptions = {
        center: new google.maps.LatLng(44.689763, -12.124659),
        zoom: 2,
        mapTypeId: google.maps.MapTypeId.TERRAIN
    };

    map = new google.maps.Map(document.getElementById('map'), mapOptions);

    parser = new geoXML3.parser({
```

```
        map: map,
        zoom: false,
        center: new
                google.maps.LatLng(49.218756, 17.694128)
    });
    parser.parse(['{{ STATIC_URL }}climaticmap/1900.kml',
        '{{ STATIC_URL }}climaticmap/1925.kml',
        '{{ STATIC_URL }}climaticmap/1950.kml',
        '{{ STATIC_URL }}climaticmap/1975.kml',
        '{{ STATIC_URL }}climaticmap/2000.kml',
        '{{ STATIC_URL }}climaticmap/2025.kml',
        '{{ STATIC_URL }}climaticmap/2050.kml',
        '{{ STATIC_URL }}climaticmap/2075.kml'
    ]);
}
```

Nejprve je inicializována mapa, poté je inicializován parser, který přebírá mapu jako parametr konstruktoru. Nakonec je volána funkce parse, která všechny KML dokumenty s daty předané v parametru metody vykreslí do podkladové mapy. Všechna data jsou tímto krokem zobrazena najednou. Důsledkem je vykreslení dat přes sebe, což vede k úplnému zakrytí podkladové mapy viz. 3.5.

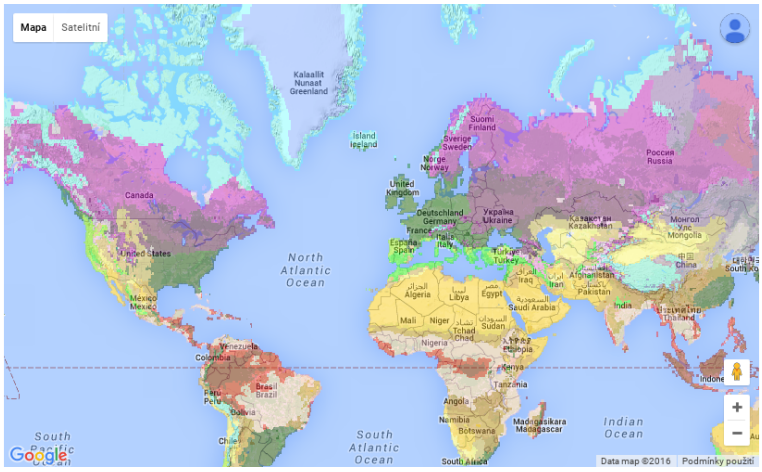


Obrázek 3.5: Vykreslení dat přes sebe

Řešením je skrytí všech dokumentů kromě jediného, který bude zobrazen při načtení stránky pomocí funkce parseru hideDocument. Ukázka popsaného kroku:

```
function hide() {
        for (var i = 0; i < 8; i++) {
        if (i != 0) {
                parser.hideDocument(parser.docs[i])
        }
        }
}
```

Nyní je po načtení stránky v GM vykreslen pouze jeden výchozí dokument, který zobrazuje data pro období 1900-1925, tedy první ze sledovaných období. Zobrazení dalších dokumentů v GM bude řešeno až v části 3.3.7. popisující implementaci animace vývoje klimatu. Finální vzhled KM po její inicializaci je na obrázku 3.6.



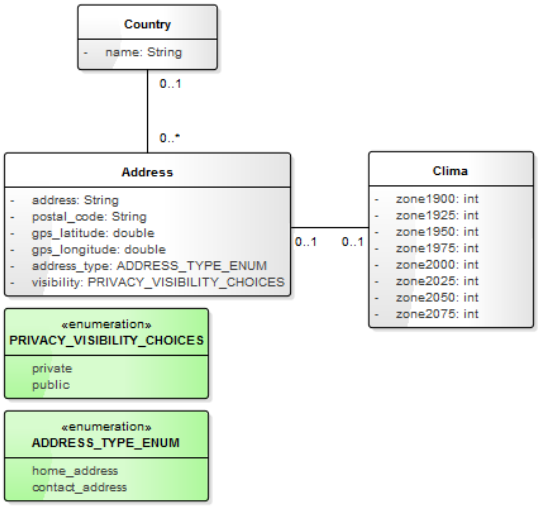
Obrázek 3.6: KM po inicializaci

3.3.6 Vykreslování uživatelů do KM

3.3.6.1 Přiřazení klimatické oblasti k uživatelově adrese

Z důvodu požadavků FK2, FK3 a FK7 pro filtrování uživatelů v KM je potřeba pro všechna období udržovat informace, do které klimatické oblasti každý uživatel v daném období spadá. Jedná se tudíž o výpočetní problém, zdali leží bod uvnitř nebo vně polygonu, kde bod jsou souřadnice uživatele v mapě a polygon je klimatická oblast v mapě. Tento výpočet musí být proveden přes data pro všechna období a dále přes všechny polygony v jednotlivých datech. V každém KML dokumentu pro reprezentaci jednoho období je v průměru 4000 polygonů. Celkově je tedy potřeba provést výpočet asi 32000krát pro každého uživatele. Vzhledem k řečenému není možné tyto výpočty provádět až při inicializaci KM, neboť s přibývajícím počtem uživatelů by se jednalo o velký výkonnostní problém.

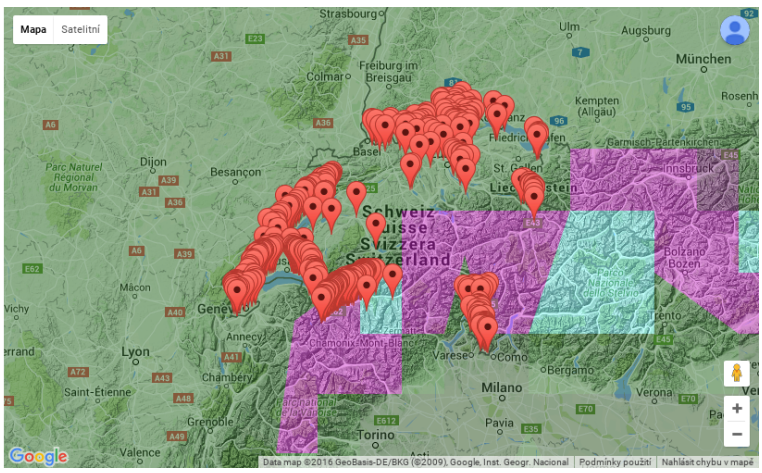
Řešením je provést výpočet pouze jednou a to při uživatelově vytváření adresy. Pro ukládání vypočtených hodnot jsem vytvořil nový model Clima se samostatným atributem pro jednotlivé oblasti a tento model navázal cizím klíčem z adresy uživatele viz. 3.7.



Obrázek 3.7: Klimatický model

Aby k vytvoření klimatického modelu došlo, musí být nejprve získány z Google Maps Geocoding API [19] souřadnice uživatele podle adresy vyplněné ve formuláři. Tato problematika není obsahem této práce a nebude zde proto detailně popsána.

3.3.6.2 Zobrazení uživatelů v mapě pomocí markerů a jejich shlukování



Obrázek 3.8: KM bez použití shlukování markerů

Pro označení, kde se uživatel v mapě nachází, jsem použil Marker z GM API [30]. Jedná

se o běžný způsob zobrazení bodů v GM. Při inicializaci KM jsou prostřednictvím JSON [23] získány z databáze informace o uživatelech včetně jejich adres a klimatických modelů. Pro všechny uživatele je na základě těchto dat vytvořen nový marker, který je ihned zobrazen v mapě. Na obrázku 3.8 je ukázka takto vykreslených uživatelů v KM. Jedná se o 604 švýcarských vinařství, jejichž adresy byly dodány od zadavatele jako počáteční data pro testování. Následující kód demonstruje zobrazení jednoho uživatele na mapě.

```
var marker = new google.maps.Marker({
    position: new google.maps.LatLng(dataMap['lat'], dataMap['lng']),
    title: dataMap['nick']
    map: map
});
```

Kde dataMap je slovník obsahující informace o uživateli získané pomocí JSON z databáze.

Z obrázku 3.8 je patrné, že s velkým počtem uživatelů se mapa stává velmi nepřehlednou. Na základě požadavku od zadavatele jsem měl nalézt řešení, které tento problém optimalizuje. Zvolil jsem MarkerClusterer [29], opět přímo z GM API. Tento nástroj vytvoří nad mapou mřížku podle nastaveného parametru gridSize a všechny markery obsažené v jednom okně této mřížky nahradí symbolem s číslem, kolik markerů je uvnitř obsaženo. Další velkou výhodou je optimalizace výkonu, neboť markery nahrazené zmíněným symbolem jsou do mapy vykresleny až ve chvíli, kdy uživatel přiblíží mapu podle nastaveného parametru maxZoom. Na obrázku 3.9 je ukázka stejného počtu uživatelů jako na obrázku předchozím s použitím MarkerClustereru. Ukázka inicializace uživatelů s použitím MarkerClustereru:

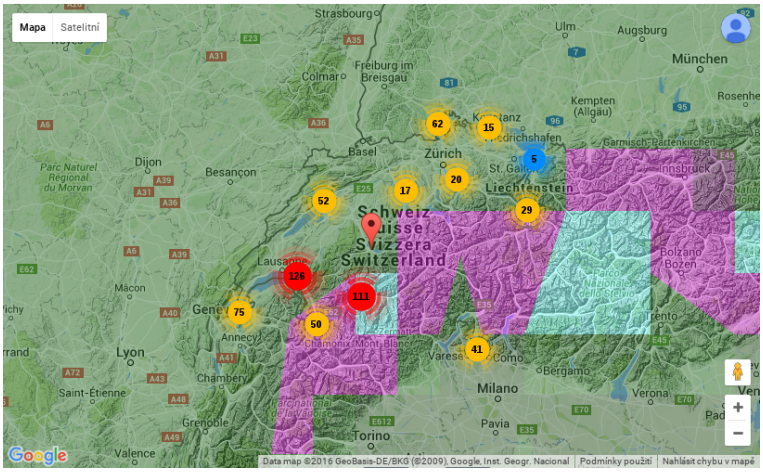
```
var mcOptions = {gridSize: 50, maxZoom: 15};

 function init_users() {
    $.each(users, function (index, dataMap) {
        markers.push(createNewMarker(index, dataMap));
    });
    markersCluster = new MarkerClusterer(map, markers, mcOptions);
}
```

Kde funkce createNewMarker je funkce pro vytvoření markeru uživatele z minulé ukázky.

3.3.7 Animace vývoje klimatu

Jedná se o implementaci požadavku FK5. Příprava dat pro tento požadavek byla popsána v části 3.3.2. V tuto chvíli bylo nutné zajistit automatické přechody mezi daty pro jednotlivá období. Současně bylo potřeba nalézt nástroj pro zajištění požadavku F4 pro zobrazení určitého období v KM. Pro tyto účely jsem do KM zakomponoval JQRangeSlider [22]. Jeho konfiguraci jsem provedl tak, aby každých pět vteřin došlo k přechodu mezi jednotlivými obdobími, přičemž původní období je v KM skryto a nové období zobrazeno. Zároveň je uživatel schopen animaci zastavit a ručně posunout slider na ose do příslušného období, čímž opět dojde k jeho zobrazení. Při přerušení animace uživatelem je spuštěn timer časovač na deset minut. Po uplynutí této doby je opět spuštěna animace. Ukázka funkce volané každých pět vteřin timerem pro posun slideru o jedno období:

```
function moveSlider() {
    var values = slider.rangeSlider("values");
```



Obrázek 3.9: KM po použití shlukování markerů

```
    if (values.max == 2100) {
        values.min = 1875;
        values.max = 1900;
    }
    slider.rangeSlider("values", values.min + 25, values.max + 25);
}
```

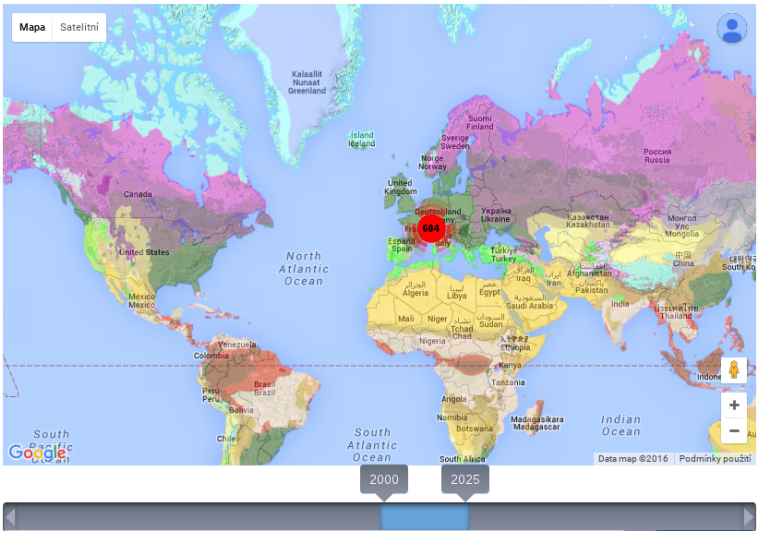
Funkce volaná při změně hodnot slideru pro zobrazení nového období a skrytí období původního:

```
slider.bind("valuesChanged", function (e, data) {
    var values = slider.rangeSlider("values");
    if (values.max - values.min != 25) {
        slider.rangeSlider("values", 1900 + valueToHide * 25,
                1900 + valueToHide * 25 + 25);
        return;
    }
    var valueToShow = (values.min - 1900) / 25;
    parser.hideDocument(parser.docs[valueToHide]);
    parser.showDocument(parser.docs[valueToShow]);
    valueToHide = valueToShow;
});
```

Ukázka KM společně se sliderem je na obrázku 3.10.

3.3.8 Vyhledávání uživatelů v KM

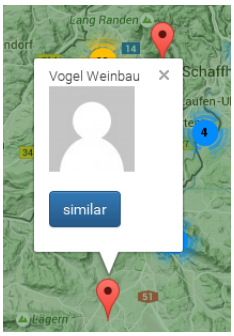
Tato část obsahuje popis implementace všech požadavků týkajících se vyhledávání uživatelů, tedy požadavků FK2, FK3, FK4, FK6. Všechna tato vyhledávání je možné řetězit. Znamená to, že uživatel může zároveň vyhledat uživatele podle klimatu, podle zemí a podle podobnosti s jiným uživatelem. Výsledkem je zobrazení těch uživatelů, kteří splňují všechny



Obrázek 3.10: KM společně se sliderem pro animaci vývoje klimatu

parametry vyhledávání najednou. Jedná se tedy o konjunkci. Výjimkou je vyhledávání uživatelů podle zemí, kde jsou vyhledáni uživatelé s adresou alespoň v jedné z vybraných zemí. Zde se tedy jedná o disjunkci. Následuje popis implementace samotných atributů pro vyhledávání.

3.3.8.1 Vyhledávání podobných uživatelů



Obrázek 3.11: Ukázka InfoWindow jednoho uživatele

Pro požadavek FK2 (vyhledání podobných uživatelů z hlediska klimatu jako vybraný uživatel) jsem u každého markeru implementoval InfoWindow [21]. InfoWindow se objeví vždy při kliknutí na marker uživatele a v mém případě obsahuje název uživatele, profilový obrázek a tlačítko pro zobrazení podobných uživatelů, které slouží právě pro požadavek FK2. Ukázka uživatelova InfoWindow je na obrázku 3.11. Ukázka inicializace:

```
marker.addListener('click', function () {
        var infoWindow = new google.maps.InfoWindow({
```

```
        content: contentString,
        maxWidth: 200
    });
});
```

Kde content string je řetězec obsahující příslušné HTML pro vzhled InfoWindow po jeho zobrazení.

3.3.8.2 Vyhledávání uživatelů podle zemí

Součástí požadavku FK3 je vyhledávání uživatelů podle zemí, přičemž musí být splněn požadavek FK7. Bylo proto potřeba nalézt prostředek umožňující více vstupních hodnot najednou, kde výběr každé z hodnot funguje jako našeptávač. Pro tento účel jsem použil knihovnu Bootstrap Tags Input [7]. Pro funkci našeptávače bylo potřeba načíst z databáze podporované země pomocí JSON a předat je jako parametr této knihovně.

3.3.8.3 Vyhledávání uživatelů podle klimatu

Zbývá popsat implementaci druhé části požadavku FK3 a to vyhledávání uživatelů podle klimatu. Na základě potřeby splnění požadavek FK6 jsem za nejvhodnější způsob pro výběr klimatu vybral vytvoření tří select boxů z knihovny Bootstrap [3] pro výběr hlavní klimatické oblasti, průměrných srážek a průměrných teplot. Nutné však bylo implementovat jejich funkcionalitu tak, aby bylo možné vybrat pouze platnou klimatickou oblast. Vytvořil jsem proto JSON objekt obsahující výčet všech povolených klimatických oblastí. U každého selektu jsem dále implementoval funkci volanou při změně hodnoty selektu, která podle zvolené hodnoty upraví možnosti pro výběr u zbylých dvou selektů. Ukázka vytvořeného JSON objektu obsahujícího povolené klimatické oblasti je v příloze C, ukázka funkce pro dynamické překreslování selektů podle vybraných hodnot je v příloze D.

Kapitola 4

Fáze 3 - Filtrování vín

Jak jsem uvedl v části 1, před zadáním mé bakalářské práce sloužil systém pouze pro hodnocení vín na soutěžích. Kapitola 2 popisuje úpravu Databázového modelu vína pro účely nových plánovaných funkcionalit systému. Jednou z nich je právě filtrování vín.

4.1 Analýza

Na rozdíl od předešlých dvou kapitol zde nebudu uvádět výčet požadavků, neboť jednotlivé požadavky se navzájem liší pouze parametrem, podle kterého se mají vína filtrovat. Jsou to atributy:

- jméno vína,
- výrobce vína,
- pouze uživatelova vína,
- rok výroby,
- obsah cukru,
- obsah alkoholu,
- obsah kyselin,
- barva vína,
- země původu,
- tradiční výraz,
- moštová odrůda.

4.1.1 Kategorizace parametrů pro filtrování

Po konzultaci se zadavatelem práce došlo k rozdělení jednotlivých atributů do kategorií podle typu atributu, hodnot kterých nabývá, rozsahu hodnot a dalších. Tato kategorizace je nezbytná pro zvolení vhodného typu filtru pro atribut kvůli následné implementaci. Například pro filtr podle barvy vína, která nabývá pouze čtyř různých hodnot, je vhodné dát uživateli možnost výběru pouze z těchto možností. Podobně i u obsahu cukru je vhodné vyhledávat vína v určitém rozsahu. Jelikož obsah cukru ve víně je desetinné číslo, je nežádoucím nechat uživatele hledanou hodnotu zadat, neboť pravděpodobnost přesné shody s obsahem cukru některého z vín v databázi je poměrně malá. Všechny vzniklé kategorie filtrů jsou popsány níže.

- Rozsahové filtry:
  Jedná se o filtry vhodné pro číselné atributy. Vstupem flitru jsou dvě hodnoty, první pro minimální hodnotu atributu a druhá pro maximální hodnotu atributu. Výsledkem filtrování jsou pak vína, kde daný atribut leží mezi zadanými hranicemi. Tento typ filtru jsem zvolil pro:
  - ročník výroby vína,
  - obsah cukru,
  - obsah alkoholu,
  - obsah kyselin.

  Všechny tyto atributy jsou buď celočíselné nebo desetinné čísla. Z toho důvodu jsem pro ně zvolil právě tento typ filtru.

- Standardní filtry:
  Tento typ filtru je tvořen pouze jedním vstupem, který není nijak omezen. Je vhodný především pro atributy nabývající velkého rozsahu hodnot a současně se předpokládá, že uživatel zná alespoň část hledané hodnoty. Filtr jsem vybral pro:
  - název vína,
  - výrobce vína.

  Oba atributy jsou řetězce, které mohou obsahovat více slov. Zároveň se předpokládá, že uživatel bude hledat vína od jednoho výrobce, jehož název bude znát. Obdobná situace je i u názvu vína.

- Příznakové filtry:
  Tento typ filtru je vhodný pro atributy nabývající hodnot pouze true nebo false. Je tedy ideální na filtrování uživatelových vín. Pokud je příznak nastaven na true, filtr vrátí pouze uživatelova vína. Jinak jsou vrácena všechna vína.

- Filtry s možností výběru jedné z hodnot:
  Vstupem tohoto filtru je jedna z možností vybraná uživatelem. Filtr je vhodný pokud atribut nabývá pouze několika málo hodnot. Zvolil jsem jej pro barvu vína, která může nabývat pouze čtyř různých hodnot.

- Filtry s možností výběru více hodnot najednou:
  Jedná se o filtry vhodné pro atributy ve vztahu ManyToMany k modelu vína. Vstupem filtru může být více hodnot najednou. Filtr dále obsahuje příznak, zdali se mají filtrovat vína obsahující všechny zadané hodnoty nebo alespoň jednu z nich. Tento filtr jsem vybral pro:
  - moštové odrůdy,
  - tradiční výrazy,
  - země původu.

  Tyto modely jsou totiž vztažené k modelu vína právě vazbou ManyToMany.

4.2 Implementace

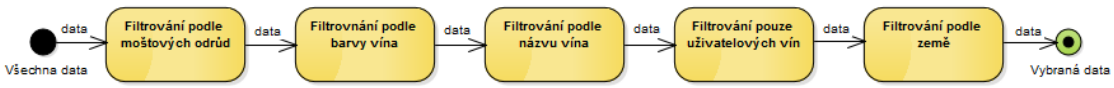
4.2.1 Možnosti implementace

4.2.1.1 Django Forms

Django Forms [11] je standardní knihovna pro vytváření formulářů ve frameworku Django. Vzhledem k faktu, že sada filtrů je vlastně formulářem, byla tato knihovna alternativou pro implementaci. Knihovna využívá ke tvorbě jednotlivých položek formuláře potomků abstraktní třídy Field [15]. Tito potomci reprezentují různé vstupy formuláře (checkbox, radio button, select a další). Pro implementaci jednotlivých typů filtrů z části 4.1.1 by se tedy museli vybrat vhodní potomci. Avšak pro rozsahové filtry tato knihovna neobsahuje vhodného potomka. Muselo by tedy dojít k vlastní implementaci takového potomka. V části 4.2.1.2 je popsán vhodnější způsob implementace tohoto problému.

Dále by při odeslání formuláře muselo ve View [38] dojít k ručnímu filtrování podle zvolených hodnot. V tomto případě není možné vyhnout se vytvoření dlouhého a nepřehledného zdrojového kódu, neboť by bylo nutné rozpoznat, které filtry jsou zvoleny, přičemž některé z filtrů vyžadují poměrně složitý způsob dotazování do databáze. Toto řešení vede k vytvoření velkého množství řádek zdrojového kódu, který je navíc špatně udržovatelný.

4.2.1.2 Django-filter



Obrázek 4.1: Ukázka použití knihovny django-filter jakožto architektonického stylu Pipes and Filters

Django-filter [10] je knihovna pro implementaci filtrování ve frameworku Django. Knihovna, podobně jako Django Forms, obsahuje několik potomků abstraktní třídy, zde Filter [13]. Na rozdíl od Django Forms knihovna obsahuje potomka RangeFilter, který slouží pro implementaci rozsahového filtru. Proces filtrování touto knihovnou odpovídá architektonickému

stylu Pipes and Filters [35]. V tomto stylu tvoří každý filtr samostatnou jednotku, která zpracovává vstup na výstup nezávisle na ostatních filtrech. Přesně takové chování je u filtrování vín očekáváno. Na rozdíl od implementace pomocí Django Forms není potřeba stanovit pořadí provedení filtrů a stačí zde nadefinovat chování jednotlivých filtrů a knihovna django-filter se o datový tok mezi filtry postará sama. Jakákoli budoucí změna tím pádem znamená pouze odstranění, přidání nebo modifikaci daného filtru. Z popsaných důvodů jsem knihovnu django-filter zvolil pro implementaci filtrování vín. Ukázka, jak může vypadat filtrování pomocí knihovny django-filter, je na obrázku 4.1.

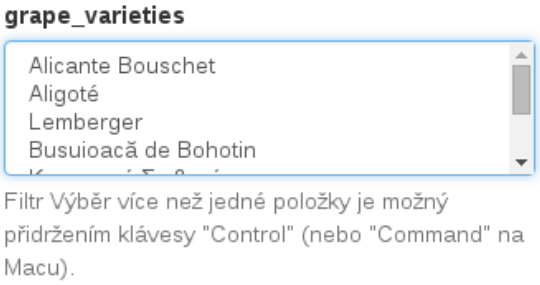
4.2.2 Mapování jednotlivých typů filtrů na třídy z knihovny django-filter

Pro potřebný vzhled a funkcionalitu filtrů jsem musel ke každému typu filtru z bodu 4.1.1 přiřadit třídu jej implementující z knihovny django-filter. Jednotlivé třídy implementují příslušné chování filtru. U každé z nich se může dále zvolit Widget [45], který definuje vzhled filtru. Pokud není zvolen žádný widget, filtr má výchozí vzhled. Seznam uvedený níže toto párování znázorňuje:

- rozsahový filtr - RangeFilter,
- standardní filtr - MethodFilter, widget TextInput,
- příznakový filtr - MethodFIlter, widget CheckboxInput,
- filtr s možností výběru jedné z hodnot - ModelChoiceFilter,
- filtr s možností výběru více hodnot najednou - ModelMultipleChoiceFilter.

4.2.3 Implementace vlastního TokenFieldu a TokenWidgetu

Třída ModelMultipleChoiceFilter z knihovny django-filter zvolená pro filtr s možností výběru více hodnot najednou vygeneruje filtr na obrázku 4.2. Zde se jedná o filtr pro moštové odrůdy.



Obrázek 4.2: Ukázka vygenerovaného filtru pro výběr více hodnot najednou

Podle části 4.1.1 musí filtr obsahovat dále příznak nastavující jeho chování. Vygenerovaný filtr však tento příznak neobsahuje a navíc v možnostech pro výběr zobrazí pouze několik

málo položek. Pokud chce uživatel vyhledat položku, která není ve výběru obsažena, musí ji pomocí scroll baru najít. Počet moštových odrůd v databázi je v řádech tisíců, z toho důvodu není tento způsob výběru hodnot pro filtrování možný. Jako ideální formát vstupu tohoto filtru je zadávání vstupních hodnot jako tokenů, přičemž tokeny vznikají psaním vstupu uživatele a funkce autocomplete pro doplnění vstupní hodnoty podle hodnoty v databázi. Vstup dále obsahuje checkbox jako příznak pro chování filtru. Jako základ pro implementaci jsem použil projekt djtokeninput [53], který mi poskytl požadovaný vzhled a JS chování vstupu. Implementace tohoto token vstupu je složená ze dvou hlavních částí uvedených níže, výsledný vzhled filtru je na obrázku 4.3.

Obrázek 4.3: Ukázka výsledného vzhledu filtru pro výběr více hodnot najednou

4.2.3.1 TokenFilter

Jedná se o třídu, která je potomkem třídy ModelMultipleChoiceFilter [33], tudíž chování filtrování podle více vstupních hodnot je již zajištěno z rodičovské třídy.Třídu jsem implementoval takovým způsobem, aby sloužila jako filtr podle parametru zadaného v kostruktoru. V této třídě bylo dále nutné implementovat chování filtru podle zvoleného příznaku. Třída ModelMultipleChoiceFilter již ve své implementaci obsahuje atribut conjoined, který zajistí chování popsané v části 4.1.1. Avšak k nastavení tohoto atributu dochází v konstruktoru rodičovské třídy. V mém případě je potřeba tento atribut nastavit dynamicky podle uživatelem zvolené hodnoty příznaku. Poté jen stačí zavolat filtrovací funkci rodiče.

4.2.3.2 TokenWidget

Třída TokenWidget je potomkem třídy TextInput [40], což je třída implementující widget pro standardní vstupní pole. Pro implementaci této třídy jsem použil třídu nesoucí stejný název z djtokeninput, která implementuje vstup pomocí tokenů, do které jsem doplnil vykreslení checkbox vstupu pro nastavení příznaku chování. Úkolem této třídy je navíc propagovat uživatelem nastavenou hodnotu příznaku (checkboxu) do třídy TokenFilter. Ukázka inicializace výsledného filtru:

```
vinification_grape_variety = TokenFilter('vinification_grape_variety',
 GrapeVariety, checkbox=True, queryset=GrapeVariety.objects.all(),
  required=False, widget=TokenWidget())
```

Kde TokenFilter je konstruktor třídy TokenFilter. Vinification_grape_variety je název atributu vína, podle kterého se má filtrovat, checkbox=true je informace, zda má filtr obsahovat

checkbox pro nastavení chování filtru a widget=TokenWidget() je nastavení třídy TokenWidget pro zajištění vykreslení filtru. Zbytek argumentů konstruktoru je zděděných z rodičovské třídy, tedy ModelMultipleChoiceFilter.

4.3 Výsledné filtry

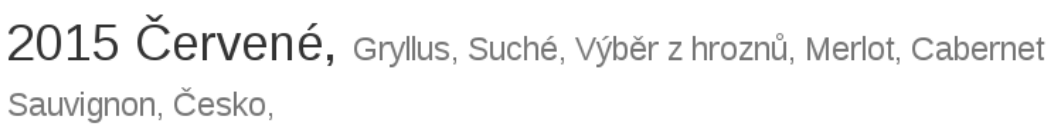
Vzhledem k faktu, že na stránce s filtry se nachází také tabulka obsahující vína, která jsou výsledkem filtrování, byly filtry uspořádány do dvou sloupců kvůli úspoře místa. Na základě konzultace s vedoucím práce však ani tato úspora nebyla dostačující. Rozdělil jsem proto filtry do kategorií základní a pokročilé. Do základních filtrů jsem zahrnul filtr uživatelových vín, filtr podle názvu vína a filtr podle výrobce vína, neboť se domnívám, že právě podle těchto kritérií budou vína nejčastěji vyhledávána. Tato kategorie je zobrazena při načtení stránky s filtry. Zbylé filtry jsou obsaženy v kategorii pokročilých, které jsou přístupné až po rozkliknutí. Výsledný vzhled stránky s filtry a obou kategorií je v příloze 4.3.

Kapitola 5

Fáze 4 - Detail vína

V části 4.3 jsem uvedl, že na stránce s filtry se dále nachází tabulka obsahující vína, která jsou výsledkem filtrování. Dalším krokem této práce bylo umožnit uživatelům systému zobrazovat si o vyhledaných vínech informace o jejich barvě, původu, pěstování apod. Tedy zobrazit si detail vína, jehož realizace je obsahem této kapitoly. Databázový model vína obsahuje velké množství atributů viz kapitola 2. Z toho důvodu bylo nutné rozmístit jednotlivé celky atributů vína, které spolu souvisí, do oddělených částí. Tyto části byly pojmenovány na základě standardu OIV [34] a jednotlivé části jsou popsány samostatně v kapitole níže. Na základě požadavku od zadavatele jsem detail vína implementoval tak, aby neobsahoval atributy, které jsou prázdé nebo nebyly vyplněny. Pro detail vína jsem dále implementoval název vína, který je obsahem části 5.1.

5.1 Název vína



Obrázek 5.1: Ukázka vytvořeného názvu vína

Formát názvu vína byl popsán v dokumentu poskytnutém od externího zadavatele a je postaven na standardu OIV. Pro dobrou viditelnost jsem název vína implementoval jako Bootstrap Page header [6], který navíc obsahuje podporu pro rozlišení důležitých informací a méně důležitých informací v názvu pomocí elementu small. Ukázka, jak může název vína vypadat, je na obrázku 5.1.

5.2 Rozdělení detailu vína dle OIV

Pro oddělení jednotlivých skupin dle OIV jsem zvolil Bootstrap nav-justified [5], který zajistí viditelnost pouze jedné ze skupin vybrané uživatelem. Jednotlivé skupiny jsou:

- Základní charakteristiky:
  Tato skupina obsahuje základní atributy vína jako jsou barva, obsah cukru, kvalita, rok výroby apod. Atributy jsou uspořádány do jedné tabulky tvořené dvěma sloupci pro název atributu a jeho hodnotu. Při načtení stránky s detailem vína je zobrazena právě tato skupina.

- Moštové odrůdy:
  Obsahem této skupiny je seznam moštových odrůd, ze kterých je víno složeno. Pro vytvoření seznamu a přehlednost výpisu jsem zvolil Bootstrap Collapse [4]. Ten vytvoří seznam, kde každý prvek obsahuje název moštové odrůdy a její procentuální podíl ve víně. Po výběru prvku uživatelem je zobrazen detail dané moštové odrůdy. Tento detail obsahuje základní informace o odrůdě, jako její barvu, rok sklizně apod. Informace o původu odrůdy jsou zobrazeny a strukturovány podle standardu OIV. Nakonec jsou zde podrobnosti o sklizni dané odrůdy, pokud existují.

- Informace o složení:
  V této sekci se nachází tabulka obsahující tři sloupce, kde první sloupec obsahuje názvy atributů reprezentujících analytické vlastnosti vína, jako jsou obsah cukru v gramech, obsah kyselin, alkoholu apod. V případě energetických hodnot jsou jména těchto atributů obsažena ve druhém sloupci. Ve třetím sloupci se nacházejí hodnoty těchto atributů.

- Informace o balení:
  Tato skupina obsahuje informace o balení a výrobě vína, jejichž struktura byla vytvořena nebo změněna v konečné fázi vývoje systému jiným členem vývojového týmu viz. kapitola 2. Tato skupina byla proto stejným členem týmu modifikována a její konečná podoba zde proto nebude uvedena.

- Informace o výsledcích:
  Zde se nachází seznam soutěží, na kterých bylo víno hodnoceno a jeho výsledek na příslušné soutěži. Stejně jako u Grape varieties jsem využil Bootstrap Collapse, kde po výběru prvku nesoucího název soutěže dojde k zobrazení detailu výsledku vína na příslušné soutěži. Tento detail obsahuje seznam medailí a ocenění, pokud existují. Dále je zde pro každé kolo soutěže, ve kterém se víno nacházelo tabulka s hodnocením vína určitou komisí v daném kole.

- Informace o osobním hodnocení:
  Obsahem poslední skupiny je tabulka se všemi hodnoceními vína od daného uživatele. Ty mohou být buď v rámci hodnocení uživatele jako člena komise na některé ze soutěží, nebo individuálního hodnocení. V druhém z případů je možné v záhlaví tabulky v příslušném sloupci individuálního hodnocení toto hodnocení změnit.

Kapitola 6

Testování

V rámci této bakalářské práce byly implementovány čtyři funkční celky viz. kapitoly 1 - 4. Z toho důvodu bylo nutné tyto funkční celky testovat zvlášť a také jiným způsobem. Jaké způsoby testování framework django umožňuje popisuje část 6.1, testování databázového modelu vína je obsahem části 6.2, testování klimatické mapy je popsáno v části 6.3 a testování filtrování vín je obsaženo v části 6.4.

6.1 Testování ve frameworku django

6.1.1 Unit testy

Framework django využívá pro unit testování [41] třídu TestCase [39], která využívá standardního modulu unittest [42] pro unit testování v jazyce python.

6.1.2 Automatické testy

Framework django obsahuje nástroj pro tvorbu automatických testů nesoucí název TestClient [59]. Jedná se o mírně odlišný přístup k automatickým testům nežli u Selenium [60]. Podle zadání této práce má být pro automatické testování využit právě nástroj Selenium. Souběžně s touto prací vzniká práce kolegy zabývající se testováním stávajícího systému [54]. V rámci jeho práce byla vytvořena infrastruktura využívající právě Selenium. Budu proto pro automatické testování využívat již vytvořenou infrastrukturu v rámci této práce.

6.2 Testování databázového modelu vína

Jedná se o testování DMV vytvořeného v rámci kapitoly 1. Vzhledem k faktu, že jazyk Python [36] je jazykem dynamicky typovaným, není nutné ani možné testovat správnost uložení jednotlivých atributů modelu vína do databáze, neboť i při pokusu o uložení například číselné hodnoty do pole typu řetězec dojde k automatickému přetypování čísla na řetězec. Je však potřeba otestovat správnou funkčnost způsobu lokalizace názvu moštové odrůdy, která byla v průběhu vývoje často měněna viz kapitola 2. Stačí tedy otestovat metodu get_local_name v modelu moštové odrůdy, která vrací pro stát v argumentu metody název

moštové odrůdy používaný v této zemi. Pokud se daná odrůda v této zemi nepěstuje, vrátí metoda její výchozí název. Vzhledem k tomu, že se jedná o testování modelu, byla použita metoda unit testování. Byly vytvořeny následující test cases:

- TCU00: Odrůda má v dané zemi hodnotu překladu -> metoda tuto hodnotu vrátí,
- TCU01: Odrůda nemá v dané zemi hodnotu překladu -> metoda vrátí výchozí název pro tuto odrůdu.

Oba tyto testy úspěšně procházejí viz. 6.1.

Test	Time elapsed	Results
test_TCU00	4 ms	Passed
test_TCU01	3 ms	Passed

Obrázek 6.1: Průběh unit testování lokalizace názvu moštové odrůdy

Další funkce překladů atributů v DMV není třeba testovat. Je to z důvodu, že o překlad se stará knihovna django-modeltranslation, nedošlo by tedy k testování mojí implementace nýbrž právě této knihovny.

Hlavním účelem DMV v systému je zajištění integrity dat každého vína v databázi. Z toho důvodu je nutné otestovat, zda DMV tuto integritu zajistí při vytváření vína. Pro vytváření vín v systému slouží formulář, který není obsahem této práce, avšak s DMV úzce souvisí. V deklaraci modelu vína totiž u každého atributu dochází k nastavení faktu, zda se jedná o atribut povinný jak v databázi, tak ve formuláři vytváření vína. Pokud tedy bude otestováno vytváření vín ve formuláři, bude otestována i správná funkcionalita DMV. Tuto vlastnost jsem se rozhodl otestovat ručně, tedy akceptačními testy. Vzhledem k řečenému stačí otestovat pouze dva případy vytváření vína a to úspěšné vytvoření vína a neúspěšné vytvoření vína v podobě odeslání prázdného formuláře. V druhém případě musí dojít k informování uživatele o nutnosti vyplnit povinná pole formuláře, která odpovídají povinným atributům v DMV. Vytvořené test cases jsou obsaženy v přiloženém DVD viz. G.1 a testování obou těchto scénářů proběhlo úspěšně.

6.3 Testování klimatické mapy

Veškeré funkcionality klimatické mapy jsou implementovány v jazyce Java Script, automatické testování je tudíž složitější. Je potřeba především otestovat, zda je výsledkem filtrování uživatelů v KM podle zadaného parametru správná skupina uživatelů. To je však pomocí automatických testů problém, neboť je potřeba přístup k implementaci GM API. Z tohoto důvodu jsem se rozhodl otestovat klimatickou mapu ručně, tedy pomocí akceptačních testů. Vytvořil jsem testovací prostředí s uživateli z České Republiky, Slovenska a Jižní Afriky, jejichž vzájemná poloha umožňuje otestovat veškeré vyhledávací funkce klimatické mapy. Seznam navržených test cases je obsažen v přiloženém DVD viz. G.1. Všechny testy prošly úspěšně.

6.4 Testování filtrování vín

Pro otestování filtrování vín jsem zvolil automatické testy. Nejprve byly otestovány veškeré filtry samostatně a poté funkce více filtrů dohromady. Příslušné test cases jsou obsaženy v přiloženém DVD viz. G.1. Všechny testy úspěšně procházejí viz. 6.2.

Filtrování vín obsahuje dále vlastní filtr pro vyhledávání více hodnot najednou viz 4.2.3. Front-endová část tohoto filtru je vytvořena převážně v JS, jedná se tedy o podobnou situaci jako v části 6.3 a stejně tak i tady lze tento filtr otestovat pouze akceptačními testy. Vytvořené test cases jsou v přiloženém DVD viz. G.1 a všechny proběhly úspěšně.

Test	Time elapsed	Results
test_TCFS00	6.562 s	Passed
test_TCFS01	5.244 s	Passed
test_TCFS02	5.315 s	Passed
test_TCFS03	5.142 s	Passed
test_TCFS04	5.01 s	Passed
test_TCFS05	5.084 s	Passed
test_TCFS06	5.292 s	Passed
test_TCFS07	5.128 s	Passed
test_TCFS08	5.476 s	Passed
test_TCFS09	5.29 s	Passed
test_TCFS10	5.222 s	Passed
test_TCFS11	5.376 s	Passed
test_TCFS12	5.172 s	Passed
test_TCFS13	5.204 s	Passed
test_TCFS14	5.323 s	Passed
test_TCFS15	5.342 s	Passed
test_TCFS16	5.522 s	Passed
test_TCFS17	5.015 s	Passed
test_TCFS18	4.917 s	Passed
test_TCFS19	5.192 s	Passed
test_TCFS20	5.372 s	Passed
test_TCFS21	5.829 s	Passed

Obrázek 6.2: Průběh automatického testování vyhledávání vín

Kapitola 7

Zhodnocení

Jak ze zadání této bakalářské práce vyplývá, její náplní byla analýza, realizace a následné testování čtyř samostatných částí systému pro hodnocení vín. Z toho důvodu budu pro přehlednost hodnotit každou z těchto částí zvlášť.

7.1 Zhodnocení optimalizace databázového modelu vína

Hlavním záměrem kapitoly 2 bylo rozšířit a optimalizovat strukturu schématu DMV, aby odpovídalo databázovým normálním formám. Tento cíl se podařilo splnit až na malou výjimku popsanou v části 2.3. Realizace tohoto cíle byla poměrně obtížná, neboť během ní přicházely neustále nové požadavky od externího zadavatele, které znemožňovaly tuto normalizaci databázového schématu provést najednou. Díky tomu docházelo v průběhu k několika neúspěšným krokům, které celou realizaci značně prodloužily viz. 2.4.3. Oproti zadání této práce byl model tudíž značně rozšířen. DMV byl nakonec úspěšně otestován v části 6.2.

7.2 Zhodnocení modulu klimatické mapy

V kapitole 3 se podařilo implementovat všechny požadavky z části 3.2.1, což považuji za úspěch. Domnívám se, že KM může sloužit pro víceúčelové používání, ať už se jedná o porovnávání vín z hlediska klimatické oblasti, ve které byla vypěstována, vyhledávání vinařství nebo třeba pozorování vývoje klimatu. KM byla úspěšně otestována v části 6.3.

Mírným nedostatkem KM je špatná kvalita KML dat pro klimatické oblasti za jednotlivá období. Pouze data pro období 2000-2050 neobsahují příliš ostrých hran a svým vzhledem připomínají opravdu klimatické oblasti. Vzhledem k omezeným testovacím prostředkům viz. 6.3 nebyla KM otestována s větším objemem dat a mohlo by tím pádem při budoucím nárůstu zobrazovaných uživatelů docházet k výkonnostním problémům.

Jako možné vylepšení aplikace klimatické mapy v budoucnu vidím komplexnější návrh systému filtrování založeném na Uživatelském testování použitelnosti [44]. Domnívám se totiž, že k dosažení optimální funkčnosti způsobu vyhledávání je to ten nejlepší způsob. Konečný vzhled KM se všemi jejími prvky je v příloze E.

7.3 Zhodnocení filtrování vín

V rámci kapitoly filtrování vín 4 se podařilo na základě požadavků od externího zadavatele v části 4.2 tuto funkcionalitu úspěšně implementovat a poté v části 6.4 také otestovat.

Podobně jako u 7.2 vidím jako možný další vývoj v podobě uživatelského testování. Tímto krokem by došlo k optimalizaci výběru atributů vína, podle kterých se má filtrovat a dále pak k vylepšení vzhledu samotných filtrů a jejich rozvržení na stránce v prohlížeči.

7.4 Zhodnocení detailu vína

Realizace detailu vína 5 byla nejméně náročnou částí této práce, avšak podobně jako u bodu 7.1 docházelo v průběhu realizace k častým změnám zaviněným postupným dodáváním informací od externího zadavatele. Detail vína v jeho finální podobě však odpovídá konečným požadavkům zadavatele a považuji ho tudíž za úspěšně realizovaný. Jelikož se jedná pouze o zobrazení dat, není potřeba tuto část nijak testovat a případná chyba by souvisela s DMV samotným, tudíž by byla odhalena v části 6.2.

Jako možné budoucí vylepšení detailu vína vidím lepší grafické zpracování jeho vzhledu, které nebylo součástí této práce.

Kapitola 8

Závěr

Z počátku této bakalářské práce došlo k analýze původního databázového modelu vína, který byl poté optimalizován tak, aby splňoval databázové normálním formy. Tento krok byl zároveň nezbytným pro následnou realizaci vyhledávání vín a detailu vína. Tyto části se poté podařilo analyzovat, implementovat a úspěšně otestovat. Nezávisle na předešlých bodech byla postupně také vytvořena analýza, implementace a testy klimatické mapy. U všech předešlých částí se podařilo splnit všechny požadavky plynoucí jak ze zadání práce, tak od externího zadavatele projektu. Původní verzi systému se tedy povedlo v rámci této bakalářské práce úspěšně rozšířit o tři nové funkcionality a jednu funkcionalitu podpůrnou v podobě rozšíření a optimalizace databázového modelu vína, což mělo být jejím hlavním účelem a přínosem. Současná verze systému může sloužit nejenom pro původní účely hodnocení vín, ale i celou řadu účelů nových. Zadání práce se tedy povedlo úspěšně splnit.

Tato práce pro mne byla obrovským přínosem v podobě rozšíření mých znalostí o celou řadu nových zkušeností a technologií. Největší přínos vidím především v programování ve frameworku Django a jazyce Python samotném. Za velice užitečnou dále považuji získanou znalost Google Maps API. Mé dosavadní zkušenosti se softwarovými projekty byly pouze v rámci školních předmětů, proto tuto zkušenost získanou prací na projektu, který je ve spolupráci s průmyslem hodnotím jako nenahraditelnou.

Literatura

[1] Disjunkce, 2016.
<https://cs.wikipedia.org/wiki/Disjunkce>, stav z 2. 5. 2016.

[2] Alfa kanál, 2016.
<https://cs.wikipedia.org/wiki/Alfa_kan%C3%A1l/>, stav z 3. 5. 2016.

[3] Bootstrap, 2016.
<http://getbootstrap.com/>, stav z 20. 5. 2016.

[4] Bootstrap Collapse, 2016.
<http://getbootstrap.com/javascript/#collapse>, stav z 5. 5. 2016.

[5] Bootstrap nav-justified, 2016.
<http://getbootstrap.com/components/#nav-justified>, stav z 13. 5. 2016.

[6] Bootstrap Page header, 2016.
<http://getbootstrap.com/components/#page-header>, stav z 13. 5. 2016.

[7] Bootstrap Tags Input, 2016.
<https://bootstrap-tagsinput.github.io/bootstrap-tagsinput/examples/>, stav z 5. 5. 2016.

[8] CharField, 2016.
<https://docs.djangoproject.com/en/1.9/ref/models/fields/#charfield/>, stav z 14. 4. 2016.

[9] Django domovská stránka, 2016.
<https://www.djangoproject.com/>, stav z 20. 5. 2016.

[10] django-filter, 2016.
<https://django-filter.readthedocs.io/en/latest/>, stav z 12. 5. 2016.

[11] Django Forms, 2016.
<https://docs.djangoproject.com/ja/1.9/ref/forms/api/#django.forms.Form>, stav z 10. 5. 2016.

[12] Modeltranslation, 2016.
<http://django-modeltranslation.readthedocs.org/en/latest/index.html>, stav z 14. 4. 2016.

[13] Filters, 2016.
<https://django-filter.readthedocs.io/en/latest/ref/filters.html#filters>, stav z 12. 5. 2016.

[14] ForeignKey, 2016.
<https://docs.djangoproject.com/en/1.9/ref/models/fields/#foreignkey/>, stav z 14. 4. 2016.

[15] Form fields, 2016.
<https://docs.djangoproject.com/en/1.9/ref/forms/fields/>, stav z 12. 5. 2016.

[16] Geoxml3 Github page, 2016.
<https://github.com/geocodezip/geoxml3>, stav z 4. 5. 2016.

[17] Google Earth, 2016.
<http://www.google.cz/intl/cs/earth/>, stav z 3. 5. 2016.

[18] Google Maps, 2016.
<https://www.google.cz/maps/>, stav z 3. 5. 2016.

[19] Google Maps Geocoding API, 2016.
<https://developers.google.com/maps/documentation/geocoding/intro?csw=1#Geocoding>, stav z 5. 5. 2016.

[20] Google Maps JavaScript API, 2016.
<https://developers.google.com/maps/documentation/javascript/>, stav z 3. 5. 2016.

[21] InfoWindow, 2016.
<https://developers.google.com/maps/documentation/javascript/examples/infowindow-simple>, stav z 5. 5. 2016.

[22] JQRangeSlider, 2016.
<http://ghusse.github.io/jQRangeSlider/>, stav z 5. 5. 2016.

[23] JSON, 2016.
<https://cs.wikipedia.org/wiki/JavaScript_Object_Notation>, stav z 5. 5. 2016.

[24] KML, 2016.
<https://developers.google.com/kml/documentation/>, stav z 3. 5. 2016.

[25] KML Layer, 2016.
<https://developers.google.com/maps/documentation/javascript/examples/layer-kml>, stav z 4. 5. 2016.

[26] KML Polygons, 2016.
<https://developers.google.com/kml/documentation/kml_tut#polygons>, stav z 3. 5. 2016.

[27] Konjunkce, 2016.
<https://cs.wikipedia.org/wiki/Konjunkce_(matematika)>, stav z 2. 5. 2016.

[28] lxml - XML and HTML with Python, 2016.
<http://lxml.de/>, stav z 3. 5. 2016.

[29] MarkerClusterer, 2016.
<https://developers.google.com/maps/articles/toomanymarkers#markerclusterer>, stav z 5. 5. 2016.

[30] Markers, 2016.
<https://developers.google.com/maps/documentation/javascript/markers>, stav z 5. 5. 2016.

[31] Migrace, 2016.
<https://docs.djangoproject.com/en/1.9/topics/migrations/>, stav z 14. 4. 2016.

[32] Modeltranslation, 2016.
<http://django-modeltranslation.readthedocs.org/en/latest/>, stav z 25. 4. 2016.

[33] ModelMultipleChoiceFilter, 2016.
<http://django-filter.readthedocs.io/en/latest/ref/filters.html#modelmultiplechoicefilter>, stav z 20. 5. 2016.

[34] International Organisation of Vine and Wine, 2016.
<http://www.oiv.int/>, stav z 13. 5. 2016.

[35] Pipes and Filters, 2016.
<https://msdn.microsoft.com/en-us/library/dn568100.aspx>, stav z 12. 5. 2016.

[36] Python, 2016.
<https://www.python.org/>, stav z 20. 5. 2016.

[37] RunPython, 2016.
<https://docs.djangoproject.com/en/1.9/ref/migration-operations/#runpython/>, stav z 14. 4. 2016.

[38] Simple view, 2016.
<https://docs.djangoproject.com/en/1.9/topics/http/views/#a-simple-view>, stav z 12. 5. 2016.

[39] TestCase, 2016.
<https://docs.djangoproject.com/en/1.9/topics/testing/tools/#testcase>, stav z 20. 5. 2016.

[40] TextInput, 2016.
<https://docs.djangoproject.com/en/1.9/ref/forms/widgets/#textinput>, stav z 12. 5. 2016.

[41] Unit testování, 2016.
<https://cs.wikipedia.org/wiki/Unit_testing>, stav z 13. 5. 2016.

[42] unittest, 2016.
<https://docs.python.org/3/library/unittest.html#module-unittest>, stav z 20. 5. 2016.

[43] URL, 2016.
<https://cs.wikipedia.org/wiki/Uniform_Resource_Locator>, stav z 20. 5. 2016.

[44] Uživatelské testování, 2016.
<https://cs.wikipedia.org/wiki/U%C5%BEivatelsk%C3%A9_testov%C3%A1n%C3%AD>, stav z 5. 5. 2016.

[45] Widget, 2016.
<https://docs.djangoproject.com/ja/1.9/ref/forms/widgets/#widget>, stav z 20. 5. 2016.

[46] Climate shifts, 2016.
<http://koeppen-geiger.vu-wien.ac.at/shifts.htm>, stav z 2. 5. 2016.

[47] ARLOW, J. – NEUSTADT, I. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* Brno, ČR : Computer Press, 2th edition, 2011. ISBN 978-80-251-1503-9.

[48] EISELT, Z. *Hodnocení vín - GUI. Bakalářská práce.* ČVUT, 2015.

[49] FARANA, R. *Tvorba relačních databázových systémů.* Ostrava, ČR : Vysoká škola báňská-Technická univerzita, 1th edition, 1999. ISBN 80-7078-706-6.

[50] HOLUBOVÁ, I. *Relational design – normal forms* [online]. 2015. [cit. 13. 4. 2016]. Dostupné z: <http://www.ksi.mff.cuni.cz/~svoboda/courses/2015-1-A7B36DBS/lectures/Lecture-08-Relational-Design-Normal-Forms.pdf>.

[51] HONS, D. *Hodnocení vín - soutěž. Bakalářská práce.* ČVUT, 2015.

[52] KOTTEK, M. et al. World Map of the Köppen-Geiger climate classification updated. *Meteorol. Z.* 2006, 15, 3, s. 259–263. doi: 10.1127/0941-2948/2006/0130.

[53] MCKAIG, A. djtokeninput Github web page, 2016.
<https://github.com/adammck/djtokeninput>, stav z 20. 5. 2016.

[54] MIHAL, M. *Testování a rozšíření aplikace pro vinařské soutěže. Bakalářská práce.* ČVUT, 2016.

[55] Přispěvatelé Wikipedie. *Geografický informační systém* [online]. 2015. [cit. 3. 5. 2016]. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=Geografick%C3%BD_informa%C4%8Dn%C3%AD_syst%C3%A9m&oldid\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{1\global\mathchardef\accent@spacefactor\spacefactor}\accent91\egroup\spacefactor\accent@spacefactor3129092>.

[56] RUBEL, F. – KOTTEK, M. Observed and projected climate shifts 1901–2100 depicted by world maps of the Koppen-Geiger climate classification. *Meteorol. Z.* 2010, 19, 2, s. 135–141. doi: 10.1127/0941-2948/2010/0430.

[57] WIEGERS, K. E. *Požadavky na software.* Brno, ČR : Computer Press, 1th edition, 2008. ISBN 978-80-251-1877-1.

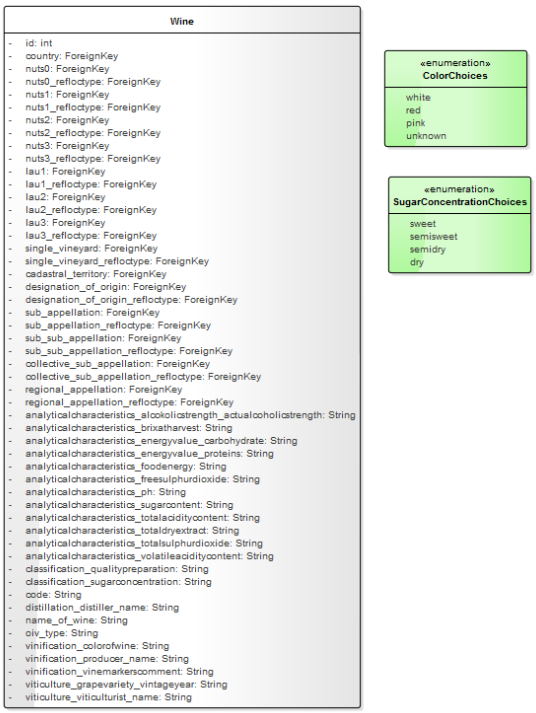
Příloha A

Seznam použitých zkratek

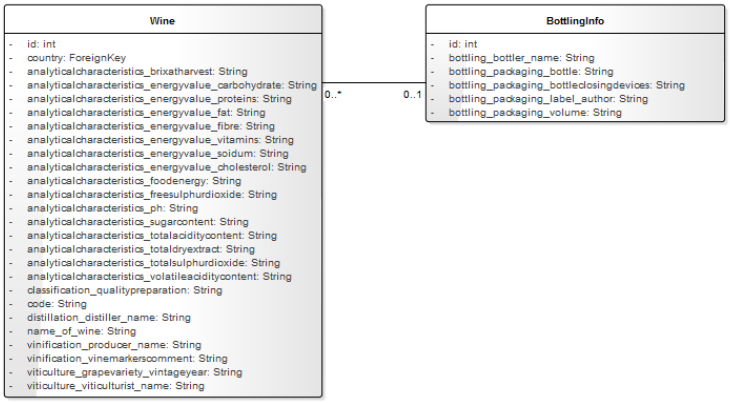
1NF	První normální forma
2NF	Druhá normální forma
3NF	Třetí normální forma
BCNF	Boyce Coddova normální forma
CSV	Comma-separated values
DMV	Databázový model vína
GM	Google Maps
JS	JavaScript
JSON	JavaScript Object Notation
KM	Klimatická mapa
KML	Keyhole Markup Language
OIV	Organisation Internationale de la Vigne et du Vin
SQL	Structured Query Language
URL	Uniform Resource Locator

Příloha B

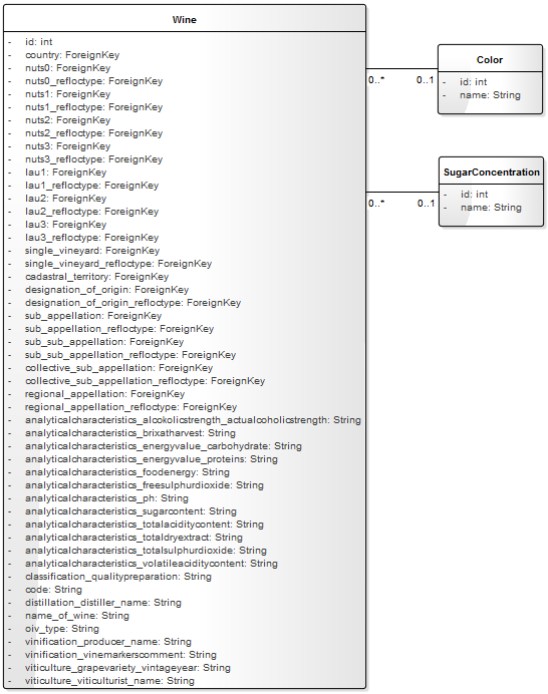
Modely



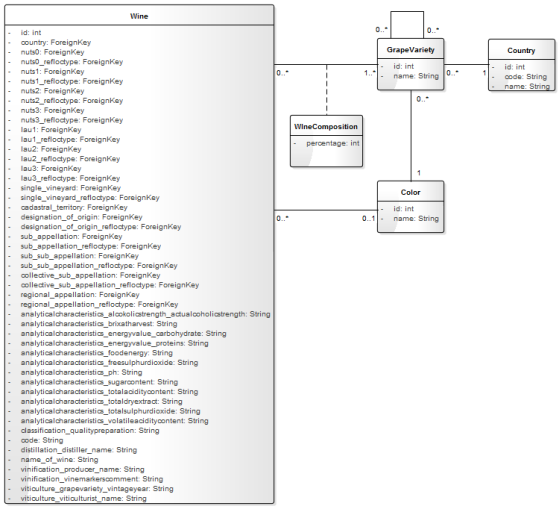
Obrázek B.1: Vytvoření možností pro cukernatost a barvu vína



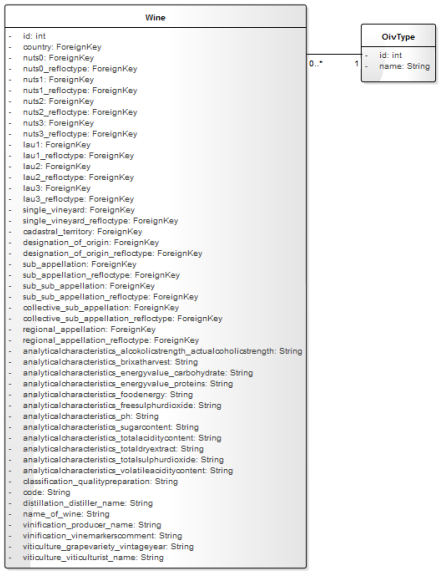
Obrázek B.2: Vytvoření modelu pro balení vína



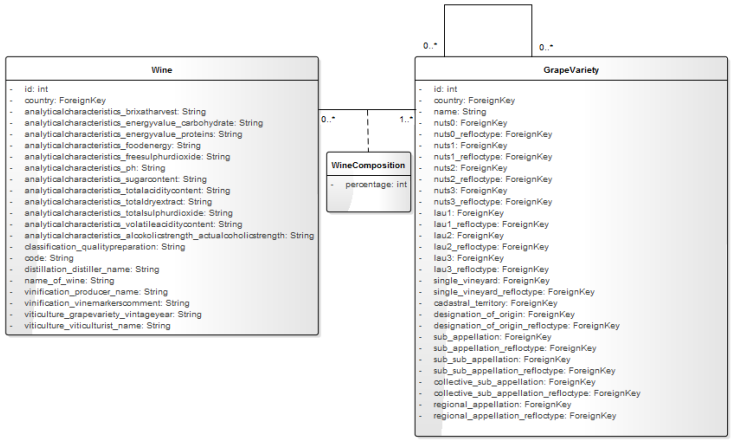
Obrázek B.3: Vytvoření samostatných modelů pro cukernatost a barvu vína



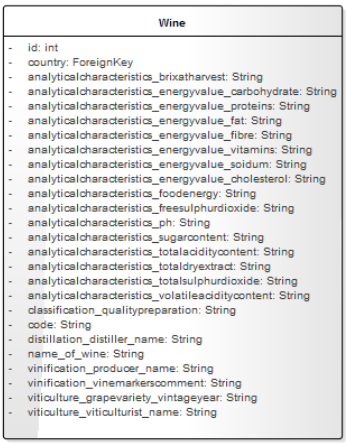
Obrázek B.4: Vytvoření lokalizace názvu moštové odrůdy pomocí synonym



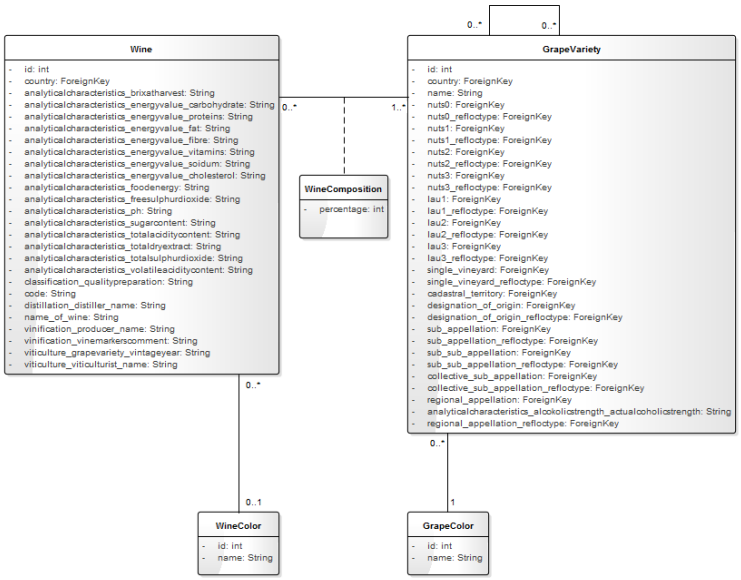
Obrázek B.5: Vytvoření samostatného modelu pro oiv_type



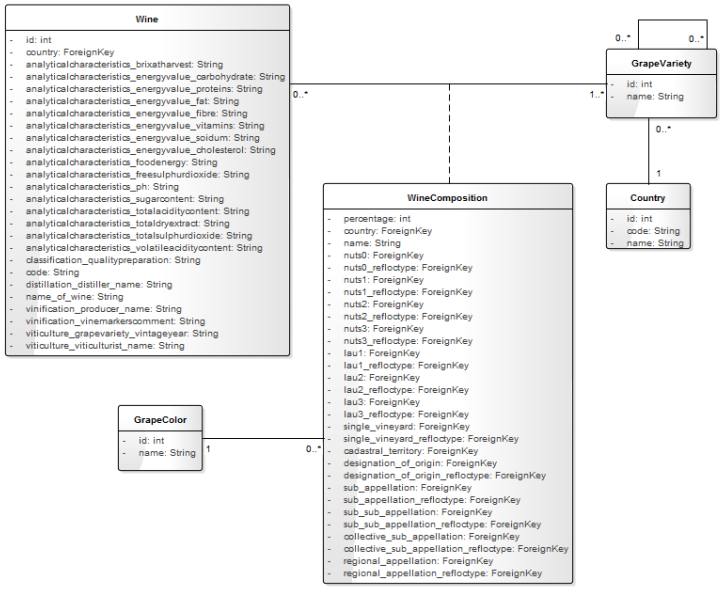
Obrázek B.6: Přesun vazeb na původ vína do modelu moštové odrůdy



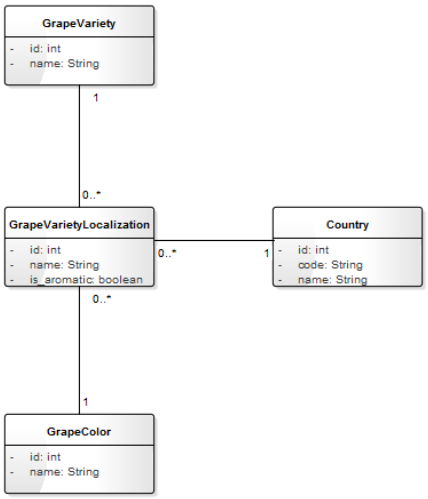
Obrázek B.7: Vytvoření nových atributů pro energetické hodnoty v modelu vína



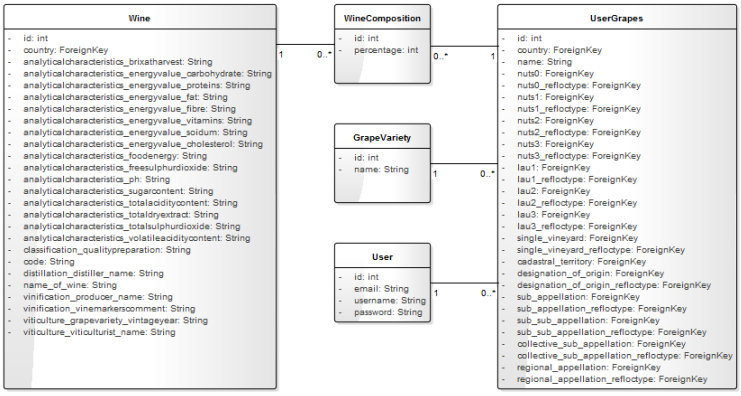
Obrázek B.8: Změna modelu barvy vína na model barvy hroznů a vytvoření nového modelu pro barvu vína



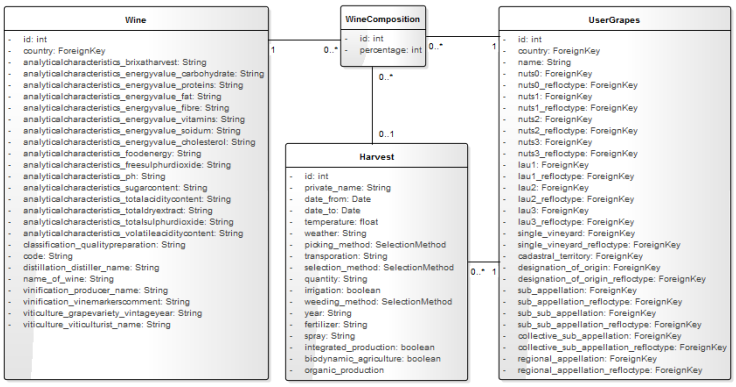
Obrázek B.9: Přesun většiny atributů z modelu moštové odrůdy do vazební tabulky mezi vínem a moštovou odrůdou



Obrázek B.10: Vytvoření modelu pro lokalizaci názvu moštové odrůdy



Obrázek B.11: Vytvoření samostatného modelu pro uživatelovu odrůdu



Obrázek B.12: Přidání vazby na sklizeň do vazební tabulky mezi vínem a uživatelovou odrůdou

Příloha C

Ukázka vytvořeného JSON souboru pro filtrování v Klimatické mapě


```
[
  { "id": 11, "climate": "Equatorial", "precipitation": "fully humid",
   "temperature": "" },
  { "id": 12, "climate": "Equatorial", "precipitation": "monsoonal",
   "temperature": "" },
  { "id": 13, "climate": "Equatorial", "precipitation": "summer dry",
   "temperature": "" },
  { "id": 14, "climate": "Equatorial", "precipitation": "winter dry",
   "temperature": "" },
  { "id": 21, "climate": "Arid", "precipitation": "desert",
   "temperature": "cold arid" },
  { "id": 22, "climate": "Arid", "precipitation": "desert",
   "temperature": "hot arid" },
  { "id": 26, "climate": "Arid", "precipitation": "steppe",
   "temperature": "cold arid" },
  { "id": 27, "climate": "Arid", "precipitation": "steppe",
   "temperature": "hot arid" },
  { "id": 31, "climate": "Warm temperate", "precipitation": "fully humid",
   "temperature": "hot summer" },
  { "id": 32, "climate": "Warm temperate", "precipitation": "fully humid",
   "temperature": "warm summer" },
  { "id": 33, "climate": "Warm temperate", "precipitation": "fully humid",
   "temperature": "cool summer" },
  { "id": 34, "climate": "Warm temperate", "precipitation": "summer dry",
   "temperature": "hot summer" },
  { "id": 35, "climate": "Warm temperate", "precipitation": "summer dry",
   "temperature": "warm summer" },
  { "id": 36, "climate": "Warm temperate", "precipitation": "summer dry",
   "temperature": "cool summer" },
  { "id": 37, "climate": "Warm temperate", "precipitation": "winter dry",
   "temperature": "hot summer" },
  { "id": 38, "climate": "Warm temperate", "precipitation": "winter dry",
   "temperature": "warm summer" },
```

```
  { "id": 39, "climate": "Warm temperate", "precipitation": "winter dry",
   "temperature": "cool summer" },
  { "id": 41, "climate": "Snow", "precipitation": "fully humid",
   "temperature": "hot summer" },
  { "id": 42, "climate": "Snow", "precipitation": "fully humid",
   "temperature": "warm summer" },
  { "id": 43, "climate": "Snow", "precipitation": "fully humid",
   "temperature": "cool summer" },
  { "id": 44, "climate": "Snow", "precipitation": "fully humid",
   "temperature": "continental" },
  { "id": 45, "climate": "Snow", "precipitation": "summer dry",
   "temperature": "hot summer" },
  { "id": 46, "climate": "Snow", "precipitation": "summer dry",
   "temperature": "warm summer" },
  { "id": 47, "climate": "Snow", "precipitation": "summer dry",
   "temperature": "cool summer" },
  { "id": 48, "climate": "Snow", "precipitation": "summer dry",
   "temperature": "continental" },
  { "id": 49, "climate": "Snow", "precipitation": "winter dry",
   "temperature": "hot summer" },
  { "id": 50, "climate": "Snow", "precipitation": "winter dry",
   "temperature": "warm summer" },
  { "id": 51, "climate": "Snow", "precipitation": "winter dry",
   "temperature": "cool summer" },
  { "id": 52, "climate": "Snow", "precipitation": "winter dry",
   "temperature": "continental" },
  { "id": 61, "climate": "Polar", "precipitation": "",
   "temperature": "polar frost" },
  { "id": 62, "climate": "Polar", "precipitation": "",
   "temperature": "polar tundra" }
]
```

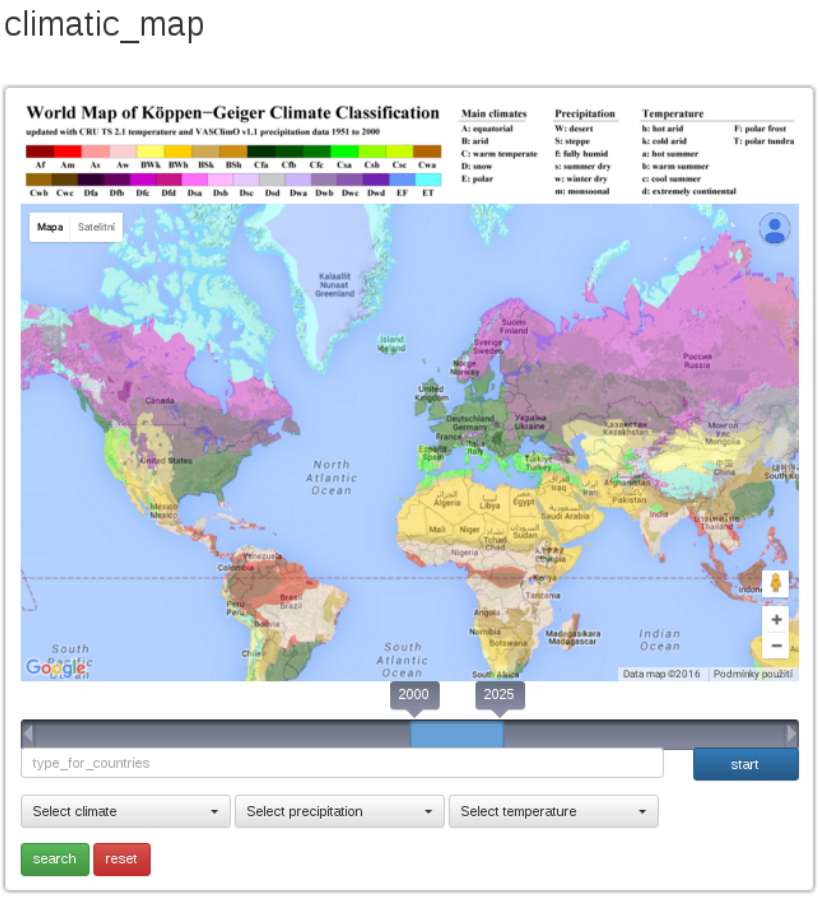
Příloha D

Ukázka funkce pro dynamické překreslování selektů podle vybraných hodnot

```
function changeOptions () {

    var climateValue = $(climateSelect).find("option:selected")
                                                        .text();
    var precipitationValue = $(precipitationSelect).find("option:selected")
                                                    .text();
    var temperatureValue = $(temperatureSelect).find("option:selected")
                                                    .text();

    resetClimateSelects();

    setClimateOptions(climateValue, precipitationValue,
                                        temperatureValue);
    setPrecipitationOptions(climateValue, precipitationValue,
                                        temperatureValue);
    setTemperatureOptions(climateValue, precipitationValue,
                                        temperatureValue);

    refreshClimateSelects();
}
```

Příloha E

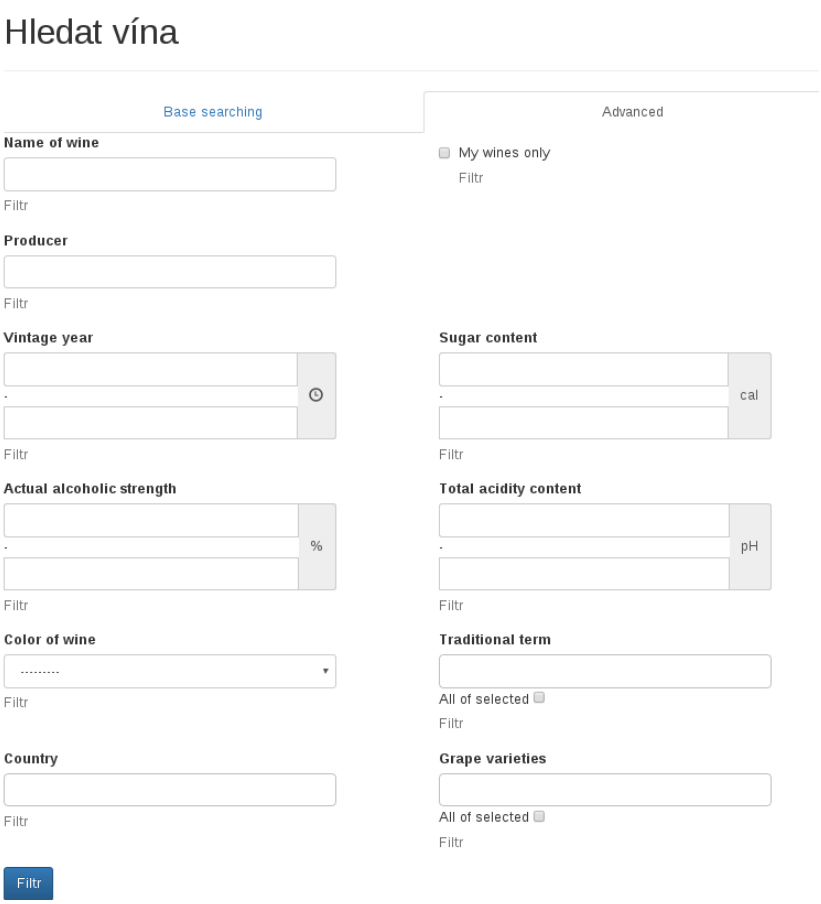
Ukázka konečného vzhledu Klimatické mapy



Obrázek E.1: Ukázka konečného vzhledu Klimatické mapy

Příloha F

Ukázka konečného vzhledu Vyhledávání vín



Obrázek F.1: Ukázka konečného vzhledu Vyhledávání vín

Příloha G

Obsah přiloženého DVD

Obrázek G.1: Obsah přiloženého DVD