UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Damjan Vidonja

Razvoj spletne aplikacije za urejanje datotek JSON z ogrodjem Django

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJ RAČUNALNIŠTVO IN IINFORMATIKA
SMER RAČUNALNIŠKI SISTEMI

Ljubljana, 2016

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Damjan Vidonja

Razvoj spletne aplikacije za urejanje datotek JSON z ogrodjem Django

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJ RAČUNALNIŠTVO IN INFORMATIKA
SMER RAČUNALNIŠKI SISTEMI

MENTOR: doc. dr. Tomaž Dobravec

Ljubljana, 2016

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Format JSON se zaradi preglednosti in enostavnosti uporabe pogosto uporablja za shranjevanje različnih strukturiranih podatkov. V diplomskem delu proučite format JSON in ga primerjajte s formatom XML. Opišite prednosti in slabosti obeh formatov. Opišite definicijo in način uporabe JSON shem, ki zagotavljajo pravilno strukturo datotek JSON. Preglejte in opišite orodja, ki omogočajo urejanje strukturiranih datotek JSON.

Razvijte tudi aplikacijo sistema Django, ki bo omogočala urejanje datotek JSON, katerih struktura naj bo podana v prirejenih shemah JSON. Aplikacija naj bo enostavna za namestitev in uporabo, vsa konfiguracija aplikacije pa naj bo zapisana v datotekah JSON.

Rad bi se zahvalil mentorju doktorju Tomažu Dobravcu za vso pomoč, ki mi jo je nudil med izdelavo diplomske naloge. Posebna zahvala gre mami in vsem prijateljem, ki so me spodbujali in podpirali v času študija.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	Slovensko
JSON	JavaScript object notation	Objektni zapis JavaScript
XML	Extensible markup language	Razširljivi označevalni jezik
HTML	Hypertext markup language	Jezik za označevanje nadbesedila
URL	Uniform, resource locator	Enolični krajevnik vira
CRUD	Create, read, update, delete	Ustvari, beri, posodobi, briši
MVC	Model-view-controller	Model, pogled, krmilnik
MVT	Model-view-template	Model, pogled, predloga
SQL	Structured query language	Strukturiran povpraševalni jezik za delo s podatkovnimi bazami
ORM	Object-relational mapping	Objektno-relacijsko preslikovanje
UML	Unified modelling language	Splošno namenski modelni jezik

Povzetek

Naslov: Razvoj spletne aplikacije za urejanje datotek JSON z ogrodjem Django

Urejanje datotek JSON na podlagi shem JSON predstavlja varen način zagotovitve pravilnosti strukture in vsebine dokumenta. Na tej osnovi je razvita spletna aplikacija, ki nam omogoča vnos podatkov preko spletnih obrazcev. Z razliko od obstoječih rešitev nam omogoča kreiranje več medsebojno povezanih datotek JSON ter nudi podporo urejanja, dodajanja in brisanja dodatnih datotek, ki so definirana v podani shemi. Aplikacija oziroma programska koda je prosto dostopna v repozitoriju GitHub in je tako na voljo širši množici za morebitni nadaljnji razvoj.

Ključne besede: datoteka JSON, ogrodje Django

Abstract

Title: Django Framework web application development for editing JSON files

The main purpose of JSON schema is to define the structure and content of JSON files. The web application created makes use of JSON schema and allows us to input data through input forms. The main difference with other known applications is that we can create a group of related JSON files and also edit, create and delete extra files which are defined in the scheme. The application is free to use and available for everyone.

Keywords: JSON files, Django framework

Poglavje 1 Uvod

Dodajanje in urejanje vnosov datotek JSON terja veliko ročnega dela. Namesto ročnega urejanja bi lahko na podlagi shem JSON generirali spletne obrazce s sprotno preverjanje podatkov. Na ta način bi se izognili zamudnemu urejanju in bi tako ustvarili veljavne datoteke JSON, ki so skladne s prirejenimi shemami JSON.

Obstoječe rešitve niso uporabne, saj ne omogočajo uporabe medsebojno povezanih datotek JSON ter ne nudijo podpore urejanja, dodajanja in brisanja dodatnih datotek, ki so definirana v podani shemi.

V okviru diplomskega dela smo razvili spletno aplikacijo, ki uporablja algoritem za preverjanje podatkov. Rešitev je zasnovana kot prilagodljiva aplikacija Django, ki se lahko na modularni način vključi v druge aplikacije Django. Dosegljiva je na spletni shrambi (*angl. repository*) GitHub.

V Poglavje 2 so predstavljene metode in orodja, ki so uporabljene pri razvoju spletne aplikacije. V drugem delu so predstavljene še podobne rešitve, ki pa ne rešujejo podanega problema. Na podlagi identificiranih pomanjkljivosti obstoječih rešitev je nato podana rešitev, ki je s pomočjo tehnike UML predstavljena v Poglavje 3. Sklepne ugotovitve in možne razširitve so podane v Poglavje 4.

Poglavje 2 Metode in orodja

V tem poglavju je opisan datotečni format JSON in primerjava z XML. Sledi opis sheme JSON in pregled obstoječih preverjalnikov (*angl. validator*). Sledi pregled prirejene sheme JSON, opis atributov in delovanja. V zaključku poglavja je opisana arhitektura ogrodja Django in pregled obstoječih rešitev.

2.1 JSON

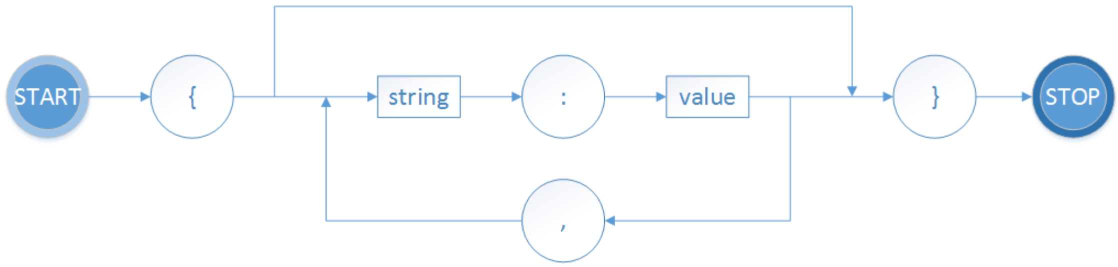
Datotečni format JSON [4],[5] je preprost format za izmenjavo podatkov, zapisan v tekstovni obliki. Podatki v datoteki JSON, so zaradi svoje preproste strukture lahko berljivi in razumljivi (Slika 2.1). Primarno se uporablja za izmenjavo podatkov med spletnimi brskalniki in strežniki in predstavlja alternativo formatu XML.

```
{
    "firstName": "John",
    "lastName": "Smith",
    "isAlive": true,
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    }
}
```

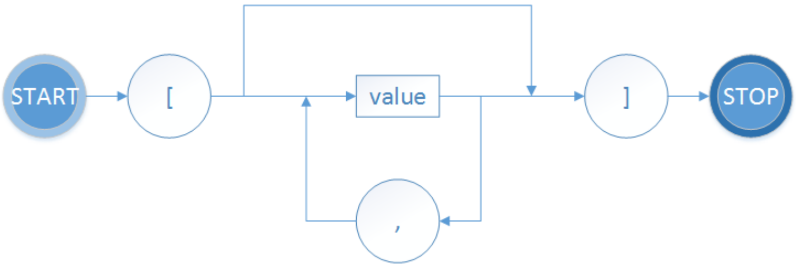
Slika 2.1.: Primer datoteke JSON

Temelji na podmnožici programskega jezika JavaScript. Je neodvisen od jezika, saj uporablja konvencije programerjev jezikov, ki so zasnovani na programskem jeziku C, kot so: C++, C#, Java, JavaScript, Perl in Python. Zaradi tega je format JSON idealen za izmenjavo podatkov.

Osnovni strukturi datoteke JSON sta objekt (*angl. object*) (Slika 2.3), ki predstavlja zbirko parov ključ-vrednost (*angl. key-value*), in tabela (*angl. array*) (Slika 2.3), ki predstavlja tabelo vrednosti. Vrednosti tabele ostanejo v istem vrstnem redu, kot so bile ob vnosu v datoteko JSON. Za razliko od tabel pri objektih JSON to ni zagotovljeno.

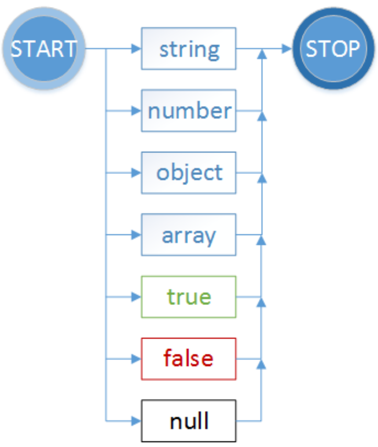


Slika 2.2.: Objekt JSON (primer: `{"Ključ1":42,"Ključ2":"Vrednost2",…}`)



Slika 2.3.: Tabela JSON (primer: `["Ime",3.14,…]`)

Vrednost para ključ-vrednost lahko zavzame različne vrednosti, kot je prikazano na (Slika 2.4). Vrednost tipa `string` je zaporedje nič ali več Unicode znakov, obdanih z dvojnimi narekovaji (primer: `"niz"`). V primeru posebnih znakov, kot so: `"/"`, `"\"`, itd., pa pred njih postavimo t.i. ubežni znak (*angl. escape character*) `"\"`. V primeru, da želimo zapisati niz `"niz\"`, moramo dodati ubežni znak `"niz\\"`. Tip `number` predstavlja vse možne formate števil razen osmiških in šestnajstiških zapisov števil.



Slika 2.4.: Vrednost lahko zavzame različna stanja

2.1.1 Primerjava JSON in XML

Jezik XML [7] je bil razvit leta 1997. Je označevalni jezik za kodiranje dokumentov v formatu, ki je lahko razumljiv tako ljudem kot računalnikom. XML uporablja značke (*angl. tag*) (Slika 2.5), podobno, kot se uporabljajo pri jeziku HTML.

```
<Person>
    <FirstName>John</FirstName>
    <LastName>Smith</LastName>
    <Relatives>
        <Relative>Joe</Relative>
        <Relative>Jaden</Relative>
        <Relative>Mary</Relative>
    </Relatives>
</Person>
```

Slika 2.5.: Primer dokumenta XML

Primer (Slika 2.5) je sicer v obliki datoteke JSON krajši in lažje berljiv, a je manj opisen. Prednost formata JSON je, da za enako količino informacij uporabimo manj podatkov. Nasprotno pa ima JSON zaradi bolj preproste sintakse manj tipov in je zato manj primeren za kompleksnejše nestrukturirane podatke.

```
{
    "firstName": "John",
    "lastName": "Smith",
    "relatives": [ "Joe", "Jaden", "Mary"]
}
```

Slika 2.6.: Primer dokumenta iz (Slika 2.5) v obliki JSON

Format JSON je razcvet uporabe doživel na račun širjenje programskega jezika JavaSript. JSON je podmnožica objektne notacije Javascript, kar pomeni da ima enake tipe podatkov. Za oba formata obstaja shema: shema JSON za format JSON in shema XML za format XML. Za oba formata obstaja orodje za ekstrakcijo globoko gnezdenih struktur, to sta JsonPath in XPath. Ena od prednosti XML je, da ima podporo za imenski prostor (*angl. namespace*). Prednost formata JSON napram XML so lažja berljivost zapisa, razširjenost jezika JavaScript in bolj zgoščeni zapis. Je pa izbira formata odvisna od namena uporabe.

2.2 Shema JSON

Shema JSON [6] opisuje format JSON. Shema ni aplikacija ali algoritem, ampak je enako kot format JSON tudi shema JSON zapisana v formatu JSON. To pomeni, da za shemo JSON veljajo enaka pravila. Shema se uporablja za preverjanje strukture datoteke JSON. S tem nam definira katera vnosna polja so zahtevana ter njihovo obliko in vrednosti. Shema je torej deklarativni format.

Shema s pari »ključ/vrednost« opisuje, kakšna naj bo izhodna datoteka JSON (Slika 2.7). Najbolj preprosto obliko sheme predstavlja zgolj `{}`, ki bi tako definirala vsako veljavno datoteko JSON.

```
{
    "title": "Example Schema",
    "type": "object",
    "properties": {
        "firstName": {
            "type": "string"
        },
        "lastName": {
            "type": "string"
        },
        "age": {
            "description": "Age in years",
            "type": "integer",
            "minimum": 0
        }
    },
    "required": ["firstName", "lastName"]
}
```

Slika 2.7.: Primer datoteke sheme JSON

Z imenom ključa `"type"` lahko definiramo različne tipe, ki jih podamo v vrednost ključa para ključ-vrednost. V primeru `{"type": "string"}` definiramo znakovne nize. Na vrhu sheme vključimo par ključ-vrednost `{"$schema": http://json-schema.org/schema#}` z namenom lažjega razločevanja sheme in datoteke JSON. Vnos je neobvezen, a priporočen.

Shema definira šest osnovnih tipov, ki jih prikazuje spodnja Tabela 1. Vsi tipi datoteke JSON imajo podobne preslikave v programska jezika JavaScript in Python.

JSON	**JavaScript**	**Python**
string	string	string

numeric types	number	int/float
object	object	dict
array	array	list
boolean	boolean	bool
null	null	None

Tabela 1.: Primerjava tipov

Če ključu `"type"` sledi tabela tipov, (npr. `{"type": ["number", "string"]})` to pomeni, da je izhodni podatek lahko ali število ali znakovni niz. Tipu `"string"` lahko sledijo še dodatna pravila (Slika 2.8), kot so `"minLength"`, `"maxLength"` in `"pattern"`, ki definirajo najdaljšo in najkrajšo dovoljeno dolžino znakovnega niza ter v primeru `"pattern"` regularni izraz (*angl. regular expression*), ki definirana obliko niza. S pravilom `"format"` pa lahko dostopamo do že vgrajenih regularnih izrazov za nekatere pogoste vzorce, kot so `"date-time"`, `"email"` itd.

```
{
    "type": "string",
    "minLength": 2,
    "maxLength": 3
}
```

Slika 2.8.: Primer tipa `"string"` z dodatnimi pravili

Numerični tip sheme se še naprej razveji na tip `"integer"` za cela števila in tip `"number"`, ki definira množico vseh števil. Oba tipa imata še dodatna pravila, kot so `"multipleOf"`, `"minimum"`, `"maximum"`, itd.

Tip `{"type": "object"}` definira objekt JSON, kar pomeni, da je izhodna vrednost par ključ-vrednost. Z dodanim pravilom `"properties"` lahko definiramo imena ključev in obliko vrednosti. Slika 2.9 prikazuje objekt z dodatnimi pravili. Pravilo `"enum"` pove, da lahko za ključ `"street_type"` izbiramo med vrednostmi, ki so podane v pripadajoči tabeli. Pravilo `"additionalProperties": false` pomeni, da lahko izbiramo le med danimi ključi `"number"`, `"street_name"` in `"street_type"`, ne smemo pa dodati drugih. Izhodna datoteka `{"number": 1600, "street_name": "Pennsylvania", "street_type": "Avenue","direction":"NW"}` je v tem primeru napačna, saj ima dodaten ključ `"direction"`, ki ni dovoljen.

```
{
    "type":"object",
    "properties":{
        "number":{"type":"number"},
        "street_name":{"type":"string"},
        "street_type":{
            "type":"string",
            "enum":["Street", "Avenue", "Boulevard"]
        }
    },
    "additionalProperties":false
}
```

Slika 2.9.: Primer tipa `"object"` z dodatnimi pravili

Pravilo `"required"` v tabeli definira kateri ključi so obvezni. Za zgornji primer bi `"required": ["number", "street_name"]` pomenilo, da sta ta dva ključa obvezna, vnos ključa `"street_type"` pa ostane neobvezen.

Tip `{"type": "array"}` definira tabelo JSON. Vrednosti znotraj tabele so lahko vseh tipov, kar vključuje tudi tip »object«. To pomeni, da je tabela `[3, "ime", {"ključ": "vrednost"}]` skladna s shemo. Tudi ta tip ima dodatna pravila `"items"`, `"additionalItems"`, `"minItems"`, `"maxItems"` in `"uniqueItems"`. Slika 2.10 prikazuje, da tipi elementov tabele sledijo predpisanemu vrstnemu redu ter da ni dodatnih elementov tabele, saj je vrednost `"additionalItems"` postavljena na `false`. Prvi element tabele je tipa `"number"`. Rezultat je lahko prazna tabela `[]`, en element `[4]`, dva elementa `[4, "Sussex"]` ali trije elementi, pri čemer mora biti zadnji element eden od predpisanih treh v pravilu `"enum"`. Rezultat `["ime"]` je tako napačen, ker je prvi tip definiran kot `"number"`.

```
{
    "type": "array",
    "items": [
        {
            "type": "number"
        },
        {
            "type": "string"
        },
        {
            "type": "string",
            "enum": ["Street", "Avenue", "Boulevard"]
        }
    ],
    "additionalItems": false
}
```

Slika 2.10.: Primer tipa "array" z dodatnimi pravili

Tip {"type": "boolean"} ima dve stanji true ali false, zapiše se brez narekovajev. Zapis "true" je napačen in ni tipa "boolean" temveč, tipa "string". Tip {"type": "null"} ima eno stanje null.

Shema JSON ima poleg naštetih tipov še dodatna pravila, (npr. "title" in "description") ki so bolj opisne narave. Ta pravila niso namenjena preverjanju, temveč so v pomoč pri razumevanju sheme. "title" označuje naslov sheme, "description" se uporablja kot komentar znotraj sheme za posamezen tip elementa.

Definirana so še dodatna pravila, ki združujejo že napisana pravila: "anyOf", "allOf" in "oneOf". Slika 2.11 prikazuje, da je izhodni podatek ali tipa "string" ali tipa "number". V primeru pravila "allOf" bi moral izhodni podatek biti skladen z obema podanima praviloma, kar v tem primeru ne bi bilo možno. Smiselnost pravil je prepuščena razvijalcem. Pravilo "oneOf" definira, da je podatek skladen izključno z enim od pravil.

```
{
    "anyOf": [
        { "type": "string", "maxLength": 5 },
        { "type": "number", "minimum": 0 }
    ]
}
```

Slika 2.11.: Primer pravila sheme "anyOf"

S kombinacijo naštetih tipov je možno tvoriti poljubno kompleksne sheme JSON. Definicija oblike sheme JSON je podana v t.i. meta shemi, ki definira vsa možna pravila, ki jih lahko uporabimo. Tekom pisanja diplomskega dela je najnovejša verzija *JSON Schema, draft v4*.

2.2.1 Algoritmi za preverjanje

Algoritme za preverjanje datotek JSON na podlagi shem JSON lahko razdelimo na dva nivoja preverjanja; strukturni nivo (oblika formata) in semantični nivo (preverjanje z algoritmom). Obstaja več vrst različni preverjalniki, napisani v različnih programskih jezikih (JavaScript, Python itd.) Preverjalniki niso nujno enaki, kajti določeni programski jeziki ne ločijo med celimi števili in plavajočo vejico. Posledično prihaja do nesoglasij med preverjalniki. Npr. preverjalnik napisan v jeziku JavaScript lahko sprejme število 1.0 kot celo število, preverjalnik v jeziku Python pa tega ne omogoča.

Preverjalniki so različnih tipov. Nekateri zgolj preverijo pravilnost datoteke ali sheme JSON. Večinoma preverjalniki vključujejo obe možnosti preverjanja. Primarna naloga preverjalnika je preverjanje skladnosti datoteke JSON na podlagi podane sheme JSON. Obstajajo različne stopnje sporočanja napak, kar seveda zavisi od implementacije podane sheme JSON. Obstajajo številni načini sporočanja napak od specifičnih napak do zgolj izpisa »pravilno/nepravilno« (Slika 2.12).

Slika 2.12.: Primer spletnega preverjalnika »JSON Schema Validator«

Poleg preverjalnikov poznamo še generatorje shem, ki na podlagi podane datoteke JSON izdelajo shemo. Generatorji podatkov pa delujejo v obratni smeri od generatorjev shem – iz sheme generirajo naključne podatke, ki so skladni s podano shemo.

2.3 Prirejena shema JSON

Potreba po izdelavi lastnih shem JSON je prišla na podlagi posebnih zahtev sistema ALGator. ALGator je sistem za izvajanje algoritmov na podanih testnih podatkih ter analizo izvajanja. Sistem omogoča dodajanje in upravljanje s poljubnim številom projektov. V okviru enega projekta je definiran problem, testne množice vhodnih podatkov ter način reševanja nalog problema. Projekt lahko vsebuje poljubno število algoritmov, ki naloge rešujejo na predpisan način. Sistem omogoča analizo izvajanja posameznega algoritma ter primerjavo med algoritmi istega projekta [3].

Sistem ALGator potrebuje veliko režijskega dela, saj je potrebno ročno dodajati veliko število testnih, opisnih in programskih datotek ter izpolniti nastavitvene datoteke JSON. Na podlagi teh pomanjkljivosti se je izdelala lastna prirejena shema JSON, ki se zgleduje po klasični shemi. Prirejena shema ima dodane tipe za rokovanje z datotekami in entitetami. Končnica prirejenega formata je ».atjs«.

2.3.1 Opis atributov

Shema je sestavljena iz 4 osnovnih atributov:

- `"Name"`
- `"Filename"`
- `"Order"`
- `"properties"`

V vrednosti atributa `"Name"` je podano ime sheme. Za primer `"Name":"User"`, bi to pomenilo, da je shema zapisana v datoteki `User.atjs`.

Atribut `"Filename"` kaže na lokacijo, kamor se bo zapisala izhodna datoteka JSON. Primer `"$1/users/USER-{}/{}.json"` nakazuje, da je potrebno dopolniti oznako `{}` ter morebitne dvonivojske parametre, ki so podani preko parametrov `[$1,$2,…]`. V oznako `{}` se vpiše ime entitete na trenutni stopnji. Če je ime entitete `User` enaka `"Joe"` dobimo `"$1/users/USER-Joe/Joe.json"`. Entiteta dobi dvonivojske parametre dobi entiteta od svoje nadrejene entitete. Če npr. nadrejena entiteta `Group` poda parameter `["Group-Admin"]`, dobimo končno relativno pot do izhodne datoteke `"Group-Admin/users/USER-Joe/Joe.json"`.

V datotekah JSON vrstni red parov ključ/vrednost nima garantiranega vrstnega reda, zato imamo dodaten atribut z imenom `"Order"`, v katerem je podana tabela vrstnega reda parov ključ-vrednost. Ta podatek je pomemben pri kreiranju spletnih obrazcev, saj nam poda vedno enak vrstni red.

Atribut `"properties"` vsebuje seznam ključev, ki se zapišejo v izhodno datoteko JSON. Vsak ključ ima gnezdene še dodatne pare ključ/vrednost, ki definirajo dodatna pravila glede na tip podatka. Tipi podatkov so:

- `String`
- `Integer`
- `Double`
- `File`
- `Files`
- `Entity`

Tip `string` definira, da je izhodna vrednost znakovni niz. Kot dodaten atribut je podan `"description"`, v kateri je podan podrobnejši opis. Atribut `"description"` je podan tudi pri vseh ostalih tipih. Atribut `"description"` v spletnem obrazcu uporabljamo kot značko (*angl. label*) pred vnosnim poljem. Možno je dodati neobvezen atribut `"pattern"`, ki z regularnim izrazom definira obliko znakovnega niza. Ta se preveri z algoritmom za preverjanje. V spodnjem primeru (Slika 2.13) je znakovni niz sestavljen iz malih ali velikih črk in je dolžine med 1 in 35 znaki.

```
"properties":{
   "UserName":{
      "description":"Name of the User",
      "type":"string",
      "pattern":"[A-Za-z]{1,35}"
   }
}
```

Slika 2.13.: Primer tipa `string`

Tip `integer` definira, da je izhodna vrednost celo število. Dodatna neobvezna parametra sta `"minimum"` in `"maximum"`, ki določata predpisano zgornjo in spodnjo vrednost izhodnega števila. V spodnjem primeru (Slika 2.14) je število zahtevano število manjše od 99 in večje od 18.

```
"properties":{
   "Age":{
      "description":"User age",
      "type":"integer",
      "minimum":"18",
      "maximum":"99"
   }
}
```

Slika 2.14.: Primer tipa `integer`

Razlika med tipoma `double` in `integer` je tem, da `double` ne predpisuje zgolj celih števil, ampak tudi števila s plavajočo vejico. To pomeni da je `double` nadmnožica tipa `integer`.

Tip `File` in `Files` imata kot dodaten atribut podan `"root"`, ki kaže na direktorij, kamor se bodo naložile uporabnikove datoteke. Enako kot pri atributu `"Filename"` se tudi pri `"root"` s pomočjo imena trenutne entitete in prejetih parametrov nadrejene entitete izgradi relativna pot do izhodnega direktorija. Slika 2.15 prikazuje rezultat izgrajene relativne poti recimo `"Group-Admin/users/USER-Joe/img"`.

```
"properties":{
   "Photo":{
      "description":"User photo",
      "type":"File",
      "root":"$1/users/USER-{}/img"
   }
}
```

Slika 2.15.: Primer tipa `File`

Tip `Entity` ima dva dodatna atributa `"eType"` in `"parameters"`. Atribut `"eType"` kaže na shemo, s katero bomo kreirali novo entiteto. V spodnjem primeru (Slika 2.16) atribut `"eType"` kaže na `User.atjs`. V tabeli atributa `"parameters"` pa je podana tabela dvonivojskih parametrov `[$1,$2,…]`, ki jih bo podrejena entiteta uporabila za izgradnjo poti `"Filename"`. Ti parametri se uporabljajo pri izgradnji poti znotraj atributa `"root"` za tipa `File` in `Files`.

```
"properties":{
   "Users":{
      "description":"Users",
      "type":"Entity []",
      "eType":"User",
      "parameters":["Group-{}"]
   }
}
```

Slika 2.16.: Primer tipa `Entity []`

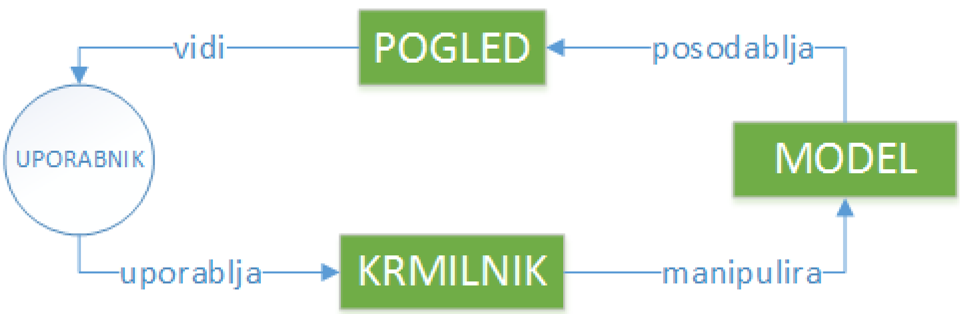
2.4 Ogrodje Django

Django [1],[8] je brezplačno prosto programje (*angl. open-source*). Spletno ogrodje napisano v programskem jeziku Python. Django sledi arhitekturi MVT, ki je zasnovana na arhitekturi MVC. Django vzdržuje neodvisna neprofitna organizacija Django Software Foundation (DSF).

Primarni cilj Djanga je olajšano ustvarjanje kompleksnih spletnih strani, zasnovanih na podatkovnih bazah. Poudarjena je ponovna uporaba kode, modularnost komponent in hiter razvoj, ki je v skladu s principom Django »ne ponavljaj se«. Številne popularne spletne strani, kot npr. Pinterest, Instagram, Mozilla, Washington Post in BitBucket, so narejene v ogrodju Django.

2.4.1 Arhitektura MVT

Arhitektura MVT zasnovana na arhitekturi MVC (Slika 2.17). S širitvijo spleta se je širila tudi arhitektura MVC, saj predstavlja najboljši način načrtovanja aplikacij tipa odjemalec/strežnik.



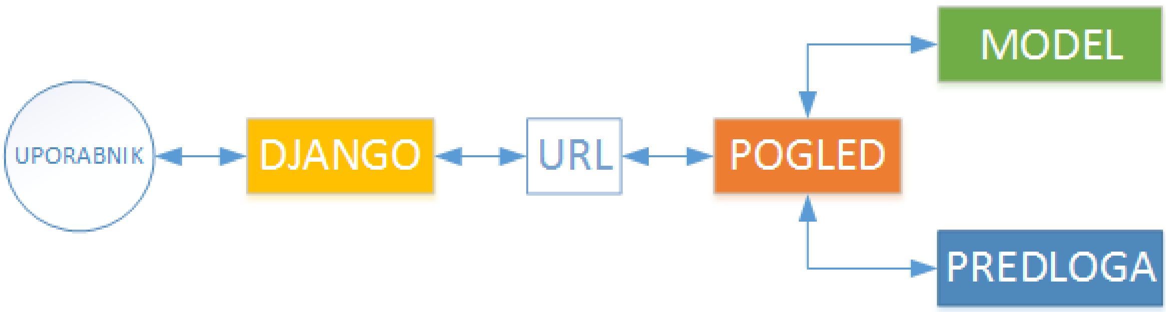
Slika 2.17.: Tipični primer sodelovanja med komponentami MVC

Model (M) je modelna predstavitev podatkov. Model ne predstavlja dejanskih podatkov, ampak definira le vmesnik do njih. Služi temu, da pridobimo podatke brez vednosti o podrobnostih podatkovne baze. Isti model lahko uporabljamo na več različnih podatkovnih bazah.

Pogled (V) (*angl. view*) predstavlja to kar vidi uporabnik. Gre torej za predstavitveni nivo modela. Pogled je to, kar vidimo v spletnem brskalniku ali pa uporabniški vmesnik, ki ga vidimo v namizni aplikaciji. Pogled prav tako nudi vmesnik, preko katerega lahko dobimo podatke od uporabnika.

Krmilnik (C) (*angl. Controller*) kontrolira pretok informacij med modelom (M) in pogledom (V). Uporablja programsko logiko, s katero definiramo katere informacije podatkovne baze pridobimo iz modela in katere informacije pošljemo do pogleda. Podatke od uporabnika pridobimo preko pogleda in z njimi implementiramo poslovno logiko. S slednjo lahko spremenimo pogled ali podatke v modelu.

Pri arhitekturi MVC na Django način so imena in funkcije rahlo spremenjene (Slika 2.18). Model (M) je podatkovni nivo. Ta nivo vsebuje vse kar se tiče podatkov, preverjanja, obnašanja in interakcije med podatki. Predloga (T) (*angl. template*) je predstavitveni nivo. To je odločitveni nivo, ki vpliva na predstavitveno obliko podatkov na spletni strani. Pogled (V) je poslovni nivo. Ta nivo vsebuje logiko, ki dostopa do modela in podatke prepušča do primerne predloge.



Slika 2.18.: Primer kolaboracije arhitekture MVT Django

2.4.2 Pogled (V)

Vsebina spletne strani se generira preko funkcije pogleda `views.py` ter naslova podanega preko naslova URL `urls.py` (Slika 2.19). Pri ujemanju podanega relativnega naslova `/hello/` se s pomočjo regularnega izraza pokliče funkcija Python z imenom `hello`(), ki je podana v datoteki `views.py`. Django se pri tem drži cikla zahteva-odgovor (*angl. request-respone*).

```
from django.conf.urls import include, url
from mysite.views import hello

urlpatterns = [
    url(r'^hello/$', hello),
]
```

```
from django.http import HttpResponse

    def hello(request):
        return HttpResponse("Hello world")
```

Slika 2.19.: Osnovni izpis »Hello world«; levo datoteka `urls.py` in desno `views.py`

2.4.3 Predloga (T)

Predloga Django je znakovni niz teksta, ki je namenjen ločevanju prezentacije dokumenta od njegovih podatkov. Predloga definira mesta, kjer lahko z osnovno logiko reguliramo na kakšen način bo predstavljen dokument. Predloge so večinoma namenjene ustvarjanju dokumentov HTML, a nam Django omogoča tudi ustvarjanje drugih tekstovnih formatov. Za sistem predlog se uporablja jezik predlog Djnago DTL (*angl. Django Template language*). Ta jezik definira 3 osnovne tipe:

- Spremenljivke (*angl. Variable*)
- Značkovne predloge (*angl. Template tag*)
- Filtre (*angl. Filter*)

Spremenljivka predstavlja poljuben tekst, ki je postavljen med dva para zavitih oklepajev `{{spremenljivka}}`. Značkovna predloga je podana znotraj zavitega oklepaja in znaka za procent `{% for i in item %}`. Filtri se uporabljajo za spreminjanje vrednosti spremenljivk na način `{{ spremenljivka|filter }}`. Spodnji primer (Slika 2.20) prikazuje uporabo spremenljivk in značkovne predloge.

```
{% for user in user_list %}
    <p>{{ user.name }}</p>
{% empty %}
    <p>There are no users.</p>
{% endfor %}
```

Slika 2.20.: Primer predloge z značkovno predlogo `for`

V predloge lahko vključimo poljubne druge predloge z ukazom `{% include "name.html" %}`, prav tako lahko trenutna predloga razširjuje poljubno drugo z ukazom `{% extends "base.html" %}`.

2.4.4 Model (M)

Model je opis podatkov v podatkovni bazi s programskim jezikom Python. Model predstavlja podatkovni nivo in je ekvivalent kodi SQL. Zapisan v jeziku Python. Django uporablja model, ki izvršuje kodo SQL izven našega pogleda. Na ta način lahko zamenjamo podložno podatkovno bazo za drugo, pri čemer Django poskrbi, da preslikave ostanejo ekvivalentne. Ta nivo abstrakcije se imenuje ORM. Prednost te abstrakcije je, da ne pišemo kode SQL, ampak kodo Python. Na ta način pohitrimo izvedbo, saj uporabljamo zgolj en programski jezik. Slika 2.21 prikazuje zapisa modela v Python in prikaz enakovredne tabele v SQLite.

```
class User(models.Model):
    name = models.CharField(max_length=30)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
```

```
CREATE TABLE User
(
UserID int,
name varchar(30),
adress varchar(50),
city varchar(60)
);
```

Slika 2.21.: Primerjava modela ogrodja Django in jezika SQL

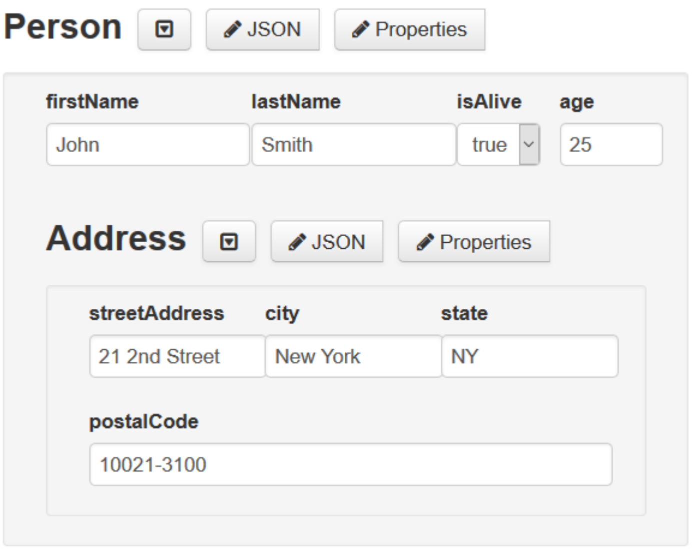
2.5 Podobne rešitve

V nadaljevanju sledi pregled podobnih rešitev, ki na podlagi podanih shem JSON kreirajo izhodne datoteke JSON. Predstavljene so njihove prednosti in slabosti.

JSON Editor [9] kreira spletne obrazce HMTL na podlagi podane sheme JSON. Implementirana je podpora za sheme JSON verzije 3 in 4. JSON Editor nima nobenih posebnih odvisnosti (*angl. dependency*), zahteva le novejši spletni brskalnik. Spletna aplikacija je v napisana v programskem jeziku JavaScript. Implementiranih je veliko dodatnih možnosti, kot je nastavljiv grafični izgled ter različni nabor ikon. Spremenimo lahko strukturno obliko med navadnim pogledom in mrežo. Možnost prikazovanja napak in proženje le-teh je prepuščeno uporabniku.

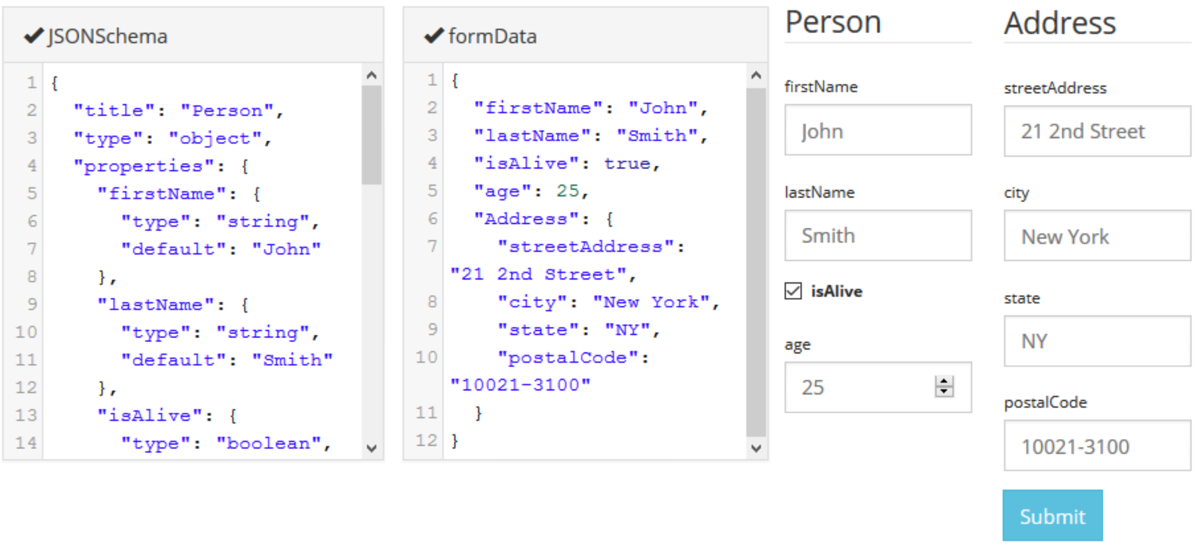
Za primer (Slika 2.1) smo izdelali shemo JSON in jo vnesli v spletno aplikacijo JSON Editor (Slika 2.22). Rezultat je spletni obrazec, ki je zelo intuitiven in pregleden. Na podlagi tega lahko sedaj sestavimo poljubne datoteke JSON, ki ohranjajo enako strukturo podatkov.

Pomanjkljivost spletne strani je, da nima podpore rokovanja s poljubnimi podpornimi datotekami.



Slika 2.22.: Primer JSON Editor

Rešitev *React-jsonschema-form* [10] je v principu podobna JSON Editor, razlikujeta se predvsem v podobi. JSON Editor ima možnost veliko bolj zgoščenih vnosnih polj, kar ga naredi uporabniku bolj prijaznega. Za primer (Slika 2.1) smo ponovno vnesli izdelano shemo v spletno aplikacijo (Slika 2.23). Rezultat datoteke JSON je enak prejšnji.



Slika 2.23.: Primer react-jsonschema-form

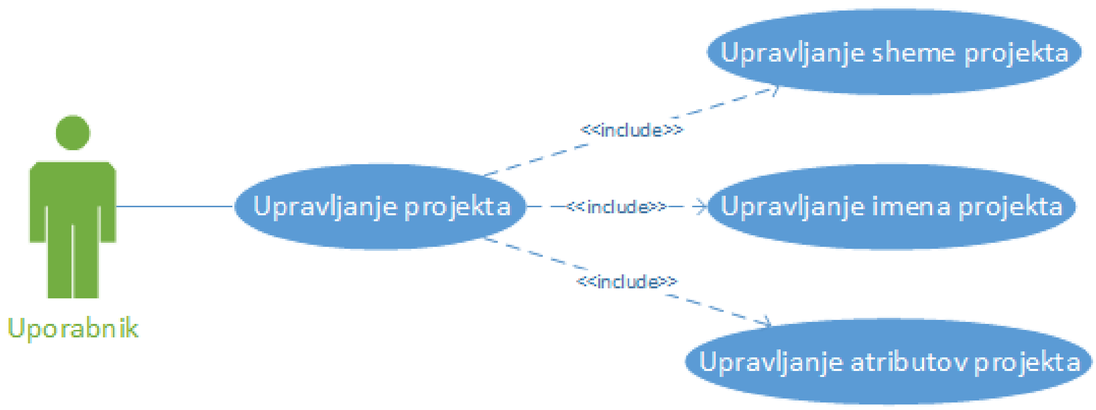
Na spletu najdemo številne rešitve, ki so si precej podobne, a nobena od njih nima podpore za kreiranje več medsebojno povezanih datotek JSON ter nudi podporo urejanja, dodajanja in brisanja dodatnih datotek, ki so definirana v podani shemi.

Poglavje 3 Opis sistema rešitve

V tem poglavju so definirane funkcionalnosti aplikacije Django za urejanje podatkov s predpisano strukturo s pomočjo diagramske tehnike primerov uporabe UML. Prikazani so primeri vnosnih polj aplikacije in rezultatov posameznih primerov uporabe. V nadaljevanju je definirana arhitekturo sistema z uporabo razrednih diagramov UML. Le-ti definirajo strukturo sistema, medtem ko z uporabo diagramov zaporedja realiziramo posamezne primere uporabe.

3.1 Primeri uporabe

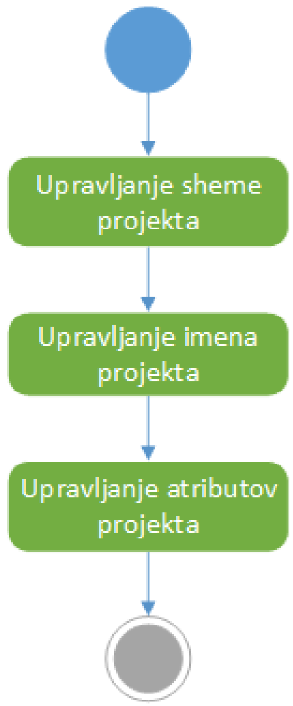
Z osnovnimi tehnikami modelov primerov uporabe UML opišemo funkcionalnost sistema (Slika 3.1). Uporabnikov osnovni primer uporabe sistema je upravljanje projekta. Ta zajema upravljanje sheme, imena in atributov projekta.



Slika 3.1.: Osnovni diagram primerov uporabe

3.1.1 Upravljanje projekta

Ko uporabnik uredi sheme, vnese ime projekta in na podlagi podane sheme vnese atribute projekta, se na podlagi sheme preverijo še atributi.



Slika 3.2.: Diagram aktivnosti

Slika 3.2 prikazuje posamezne akcije diagrama aktivnosti. Znotraj posameznih akcij diagrama se vršijo operacije CRUD. Kratica CRUD pomeni ustvari (*angl. create*), beri (*angl. read*), posodobi (*angl. update*) in briši (*angl. delete*). V akciji upravljanja imena projekta, lahko tega ustvarimo, brišemo ter urejamo, pri čemer sta pri urejanju vključeni operaciji beri in posodobi. Podobno velja za preostali akciji.

Sistem nam omogoča kreiranje datotek JSON, dodajanje in urejanje poljubnih datotek na podlagi opisanih prirejenih shem JSON. Urejanje je omogočeno preko uporabniškega vmesnika spletne strani, katero poganja ogrodje Django.

Rezultat je kreirana datotečna struktura (Slika 3.3). Znotraj korenskega direktorija `G:\projects` je entiteta z imenom `ImeProjekta1`, ki se nahaja v direktoriju `\PROJ-ImeProjekta1`. Datoteka JSON entitete se nahaja v poddirektoriju z imenom `\proj\ImeProjekta1.atp`. Preostali direktoriji te entitete se nahajajo v poddirektorijih `\doc`, `\lib` in `\src`, kot je razvidno z drevesne strukture na podani sliki. V poddirektoriju matične entitete »ImeProjekta1« z imenom direktorija `\algs` se nahaja entiteta `alg1` s podobno podstrukturo. Podobno velja za entiteto `Test1`, ki je prav tako podentiteta entitete `ImeProjekta1` in se nahaja v direktoriju `\tests`. Vseh entitet je lahko poljubno mnogo, odvisno od podane prirejene sheme JSON.

```
G:\PROJECTS
└───PROJ-ImeProjekta1
    ├───algs
    │   └───ALG-alg1
    │       │   alg1.atal
    │       │
    │       ├───doc
    │       │       alg1.html
    │       │
    │       └───src
    │               alg1.jar
    │
    ├───proj
    │   │   ImeProjekta1.atp
    │   │
    │   ├───doc
    │   │       opis.html
    │   │
    │   ├───lib
    │   │       proj1.jar
    │   │
    │   └───src
    │           main.java
    │
    └───tests
        │   Test1.atts
        │
        └───doc
                testset.txt
```

```
{
  "Algorithm" : {
   "ShortName"       : "Kruskal",
   "Description"     : "Kruskal's algorithm",
   "HTMLDescFile"    : "Kruskal.html",
   "Author"          : "Joseph",
   "Date"            : "1956",
   "Classes"         : ["KruskalAlgorithm.java"],
   "MainClassName"   : "KruskalAlgorithm"
  }
}
```

Slika 3.3.: Primer datotečne strukture levo in izhodne datoteke JSON desno

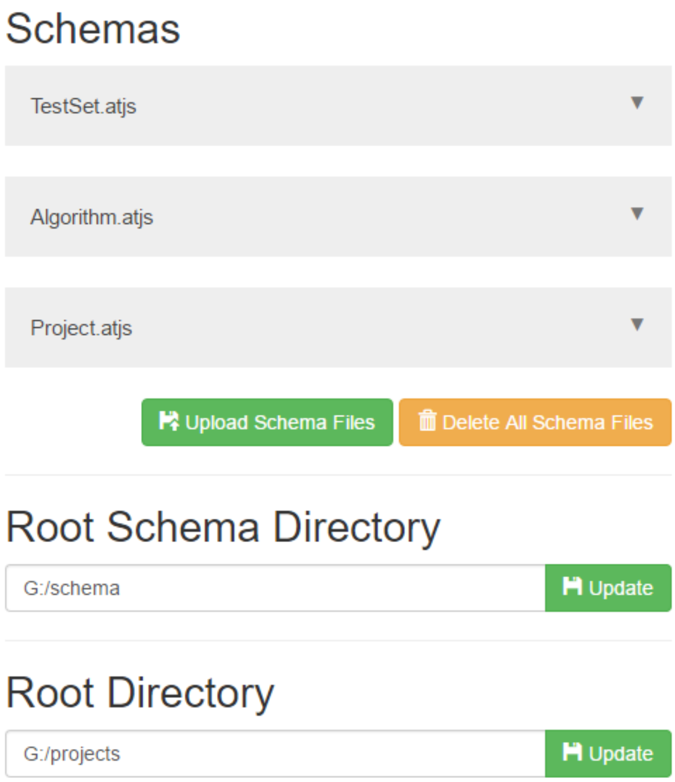
3.1.2 Upravljanje sheme projekta

Slika 3.4 prikazazuje uporabniški vmesnik upravljanja shem projekta. Spletne strani se nahaja na relativnem naslovu `/schema/`.

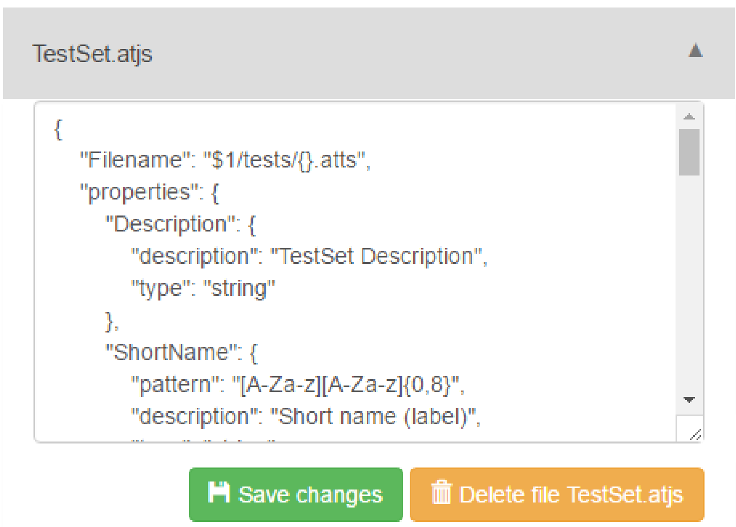
V razdelku »Schemas« je pregled shem JSON, ki so naložene v izbranem direktoriju shem. S tipko »Upload Schema Files« lahko iz svoje lokacije naložimo lastne sheme, katere se shranijo v izbran direktorij. S tipko »Delete All Schema Files« pa izbrišemo vse sheme v izbranem direktoriju. Na elemente seznama shem lahko kliknemo, s tem se nam odpre vnosno področje za tekst (*angl. textarea*), v katerem lahko spreminjamo vsebino sheme (Slika 3.5). S klikom na tipko »Save changes« shranimo spremembe. S tipko »Delete file `<ime.atjs>`« zbrišemo datoteko.

V vnosno polje z imenom »Root Schema Directory« (Slika 3.4) zapišemo pot do direktorija, ki vsebuje sheme. Nastavitev shranimo s pritiskom na tipko »Update«.

Podobno velja za vnosno polje z imenom »Root Directory«. S podanim direktorijem nastavimo pot, v katero se bodo shranjevale naše izhodne datoteke JSON s priležno podatkovno strukturo.



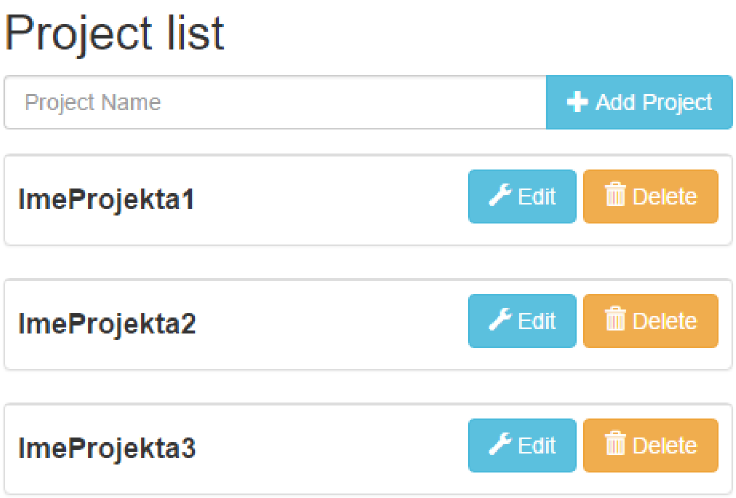
Slika 3.4.: Uporabniški vmesnik upravljanja shem



Slika 3.5.: Uporabniški vmesnik upravljanja shem – posodabljanje vsebine

3.1.3 Upravljanje imena projekta

Slika 3.6 prikazuje uporabniški vmesnik dela spletne strani, ki skrbi za upravljanje imena projekta. Ta stran se nahaja na relativnem naslovu `/index/`.



Slika 3.6.: Uporabniški vmesnik upravljanja imen

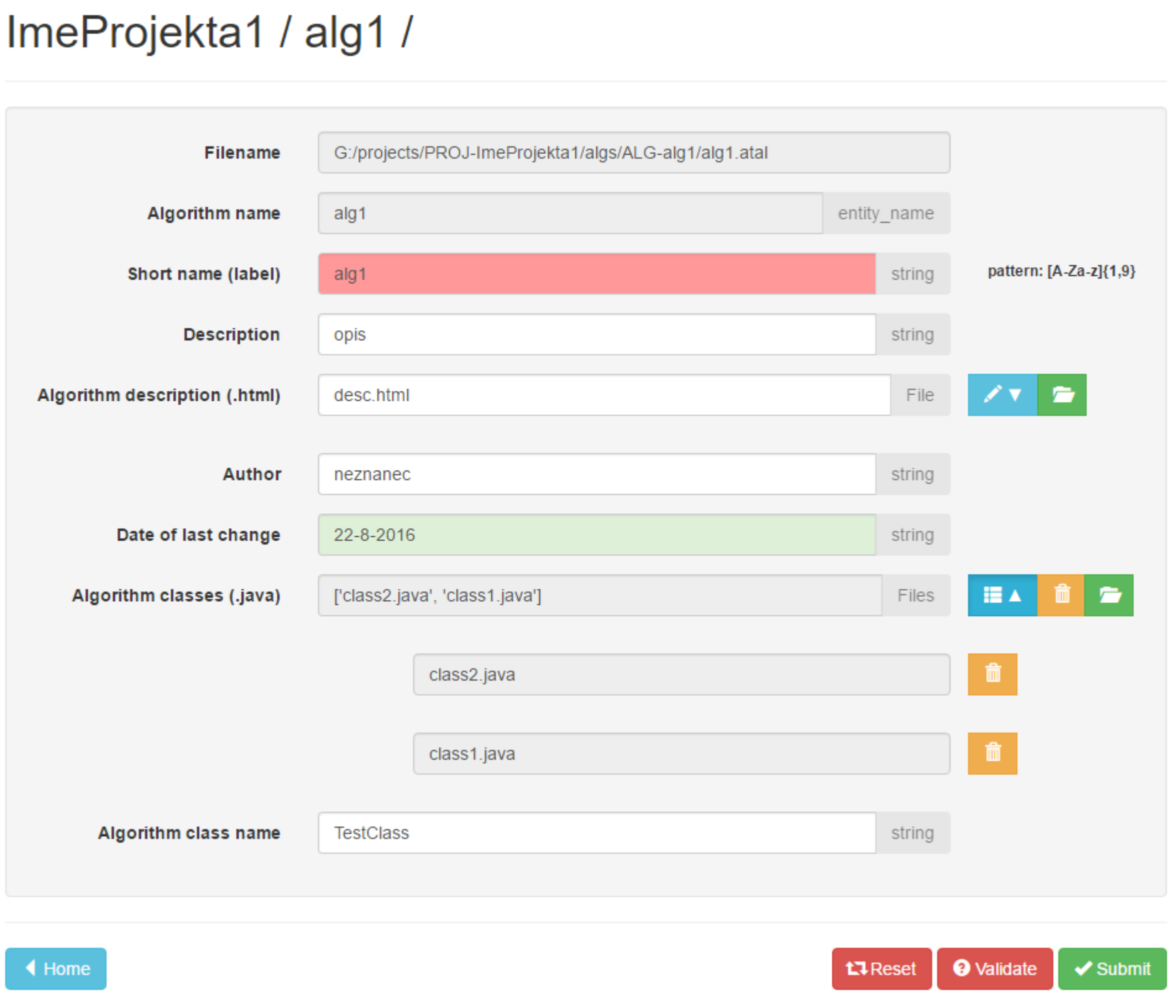
V vnosno polje z imenom »Project Name« vnesemo ime projekta. S tipko »Add Project« ga ustvarimo. Pri tem se generira tudi izhodna datoteka JSON, kamor se bodo ob vnosu atributov shranjevali podatki. Če ima prirejena shema JSON podane atribute tipa `file` in `files`, se preko njihovih lastnosti `"root"` (primer: `"root":"PROJ-{}/proj/doc"`) ustvarijo vsi podani direktoriji. `PROJ-{}` se dopolni, kot je opisano v poglavju 2.3.

Pri brisanju entitete z gumbom »Delete« se izbriše celotna datotečna struktura entitete. V primeru `ImeProjekta1` iz (Slika 3.6) to pomeni, da bo izbrisan direktorij `G:/projects/PROJ-ImeProjekta1`, kot je razvidno iz (Slika 3.3). V tem primeru bodo izbrisane tudi vse podentitete, ki so se nahajale znotraj izbrisanih entitet.

Tipka »Edit« omogoča urejanje izbrane entitete ter s tem posledično tudi vse njene morebitne pod entitete. Klik na omenjeno tipko nas usmeri na relativni naslov `/js2ds/`.

3.1.4 Upravljanje atributov projekta

Slika 3.7 prikazuje uporabniški vmesnik dela spletne strani, ki skrbi za upravljanje atributov projekta. Ta del spletne strani se nahaja na relativnem naslovu »/js2ds/«.



Slika 3.7.: Uporabniški vmesnik upravljanja atributov

V naslovu uporabniškega vmesnika »ImeProjekta1/alg1/« je podana lokacija trenutne entitete. V našem primeru smo v podentiteti z imenom »alg1«, katere nadrejena entiteta je »ImeProjekta1«. S klikom na ime »ImeProjekta1« se lahko vrnemo na obrazec za izpolnjevanje entitete »ImeProjekta1«. S tipko »Home« se vrnemo na upravljanje imena projekta na naslovu `/index/`.

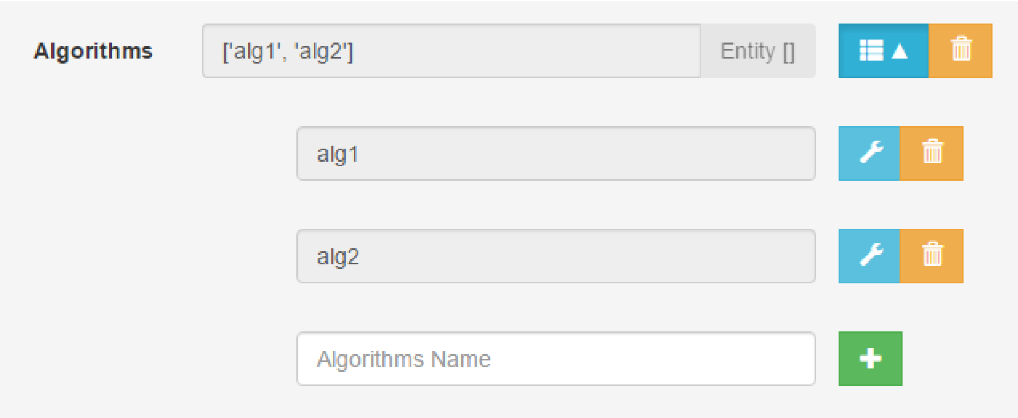
Uporabniški vmesnik je sestavljen iz vrstic, ki jih tvorijo trije deli. Pri tipih `string`, `double` in `integer` je to oznaka (*angl. label*), vnosno polje (*angl. input field*) in morebiten izpis v desnem robu uporabniškega vmesnika, v katerem se izpiše ali minimalna in/ali maksimalna vrednost v primeru tipa `integer` in `double` oz. vzorec (*angl. pattern*) v primeru tipa `string`. Pri tipih `File`, `Files` in `Entity` so v desnem robu tipke.

Pri tipu `File` se s klikom na tipko z ikono svinčnika odpre vnosno področje za tekst, v katerem lahko spreminjamo tekst. S pritiskom na gumb z ikono mapa se nam odpre izbirno okno, v katerem izberemo lokalno datoteko, katera se nato naloži v vnosno področje za tekst. Datoteka se shrani na lokacijo podano s parametrom `"root"`.

Pri tipu `Files` lahko s pritiskom na tipko z ikono mape izberemo več lokalnih datotek naenkrat in jih shranimo na lokacijo podano s parametrom `"root"`. Imamo tudi možnost brisanja vseh oz. posameznih datotek s klikom na tipko z ikono smetnjak.

V vrstici z oznako `"Filename"` je pot do datoteke JSON, v katero se shrani trenutna entiteta. Shranjevanje prožimo s klikom na tipko »Submit«.

Vse spremembe, ki jih naredimo v poljih trenutne entitete, lahko resetiramo na začetno stanje s tipko »Reset«.

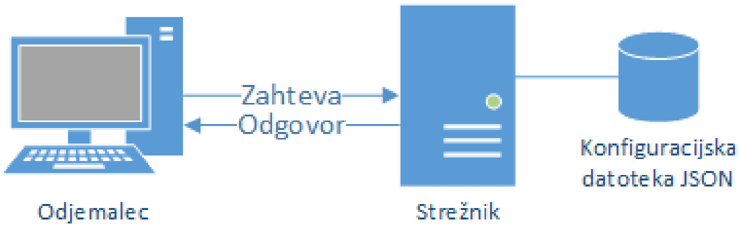


Slika 3.8.: Uporabniški vmesnik upravljanje atributov – upravljanje entitete

Slika 3.8 prikazuje tip `Entity`, kadar je podani v obliki seznama (*angl. array*). `Entity []` ima podani seznam vseh podentitet trenutne entitete. Izbrišemo lahko posamezne entitete ali vse entitete hkrati. Posamezne entitete lahko urejamo s pritiskom na tipko z ikono francoskega ključa. Pri tem se nam odpre nova spletna stran, v kateri urejamo izbrano podentiteto. V vnosno polje z označbo mesta (*angl. placeholder*) »Algorithms Name« vpišemo ime podentitete, ki jo želimo kreirati. Ob pritisku na tipko z ikono znaka plus se kreirajo vse potrebne datotečne strukture. Odpre se nam nova spletna stran, v kateri lahko urejamo novonastalo entiteto.

3.2 Arhitektura sistema

Arhitektura sistema bazira na modelu odjemalec-strežnik (angl. client-server) (Slika 3.9).



Slika 3.9.: Model odjemalec/strežnik

Na odjemalčevi strani je spletni brskalnik (*angl. web browser*), ki preko cikla zahteva-odgovor (*angl. request-respon*) dobi vse potrebne HTML/CSS/JavaScript datoteke, s katerimi lahko upodobi obliko in funkcionalnost spletne strani. Za odzivno (*angl. responsive*) in lepotno obliko spletne strani poskrbi paket z imenom django-bootstrap3.

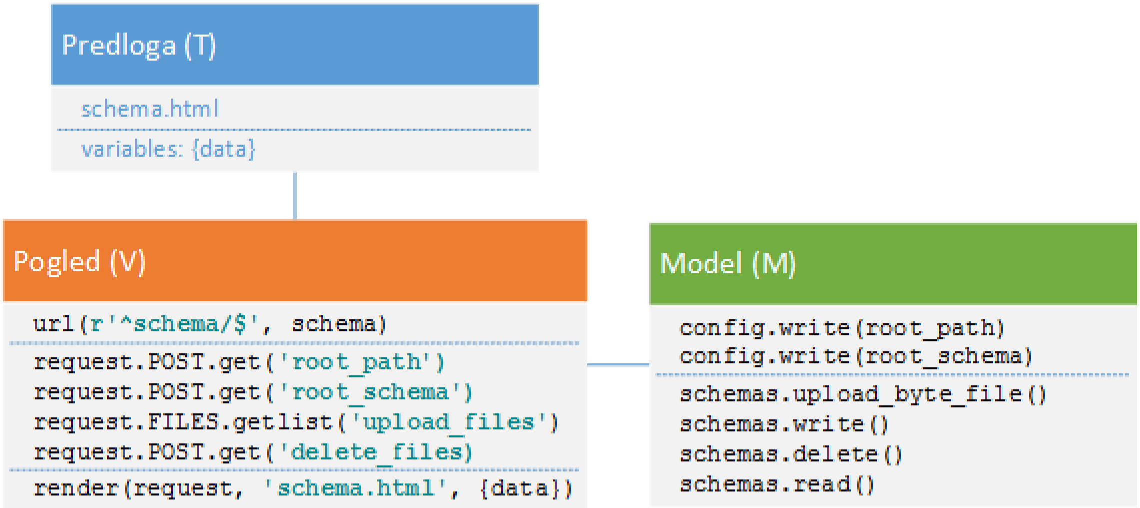
Na strani strežnika teče ogrodje Django, ki za svoje delovanje uporablja Python. Django sprejme zahtevo in jo obdela z metodami Python. Nato naloži ustrezne morebitne podatke iz podatkovne baze. V našem primeru so podatki v datoteki JSON. Nato kreira predlogo (*angl. template*) na podlagi zahtevanih podatkov. Tako tvorjen HTML/CSS/JavaScript se pošlje odjemalcu. Ta proces ogrodja Django se imenuje MTV, kot je opisano v poglavju 2.4.1.

3.2.1 Struktura sistema

Sledi analiza primerov uporabe s prikazom razrednih diagramov UML za vsak primer uporabe. Razredni diagrami so sestavljeni iz treh elementov, ki predstavljajo osnovne elemente arhitekture Django MVT: predlogo (T), pogled (V) in model (M).

3.2.1.1 Upravljanje sheme projekta

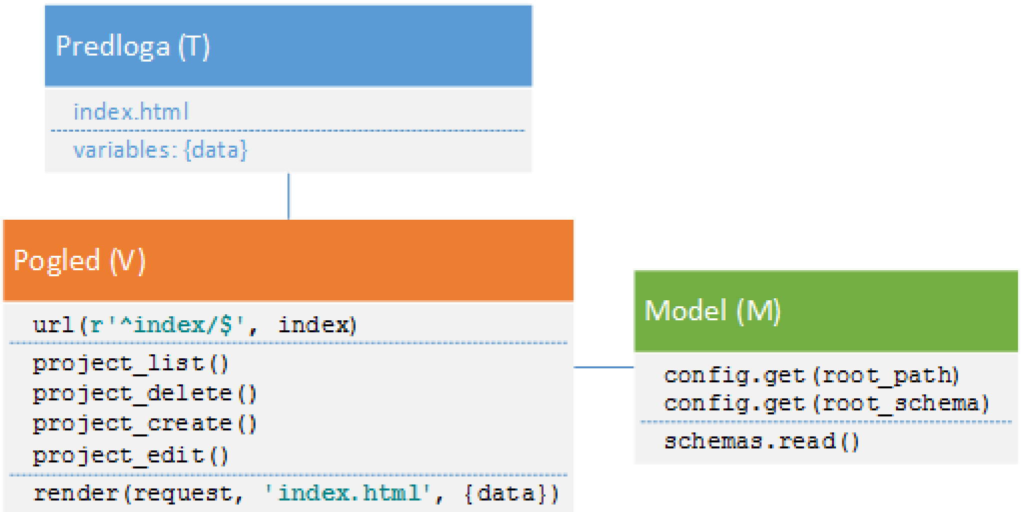
Razredni diagram (Slika 3.10) analizira primer uporabe upravljanja sheme projekta. Pogled (V) sprejete podatke preko metode request posreduje Modelu (M). Model (M) jih zapiše v izhodno konfiguracijsko datoteko. Ko se procedura zaključi, Pogled (V) vključi morebitne podatke preko spremenljivk v Predlogo (T).



Slika 3.10.: Razredni diagram upravljanja sheme projekta

3.2.1.2 Upravljanje imena projekta

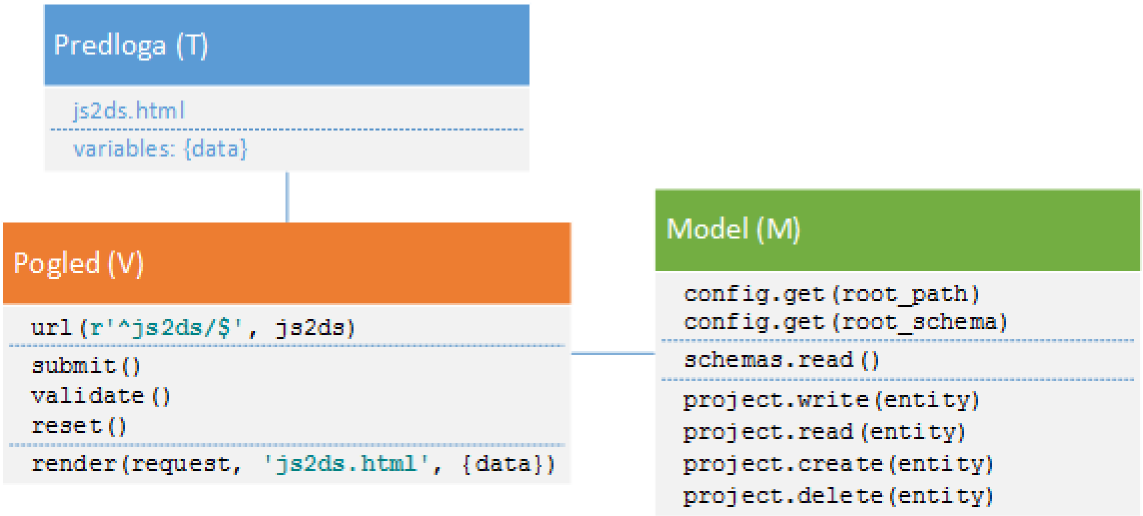
Razredni diagram (Slika 3.11) analizira primer uporabe upravljanja imena projekta. Preko klicev project, ki delujejo po metodi CRUD, se izvedejo vse morebitne operacije nad projektom. Pogled (V) prejme konfiguracijske podatke od Modela (M). Ko se procedura zaključi, Pogled (V) vključi morebitne podatke preko spremenljivk v Predlogo (T).



Slika 3.11.: Razredni diagram upravljanja imena projekta

3.2.1.3 Upravljanje atributov projekta

Razredni diagram (Slika 3.12) analizira primer uporabe upravljanja atributov projekta. Pogled (V) dobi konfiguracijske podatke od Modela (M). Ob klicih znotraj Pogleda (V) se prenesejo podatki v Model (M), kjer se zapišejo v izhodni direktorij. Ko se procedura zaključi, Pogled (V) vključi morebitne podatke preko spremenljivk v Predlogo (T).



Slika 3.12.: Razredni diagram upravljanja atributov projekta

3.2.2 Obnašanje sistema

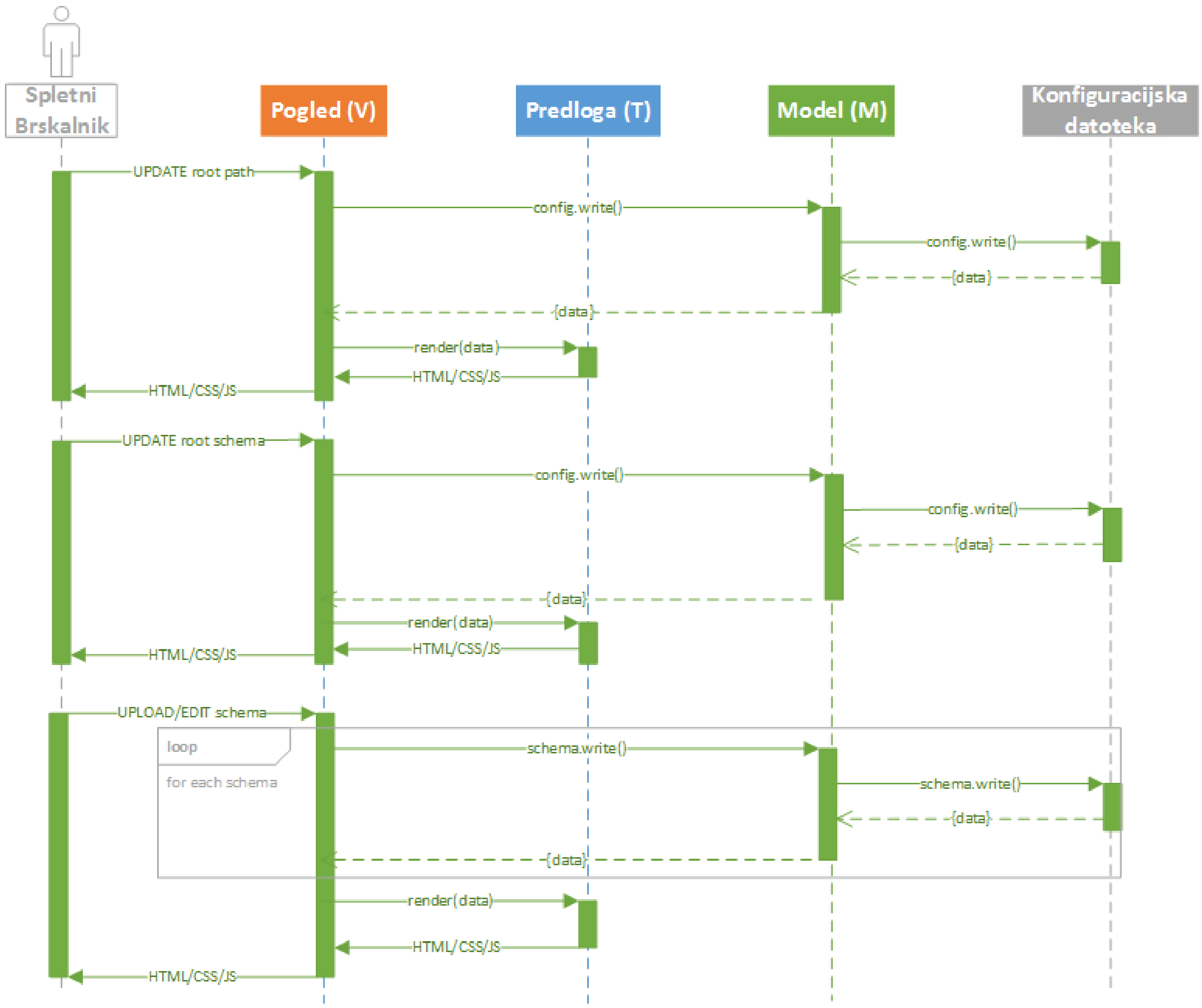
Sledi realizacija primerov uporabe s prikazom sekvenčnih diagramov UML za vsak primer uporabe. Sekvenčni diagrami so sestavljeni iz treh osnovnih elementov, ki predstavljajo osnovne elemente arhitekture Django MVT: predlogo (T), pogled (V),model (M). Dodatna elementa sta akter, ki je ponazorjen kot spletni brskalnik in element, ki predstavlja izhodne podatke.

3.2.2.1 Upravljanje sheme projekta

Sekvenčni diagram (Slika 3.13) realizira primer uporabe upravljanja sheme projekta. Na njem so podani trije možni sekvenčni toki izvedbe, ki prikazujejo interakcije med brskalnikom, elementi Django MVT in konfiguracijsko datoteko.

V prvih dveh primerih prikazuje posodobitev korenskih imenikov sheme in izhodnega direktorija. Tok interakcije se začne pri brskalniku, ko uporabnik zahteva spremembo. Ta pride do Pogleda (V), ki zahtevo poda Modelu (M), slednji pa jo zapiše v konfiguracijsko datoteko.

Tretji tok prikazuje manipulacije s shemami. V primeru operacij CRUD se podobno kot v prvih dveh primerih izvrši zapis v izhodne datoteke. V tem primeru se lahko izvaja več operacij hkrati, zato se notranji tok lahko ponovi večkrat.



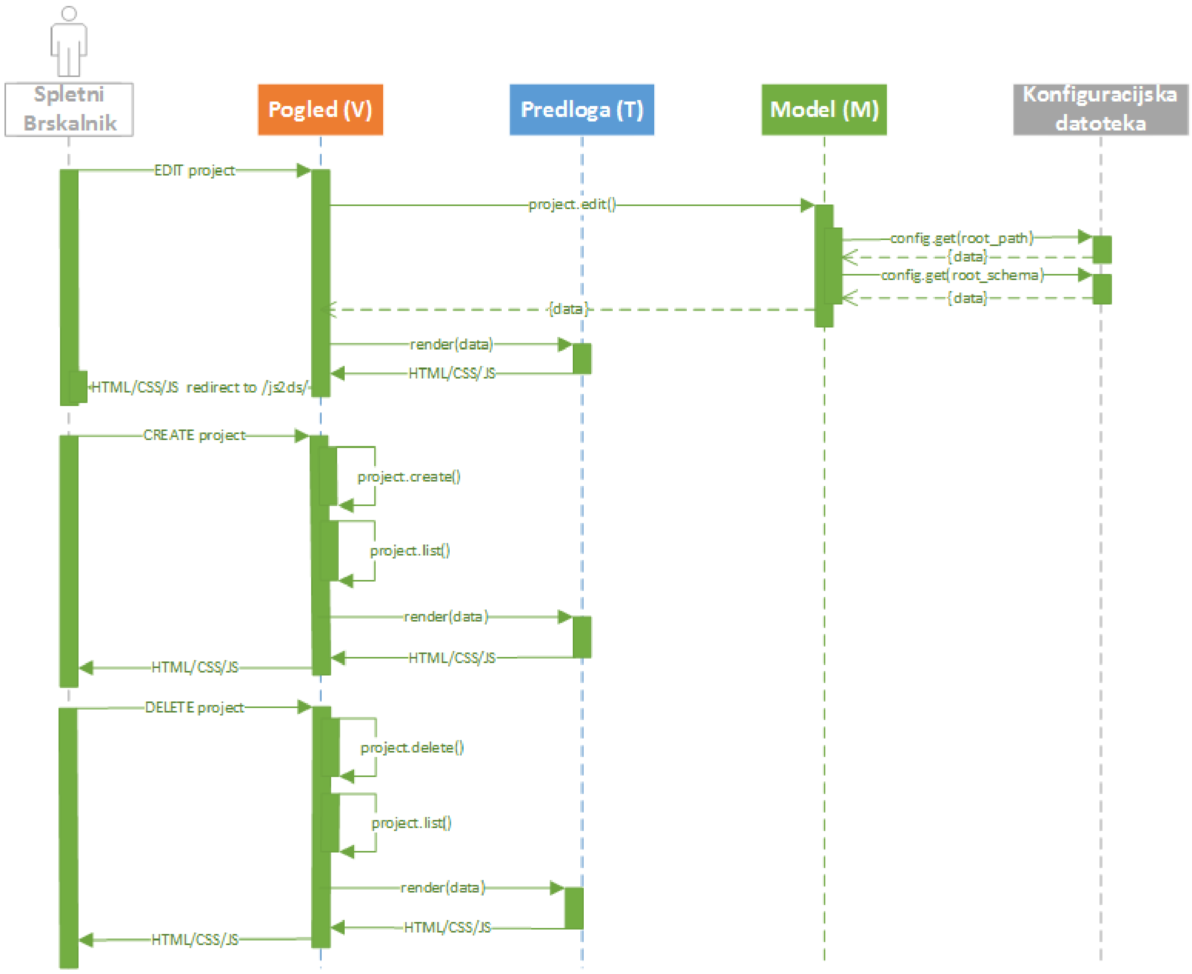
Slika 3.13.: Sekvenčni diagram upravljanja sheme projekta

3.2.2.2 Upravljanje imena projekta

Sekvenčni diagram (Slika 3.14) realizira primer uporabe upravljanja imena projekta. Elementi na njem so enaki kot v prejšnjem primeru.

V prvem primeru je prikazano urejanje projekta. Pri tem mora Pogled (V) pridobiti podatki iz konfiguracijske datoteke preko Modela (M). Z pridobljenimi podatki preusmeri spletni brskalnik na drugo stran spletne aplikacije.

Zadnja toka predstavljata kreiranje in brisanje projekta. Pri tem se izvede klic za brisanje oz. kreiranje, kateremu sledi klic za ponovni izpis posodobljenega seznama projektov. Podatki se pošljejo v Predlogo (T) in posodobijo, novo osveženo spletno stran pa servira Pogled (V) nazaj k spletnemu brskalniku.



Slika 3.14.: Sekvenčni diagram upravljanja imena projekta

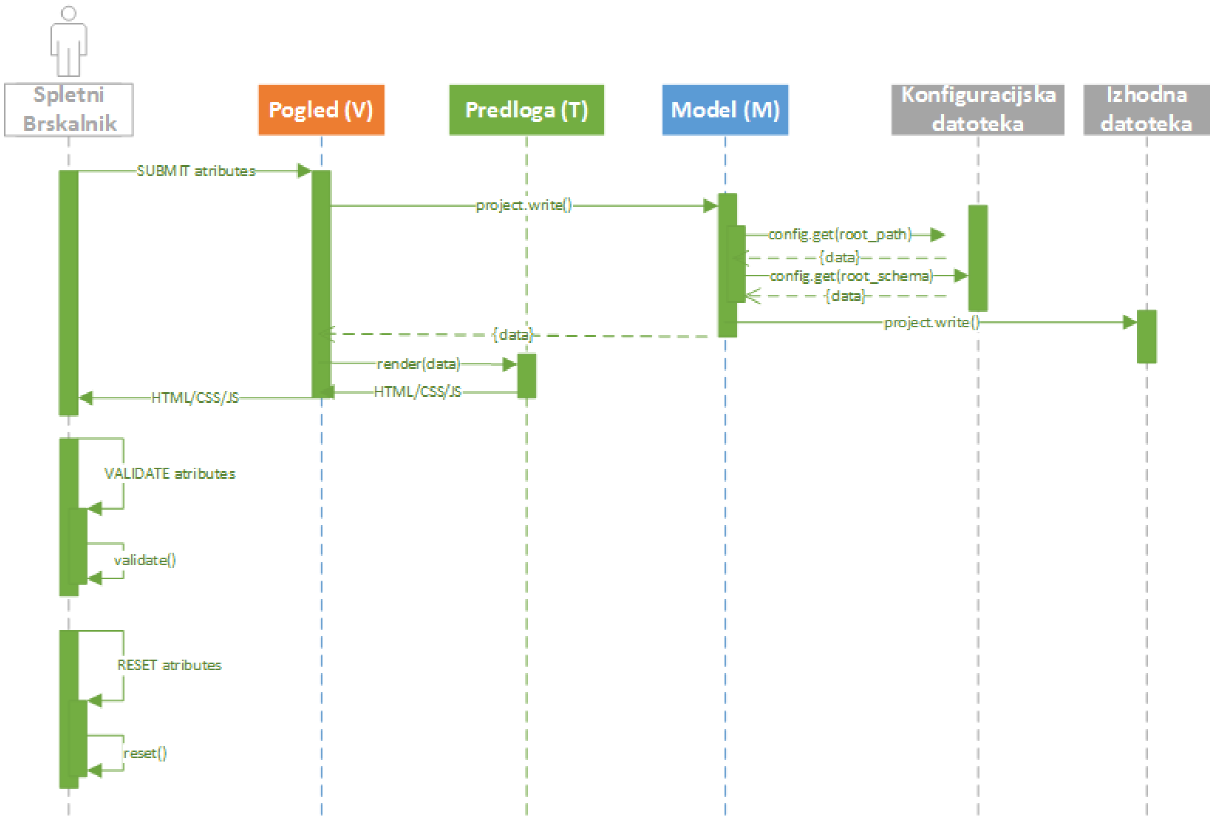
3.2.2.3 Upravljanje atributov projekta

Sekvenčni diagram (Slika 3.15) realizira primer uporabe upravljanja atributov projekta. Elementi na njem so enaki kot v prejšnjih primerih z dodatnim elementom »Izhodna datoteka«.

V prvem primeru prikazujemo shranjevanje vnesenih atributov v spletnem obrazcu. Pogled (V) preda podatke Modelu (M), ki pridobi podatke iz konfiguracijske datoteke. Nato zapiše podatke v izhodni korenski direktorij.

Drugi tok prikazuje postopek preverjanja, ki se izvrši kar na spletnem brskalniku, saj je preverjevalnik zapisan v programskem jeziku JavaScript.

V tretjem primeru je funkcija reset, ki je del jezika HTML.



Slika 3.15.: Sekvenčni diagram upravljanja atributov projekta

3.3 Algoritem za preverjanje

Ob vsej množici tipov podatkov, ki jih imamo na voljo, je treba poskrbeti za pravilnost podatkov. Preverjanje se izvaja na odjemalčevi strani. Za to skrbi funkcija `validate()`, ki je zapisana v programskem jeziku JavaScript. V primeru uspešnega preverjanja se vnosno polje obarva z zeleno barvo, v nasprotnem primeru pa z rdečo.

Pri podatkovnem tipu `string` je na voljo opcija dodajanja vzorca (*angl. pattern*), ki nam definira obliko podatkov. Vzorec je podan v obliki regularna izraza. Primer regularnega izraza npr. `/^[A-Z]{3}[0-9]{1,2}$/` pomeni, da si najprej sledijo tri velike črke med A in Z, pa sledi ena ali dve števki med 0 in 9. Znak `^` pomeni, da primerjamo niz od njegovega začetka. Znak `$` pa pomeni, da niz primerjamo do konca. Regularni izrazi so lahko precej kompleksni, z njimi se definira podatke v obliki elektronskega naslova, datuma, itd. Primerjavo izvedemo z ukazom `pattern.test(str)`, pri čemer je `pattern` regularni izraz, `str` pa niz, ki ga primerjamo. Metoda `test()` vrne vrednost `True`, če niz ustreza podanemu vzorcu regularnega izraza.

Pri podatkovnih tipih `integer` in `double` je na voljo opcija dodajanja maksimalnega in minimalnega števila. Algoritem za preverjanje preveri ali so vnosi znotraj podanih meja. Preveriti je potrebno pravilnost tipov, v primeru tipa `integer` preverimo, če je res celo število, pri tipu `double` pa če imamo opravka s številom s plavajočo vejico.

3.4 Algoritem za generiranje vnosnih obrazcev

Algoritem za generiranje vnosnih obrazcev se nahaja na strežniku. Napisan je v programskem jeziku Python. Obrazce generiramo na podlagi podane sheme JSON. V shemi je podan ključ z imenom `"Order"`, preko katerega dobimo vrstni red prikaza vnosnih polj na spletni strani. Ključ z imenom `"properties"` nam poda novi objekt JSON, preko katerega dobimo elemente vnosnih polj. Znotraj teh dobimo pomembne opise posameznega elementa, kot je opisano v poglavju 2.3.

Za vsak element znotraj ključa `"properties"` izrišemo vnosno polje. Oznaka ob posameznem vnosnem polju je podana s ključem `"description"`. Preko ključa `"type"` dobimo informacijo o tipu elementa in na podlagi tega izberemo predlogo. Pri tipu `string`, `double` in `integer` izpišemo desno od vnosnega polja še opcijske informacije, kot so `"pattern"`, `"minimum"` in `"maximum"`.

Za tipe `File` generiramo tipke »skrij/prikaži«, »briši« in »naloži«. S tipko »skrij/prikaži« lahko skrijemo oz. prikažemo vnosno polje za tekst, v katerem lahko urejamo tekstovno datoteko. Pri tipu `Files` generiramo tipke »skrij/prikaži« za pregled vseh naloženih datotek. Pri posameznih datotekah imamo tipko »briši« s katero zbrišemo datoteko v izbrani vrstici. Pri tipu `Files` imamo možnost izbrati več datotek hkrati. Da lahko z vnosnim obrazcem jezika HTML naložimo datoteko, je potrebno v značko (*angl. tag*) `<form>` dodati `enctype="multipart/form-data"`. Pri tipki »naloži« pa zaradi podpore nalaganja večih datotek hkrati dodamo oznake `type` atribut `multiple`. Tip tipke pri `File` in `Files` je tipa `file` (Primer: `<input type="file">`).

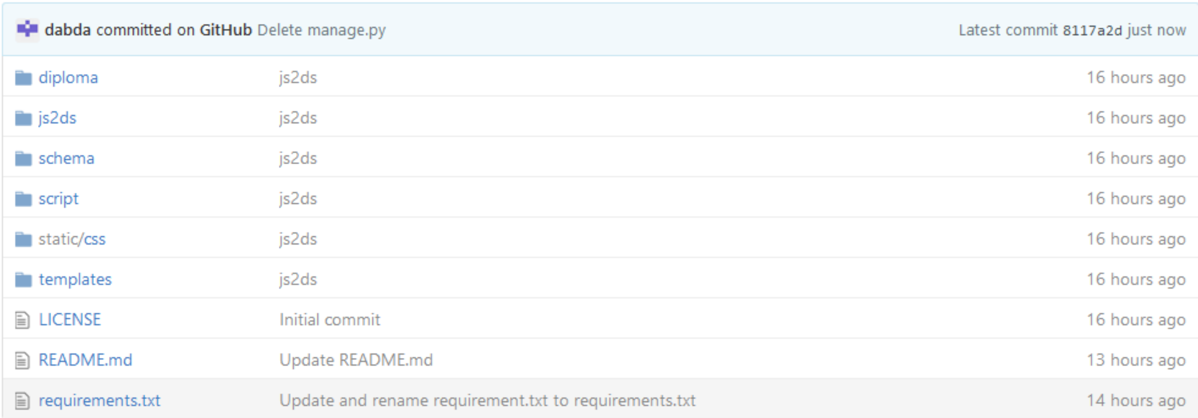
Tako kot pri tipu `Files` se tudi pri tipu `Entity` izriše tipka »skrij/prikaži«, s katero lahko pregledamo seznam že obstoječih podentitet. Ob glavnem vnosnem polju je tipka »briši«, ki izbriše vse podentitete. Pri posameznih entitetah se generirajo tipke »briši« in »urejaj«.

Algoritem med drugim skrbi za poimenovanje elementov vnosnih polj, tipk, itd. Na tak način, da dobijo unikatna imena `name`. To storimo s kombinacijami prepon in vrednosti, ki se nahajajo v ključu `"Order"`. Ob proženju tipke »submit« (spletni brskalnik izvede ukaz

`POST`) morajo imeti elementi unikatna imena, drugače jih preko metode `request.POST.get("name")` ne moremo pridobiti. Če imata elementa enako ime, se prikaže vrednost zadnjega.

3.5 Postavitev sistema

Aplikacija Django z imenom *js2ds* je dosegljiva na spletni strani GitHub (Slika 3.16) na naslovu »https://github.com/dabda/js2ds.git« pod licenco GNU GENERAL PUBLIC LICENSE, Version 3 [11].



Slika 3.16.: Aplikacija django na spletni strani GitHub

Aplikacijo Django *js2ds* lahko vključimo v svojo lastno Django aplikacijo. V datoteki README.md so zapisana vsa potrebna navodila. Datoteka requirements.txt obsega potrebna orodja za delovanje aplikacije:

- Python 3.5.1
- Django 1.9.7
- Django-bootstrap3

Verzija Python 3.5.1 se nahaja na uradni spletni strani »https://www.python.org/downloads/«. Pri namestitvi Python se namesti tudi »pip«. S pomočjo pip lahko nameščamo pakete, ki so napisani v programskem jeziku Python.

Novi projekt Django ustvarimo z ukazom `django-admin.py startproject imestrani`, ob tem smo postavljeni znotraj direktorija z imenom našega projekta. Rezultat je sledeča datotečna struktura (Slika 3.17).

```
F:\DJANGO\IMEPROJEKTA
│   manage.py
│
└───imestrani
        settings.py
        urls.py
        wsgi.py
        __init__.py
```

Slika 3.17.: Datotečna struktura novega projekta Django

V izogib morebitnim problemom z že nameščenimi verzijami paketov je priporočljivo, s pomočjo `virtualenv` ustvariti virtualno okolje, v katerem bo nameščena aplikacija Django. Virtualenv se namesti z ukazom `pip install virtualenv`. Pomaknemo se v direktorij, v katerem želimo imeti naš projekt Django ter z ukazom `virtualenv` ustvarimo virtualno okolje. Za aktivacijo okolja izvedemo ukaz `source /bin/activate`. Vsi novi novi paketi Python se bodo odslej nameščali v virtualno okolje.

Z ukazom `pip install django==1.9.7` namestimo zahtevano verzijo ogrodja Django. Če želimo najnovejšo verzijo Djanga uporabimo ukaz `pip install django`. Podobno z ukazom `pip install django-bootstrap3` naložimo bootstrap3, ki skrbi za vizualizacijo in obnašanje naše aplikacije Django. Poenostavljen postopek namestitve lahko izvedemo z ukazom `pip install -r requirements.txt`.

Js2ds aplikacijo lahko prenesemo iz GitHub strani ali pa ročno z ukazom `git clone https://github.com/dabda/js2ds.git`. V slednjem primeru je potrebno imeti nameščeno orodje git.

V datoteko `manage.py` dodamo vnosa *bootstrap3* in *js2ds* v seznam nameščenih aplikacij Django. Na ta način registriramo aplikacijo Django in ostale pakete, do katerih bomo dostopali v okviru projekta Django:

```
INSTALLED_APPS = [
    …
    'bootstrap3',
    'js2ds',
]
```

Datoteki `urls.py` dodamo naslednje vrstice:

```
urlpatterns = [
    …
    url(r'^index/$', index),
    url(r'^schema/$', schema),
    url(r'^js2ds/$', js2ds),
]
```

Na ta način se na vsaki posamezni URL naslov veže ime pogleda, ki se izvrši ob ujemanju naslova. Vsa ujemanja se izvedejo z regularnimi izrazi. V primeru `r'^index/$` gre za relativni naslov `/index/`. V primeru arbitrarne domene npr. `www.domain.name`, bi to pomenilo da ujamemo podnaslov `www.domain.name/index/`. Enako velja za relativna naslova `/schema/` in `/js2ds/`. Ob ujemanju naslova URL se izvrši klic na, ki je podana kot drugi argument URL funkcije. Te metode so podane znotraj datoteke `views.py`, ki se nahaja v direktoriju aplikacije *js2ds*.

Poglavje 4 Sklepne ugotovitve

Izdelana spletna aplikacija izpolnjuje zadane cilje. Na podlagi podanih shem se tvorijo spletni obrazci, s katerimi urejamo množico med sabo povezanih entitet. Podprto je urejanje izhodnih datotek JSON ter urejanje dodatnih datotek. Sistem lahko deluje kot samostojna aplikacija in se lahko implementira v druge aplikacije Django.

Ogrodje Django se je izkazalo kot izredno prilagodljivo orodje pri razvoju spletne aplikacije, saj v veliko primerih poenostavlja rokovanje med uporabnikom in podatki. Jezik Python kot srce ogrodja Django s svojo sintakso (*angl. syntax*), filozofijo ponovne uporabe kode in dobro dokumentacijo pohitri in poenostavi uporabo.

4.1 Razširitve za prihodnost

Razvita rešitev v okviru diplomskega dela ima prostor za izboljšave. Prirejenim shemam bi se na primer lahko dodala dodatna pravila, kot so obveznost določenih polj vnosov ali pravila za globljo gnezdenje izhodnih datotek JSON. Implementirali bi se lahko različni tipi uporabnikov z različnimi stopnjami dostopa do podatkov oz. pravicami. Ena od smiselnih nadgradenj bi bila uporaba podatkovne baze namesto datotek. Na ta način bi bile omogočene direktne poizvedbe, indeksiranje podatkov, integriteta podatkov in stopnjevanost (*angl. scalability*) itd.

Literatura

[1] N. George. *Mastering Django: Core*. GNW Independent Publishing. 2016.

[2] Michael Droettboom. *Underastanding JSON Schema*. 2015. Space Telescope Science Institute [Online]. Dosegljivo: https://spacetelescope.github.io/understanding-json-schema/.

[3] ALGator. *Dokumentacija sistema ALGator* [Online]. Dosegljivo: https://github.com/ALGatorDevel/Algator/blob/master/doc/ALGator.docx.

[4] JSON. *Wikipedia* [Online]. Dosegljivo: https://en.wikipedia.org/wiki/JSON.

[5] Introducing JSON. *json.org* [Online]. Dosegljivo: http://www.json.org/.

[6] JSON Schema. json-schema.org [Online]. Dosegljivo: http://json-schema.org/.

[7] XML. *Wikipedia* [Online]. Dosegljivo: https://en.wikipedia.org/wiki/XML.

[8] Django. *Wikipedia* [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Django_(web_framework).

[9] JSON Editor. *Repozitorij GitHub* [Online]. Dosegljivo: https://github.com/jdorn/json-editor.

[10] React-jsonschema-form. *Repozitorij GitHub* [Online]. Dosegljivo: https://mozilla-services.github.io/react-jsonschema-form/.

[11] GNU General Public Licence. (junij 2007). *GNU Operating System* [Online]. Dosegljivo: https://www.gnu.org/copyleft/gpl.html.