Web Engineering Project: WDBMS REST-API

Radu Rebeja S4051297, r.rebeja@student.rug.nl

February 4, 2022

Introduction

Project WDBMS (Web Database Management System) API's purpose is to provide a database system management software that is accessible through the web and provides all CRUD model operations. The interface is provided by a frontend layer implemented in React and using the bootstrap framework. The backend is created with Node.js (Express.js) and uses MySQL as the database management system.
In this project we provide a tool with a defined set of operations that can be applied to a dataset. Our app provides an interface to the data contained in the Netherlands Rental Properties dataset.

Naming convention: We will interchangeably refer to a database **entry** which contains fields from a Kamernet post as an **article** .

Update Version 2.0:
NRP is now a configuration script set *(and not a monolithic API)* for the database to be registered and loaded within WDBMS. The configuration includes both backend and frontend javascript scripting files.The backend scripts specify the database metadata, schema, DBMS config parameters and other attribures. The frontend scripts specify the forms to be used with the specific database. The frontend offers minimal coupling with backend. The frontend is adapted to the demands of the NRP configuration, but may also be changed to a dynamic application layer in future iterations.

Supported operations:

- Create & store new entries in the database
- Use custom filters to manipulate database entry sets
  Version 2.0: (*) all model attributes are supported for inclusion in filtering. More details offered in *Filters* sub-section 4]
- **Retrieve**, **delete** and **update** entries by applying a filter
- Retrieve statistical data (**mean , median, standard-deviation**) of : rental cost / deposit, or any other numerical attribute
- Customize number entries to be retrieved

Database schema: kamernet_db

TABLE: properties

Field	Type	Null	Key	Default	Extra
externalId	char(36)	NO	PRI	UUID4	
title	varchar(255)	YES		Empty	
postalCode	varchar(255)	YES		Empty	
city	varchar(255)	YES		Empty	
areaSqm	int	NO		0	
rent	int	NO		0	
deposit	int	NO		0	
isRoomActive	tinyint(1)	NO		1	
latitude	varchar(255)	YES		NULL	
longtitude	varchar(255)	YES		NULL	
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	

Database schema: cities_db

TABLE: cities

Field	Type	Null	Key	Default	Extra
city	varchar(255)	YES	PRI	UUID4	
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	

Lunch instructions

- Install Docker and Docker compose
- Place the *properties.json* file in */backend/local_ databases/*
- Run **docker-compose up −−build** from the root folder in your preferred terminal
- You can now access the frontend at BaseURL 3 or with *curl*

Frontend usage instructions

-Number fields in the Search Form that are doubled define bounds [min (left) , max (right)].
-Field filling is optional and empty fields are ignored.
-When executing bulk **Delete** or **Update** queries (buttons next to 'Search' button), the filtering from the filled fields found in the search form is applied.
-When clicking on the **Update** button *(bulk update)* a second form appears below. The data filled in this form will overwrite the data in the subset of entries found through filtering (from Search Form above). As in the Search Form case, empty fields will be ignored and thus not overwritten.

API Design

Version

V2.0

Base-URL

http://localhost:6868

Build description

This build was re-adjusted to allow functionality assessment on any running OS. Instead of a live frontend, we serve a static build which limits the ability to navigate or query by link (URL navigation) on browsers . Instead, the frontend is served and only accessible at the Base-URL 3 specified above from which the user navigates to the specific pages using the navigation-bar and executes the provided commands. Frontend offers searching and deleting,updating singular or multiple entries. Clicking on the entry displays the data and provides further options to update or delete the selected entry. **Data representation is offered only for .json requests. For other formats a download link will be supplied.** Another way of accessing the backend functionalities without the frontend hindrances is by using *curl*. See *Filters* 4 for examples. 3

API Request Metadata

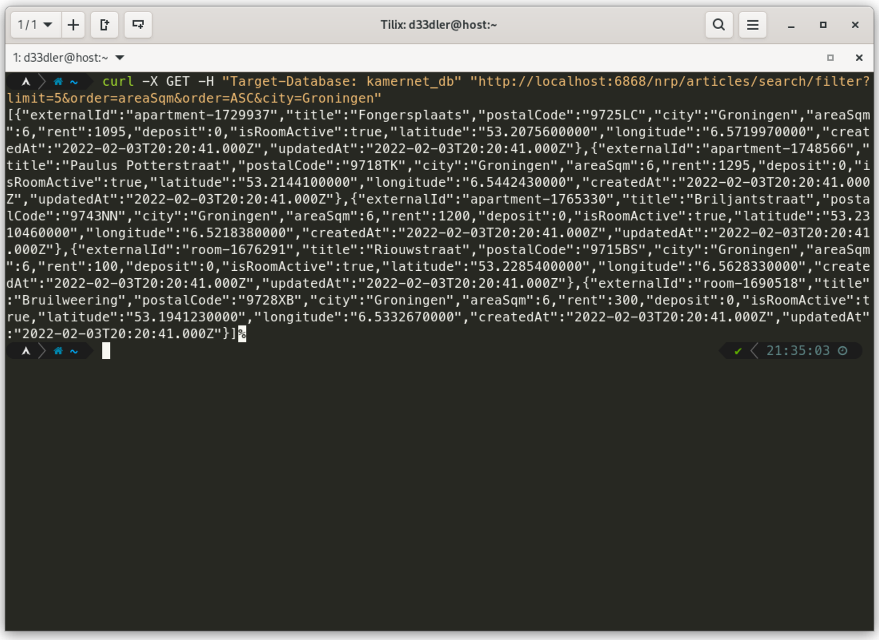
API endpoints use the following headers:

- (Default) *Content-Type* : **application\json**
- *Accept* : (**text\csv** || **application\json**)
- *Target-Database* : (**kamernet_db** || **cities_db** || *****)
- *Target-Database* : (**kamernet_db** || **cities_db** || *****)

WDBMS API will accept only *.json* format as Content-Type for the request. Target-Database is a custom header allowing specifying the ID of the database that is addressed for querying. (The value can be altered with in the frontend or specified in the Header metadata *(e.g. when using curl)*

Filters

When querying a database with GET requests for a subset of entries, we use dynamic filtering - a set of constraints that is satisfied by entries which are included in the subset. The subset's size is a constraint itself defined by the *limit* attribute.
Querying scenarios:
Frontend: With a static frontend build, querying is allowed **only** with manual form inputs. When updating/deleting, the search query fields are used for filtering
curl: With curl querying is more flexible.**Rules apply**:
-We must specify the verb (e.g. -X GET) and the mandatory header *Target-Database* and optionally the *Accept* header.
-Attributes containing typos will be ignored in the query.
-The querying parameters are assigned dynamically and the order and number does not matter. Exception for PATCH requests : minimum 1 'where' attribute must be provided
-Note! The querying includes arrays in certain cases (attributes with a set of values) and this requires knowledge of how the data is parsed on the server-side (Example: Ordering is configured by *ASC* (ascending) and *DESC* (descending) and an focus **(numerical)** parameter (e.g. : order = [areaSqm, ASC]) . In *curl* we simply write multiple instances of the same parameter with different values)). Example:



```
curl -X GET -H "Target-Database: kamernet_db" "http://localhost:6868/nrp/articles/search/filter?
limit=5&order=areaSqm&order=ASC&city=Groningen"
[{"externalId":"apartment-1729937","title":"Fongersplaats","postalCode":"9725LC","city":"Groningen","areaSqm
":6,"rent":1095,"deposit":0,"isRoomActive":true,"latitude":"53.2075600000","longitude":"6.5719970000","creat
edAt":"2022-02-03T20:20:41.000Z","updatedAt":"2022-02-03T20:20:41.000Z"},{"externalId":"apartment-1748566","
title":"Paulus Potterstraat","postalCode":"9718TK","city":"Groningen","areaSqm":6,"rent":1295,"deposit":0,"i
sRoomActive":true,"latitude":"53.2144100000","longitude":"6.5442430000","createdAt":"2022-02-03T20:20:41.000
Z","updatedAt":"2022-02-03T20:20:41.000Z"},{"externalId":"apartment-1765330","title":"Briljantstraat","posta
lCode":"9743NN","city":"Groningen","areaSqm":6,"rent":1200,"deposit":0,"isRoomActive":true,"latitude":"53.23
10460000","longitude":"6.5218380000","createdAt":"2022-02-03T20:20:41.000Z","updatedAt":"2022-02-03T20:20:41
.000Z"},{"externalId":"room-1676291","title":"Riouwstraat","postalCode":"9715BS","city":"Groningen","areaSqm
":6,"rent":100,"deposit":0,"isRoomActive":true,"latitude":"53.2285400000","longitude":"6.5628330000","create
dAt":"2022-02-03T20:20:41.000Z","updatedAt":"2022-02-03T20:20:41.000Z"},{"externalId":"room-1690518","title"
:"Bruilweering","postalCode":"9728XB","city":"Groningen","areaSqm":6,"rent":300,"deposit":0,"isRoomActive":t
rue,"latitude":"53.1941230000","longitude":"6.5332670000","createdAt":"2022-02-03T20:20:41.000Z","updatedAt"
:"2022-02-03T20:20:41.000Z"}]%
                                                                                          ✔  21:35:03
```

Third-Party API Usage

WDBMS makes use of the Geocode.Earth - API Search Endpoint, to identify property location in terms of latitude and longitude coordinates when creating a new entry based on the user's optional input data (?Street address, ?Postal code, ?City) (Endpoint using this feature : POST > /articles/new (6))

HTTP status code summary and method usage

200 - OK : Returns expected value; *[*]*
204 - ERROR : 0 Results => Using Express responses configured with this status code will have an empty body; *[GET]*
400 - Bad Request : Wrong request syntax; *[*]*
404 - Not Found : Requested resource doesn't exist; *[PUT, DELETE]*
500 - Method Error* : Error caught during method execution *[*]*

**Note:* This error indicates that the backend layer method is not handling the request correctly.

Resource usage

POST **/articles/new**

Create a new article entry in the database. Latitude and longitude will be provided by Geocode.Earth API

Parameter

Header Parameters

Accept	: response content format (json *(default)* or csv)
Content-Type	: request content format (json *(default)* or csv)
Target-Database	: request content format (json *(default)* or csv)

Body Parameters

title	: property address
postalCode	: property postalCode
city	: property city
areaSqm	: property area size in square meters
rent	: property monthly rent
deposit	: property deposit
isRoomActive	: property true if not occupied
latitude	: property location latitude
longitude	: property location longitude

Response application/json

200 ok

```
{
    "title": "Dr. Benthemstraat",
    "postalCode": "7514CL",
    "city": "Enschede",
    "areaSqm": 125,
    "rent": 995,
    "deposit": 0,
    "isRoomActive": true,
    "latitude": "52.2255530000",
    "longitude": "6.8971690000",
    "createdAt": "2021-11-29T11:29:13.000Z",
    "updatedAt": "2021-11-29T11:29:13.000Z"
}
```

500 error: method exception

```
{
    "message": "An error occurred while creating
       new rental post"
}
```

GET **/search/filter?{options}**

Search the dataset for entries that satisfy the filter constraints set by the inputs.

Parameter

Header Parameters	
Accept	: response content format (json *(default)* or csv)
Content-Type	: request content format (json *(default)* or csv)
Target-Database	: request content format (json *(default)* or csv)
Query Parameters	
externalId	: entry externalId
title	: property address
postalCode	: property postalCode
city	: property city
areaSqm_min	: lower bound property area size in square meters
areaSqm_max	: higher bound property area size in square meters
rent_min	: lower bound for attribute rent value
rent_max	: higher bound for attribute rent value
deposit_min	: lower bound for deposit attribute value
deposit_max	: higher bound for deposit attribute value
isRoomActive	: property true if not occupied
isRoomActive	: property true if not occupied
latitude	: property location latitude
longitude	: property location longitude
order	: array[<direction>,<column>] defines the ordering of the results
limit	: query result max allowed size

Response application/json

200 ok

```
{
    "externalId": "apartment-1775319",
    "title": "Dr. Benthemstraat",
    "postalCode": "7514CL",
    "city": "Enschede",
    "areaSqm": 125,
    "rent": 995,
    "deposit": 0,
    "isRoomActive": true,
    "latitude": "52.2255530000",
    "longitude": "6.8971690000",
    "createdAt": "2021-11-29T11:29:13.000Z",
    "updatedAt": "2021-11-29T11:29:13.000Z"
}
```

400 error: bad syntax

```
{   "message": "Request failed with status code 400
    & Cause: [...]"}
```

404 error: database not found

```
{   "message": "Server has responded with status
    code 404"}
```

500 error: method exception

```
{   "message": "Request failed with status code 500
    . Internal server error"}
```

GET **/city/:city?**

Search the dataset for entries that satisfy the city property constraint. To be deprecated in favor of a dynamic param search.

Parameter

Header Parameters

Accept	: response content format (json *(default)* or csv)
Content-Type	: request content format (json *(default)* or csv)
Target-Database	: request content format (json *(default)* or csv)

Query Parameters

value	: city name, the comparison will validate sub-strings

Response application/json

200 ok

```
{
    "city": "Enschede",
    "createdAt": "2021-11-29T11:29:13.000Z",
    "updatedAt": "2021-11-29T11:29:13.000Z"
}
```

400 error: bad syntax

```
{   "message": "Request failed with status code 400
     & Cause: [...]"}
```

404 error: database not found

```
{   "message": "Server has responded with status
    code 404"}
```

500 error: method exception

```
{   "message": "Request failed with status code 500
    . Internal server error"}
```

GET **/articles/statistics/{city}?{options}**

Get statistical data (entry count, mean, median, standard-deviation) for rent and deposit values for a specific city or all cities

Parameter

Header parameters

Accept	: response content format (json *(default)* or csv)
Content-Type	: request content format (json *(default)* or csv)
Target-Database	: request content format (json *(default)* or csv)

Path parameters

city	: city where the samples are used from

Query parameters

population	: number of samples
mean$_cost$	: mean value for cost attribute
mean$_deposit$	: mean value for deposit attribute
median$_cost$	: median value for cost attribute
median$_deposit$	: median value for deposit attribute
sd$_deposit$	: standard deviation value for deposit attribute
sd$_cost$	: standard deviation value for deposit attribute

Response application/json

200 ok

```
{
        "population": 5084,
        "mean_cost": "531.0334",
        "sd_cost": 276.75173793908016,
        "mean_deposit": "358.5024",
        "sd_deposit": 415.3804553576052,
        "median_cost": 410,
        "median_deposit": 320,
        "city": "Groningen"
}
```

400 error: bad syntax

```
{   "message": "Request failed with status code 400
    & Cause: [...]"}
```

404 error: database not found or Target-Database header missing

```
{   "message": "Server has responded with status
    code 404. Cause [...]" }
```

500 error: method exception

```
{   "message": "Failed to fetch database statistics
    ." }
```

PATCH **/articles/search/filter?{options}**

Update the database article(s) (that satisfy the filter constraints set by the inputs) with the provided data in the body. This endpoint may be used to update a single article by specifying only the primary key

Parameter

Header parameters

Accept	: response content format (json *(default)* or csv)
Content-Type	: request content format (json *(default)* or csv)
Target-Database	: request content format (json *(default)* or csv)

Query parameters

*	*dynamic* = may include all or some parameters specified in the {get}{/search/filter} endpoint 7

Body parameters

*	*dynamic* = may include all or some parameters specified in the database model

Body application/json

```
{
    "externalId": "apartment-1775319",
    "title": "API Doc Example Request",
    "areaSqm": "999",
    "postalCode": "2021WE",
    "rent": "0"
}
```

Response application/json

200 ok

```
{  "message": "Article was updated successfully."
   }
```

400 error: bad syntax (or 0 filter 'where' attributes inputs)

```
{   "message": "Request failed with status code 400
    & Cause: [...] / Missing 'where' attribute"}
```

404 error: database not found or Target-Database header missing

```
{   "message": "Server has responded with status
   code 404"}
```

500 error: method exception

```
{   "message": "Request failed with status code 500
   . Internal server error"}
```

DELETE **/articles/search/filter?{options}**

Delete the database article(s) (that satisfy the filter constraints set by the inputs). This endpoint may be used to delete a single article by specifying only the primary key

Parameter

Header Parameters

Accept	: response content format (json *(default)* or csv)
Content-Type	: request content format (json *(default)* or csv)
Target-Database	: request content format (json *(default)* or csv)

Query parameters

*	*dynamic* = may include all or some parameters specified in the {get}{/search/filter} endpoint 7

Response application/json

200 ok

```
{  "message": "Deletion request validated."  }
```

400 error: bad syntax

```
{   "message": "Request failed with status code 400
    & Cause: [...]"}
```

404 error: database not found

```
{  "message": "Cannot update property with id=
   apartment.Property not found or input is empty
   !""  }
```

500 method exception

```
{  "message": "Error updating property with id=
   apartment"  }
```