

Contact registry

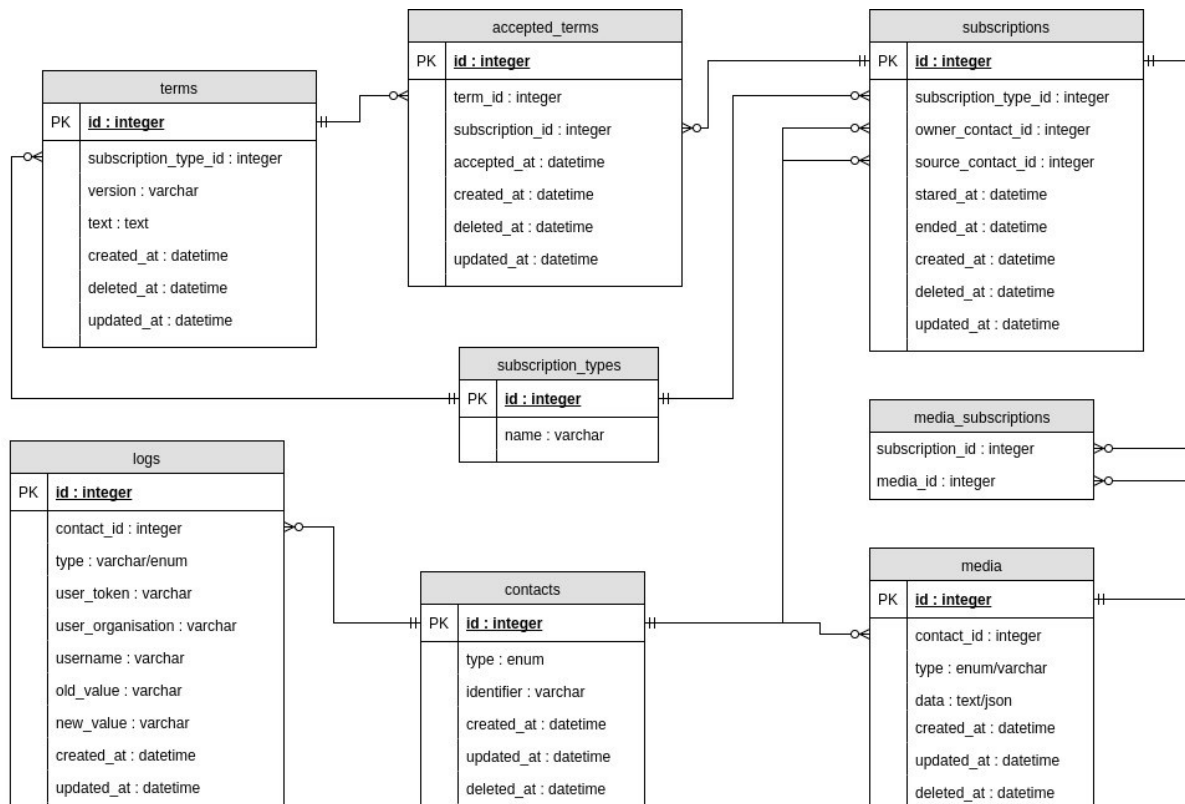
Project report

Tabel of Content

Database	3
Design	3
Tables structure	3
Software structure	5
System Design	5
Subscription Types	6
Project structure	6
Security	7
Requests	8
Documentation	10
Responses	11
Logs	11
Commands	12
Import original data	12
Limitations	12
Out of scope	13
Media confirmation	13
Send later	13
CVR/CPR integration	13
Performance	13
Testing servers	14
Results	16
Development setup	17
Docker	17
Automatic testing	18
Coverage	18
Deployment requirements	19
User stories	19
Data analysis	20
Recommendations	20
Importing data sets	20
Contacts table	20
Contact address data table	20
Appendix	22
Database table data sample	22

Database

Design



Tables structure

The date fields `created_at`, `updated_at`, and `deleted_at` are all technical fields introduced by the Laravel framework used for this solution and mostly serve as a kind of audit trail for the data itself. The application will interpret all records with a `deleted_at` as not existing in the database at all, which referred to as “soft deleted” or nondestructive deletes. Soft deleted field could prove useful in the scenario where a customer is reported dead and thus “removed,” but a few days later “resurrected” due to the initial notification being an error.

It will allow for easy cleanup of old data, i.e. deleting records permanently after a grace period. The `created_at` and `update_at` fields will be useful in the scenario where the need to extract delta changes from the database.

All field names that end with `_id` refers to a primary (internal) id of another table.

Contacts

The table contacts contain the different types of contacts both private persons and business. In the future perhaps also authorities. The identifier field will currently include either CPR or CVR but in the future when other identifiers emerge i.e. the future replacement for the CPR or if authorities get their identifier.

Contact logs

This table is the audit log for everything that happens related to a contact (`contact_id`). Logging will allow seeing the context in which the contact's data is looked up, deleted, created or changed, and what changes if any (`old_value`, `new_value`) become part of the request's log including who (`user_token`) made it. The UI should never see the user token instead `user_organization` and `username` will be used to show a human-friendly representation of those who can access the contact.

Subscription types

A subscription type represents notifications, reminders and digital post (adviseringer, NemSMS and Digital postkasse), but can include other subscription types in the future due to having these defined in the database.

Subscriptions

To associate a subscription type (`subscription_type_id`) to a contact (`owner_contact_id`) subscriptions are needed. Usually, the source and owner will be related to the same contact but in the scenario where a niece needs access to notifications (or other subscription types) on behalf of another contact the source (`source_contact_id`) will point to another contact.

Media

The different forms to contact someone is described in media. A media has a `type` that will represent either a digital mailbox, email, mobile phone. In the future, this could be any contact form like physical addresses, smartphone device ids, Messenger, Dropbox, Google Drive, etc. The actual endpoint (i.e. email or phone number) will be stored in the `data` field which due to the `text/JSON` data type allows for complex data structures. Having media separate from the subscriptions allows for having multiple medias attached to a subscription. It also allows for removing an email that bounces or adding a new media to an existing subscription without mixing in terms (since they are already accepted).

Terms

The terms table describe the terms that need to be accepted before allowing a contact to be contacted for certain types of subscription types like notifications, reminders and whatever the future brings. The `version` and `text` fields are used to add some metadata to the terms and this table could be extended to include more fields as necessary.

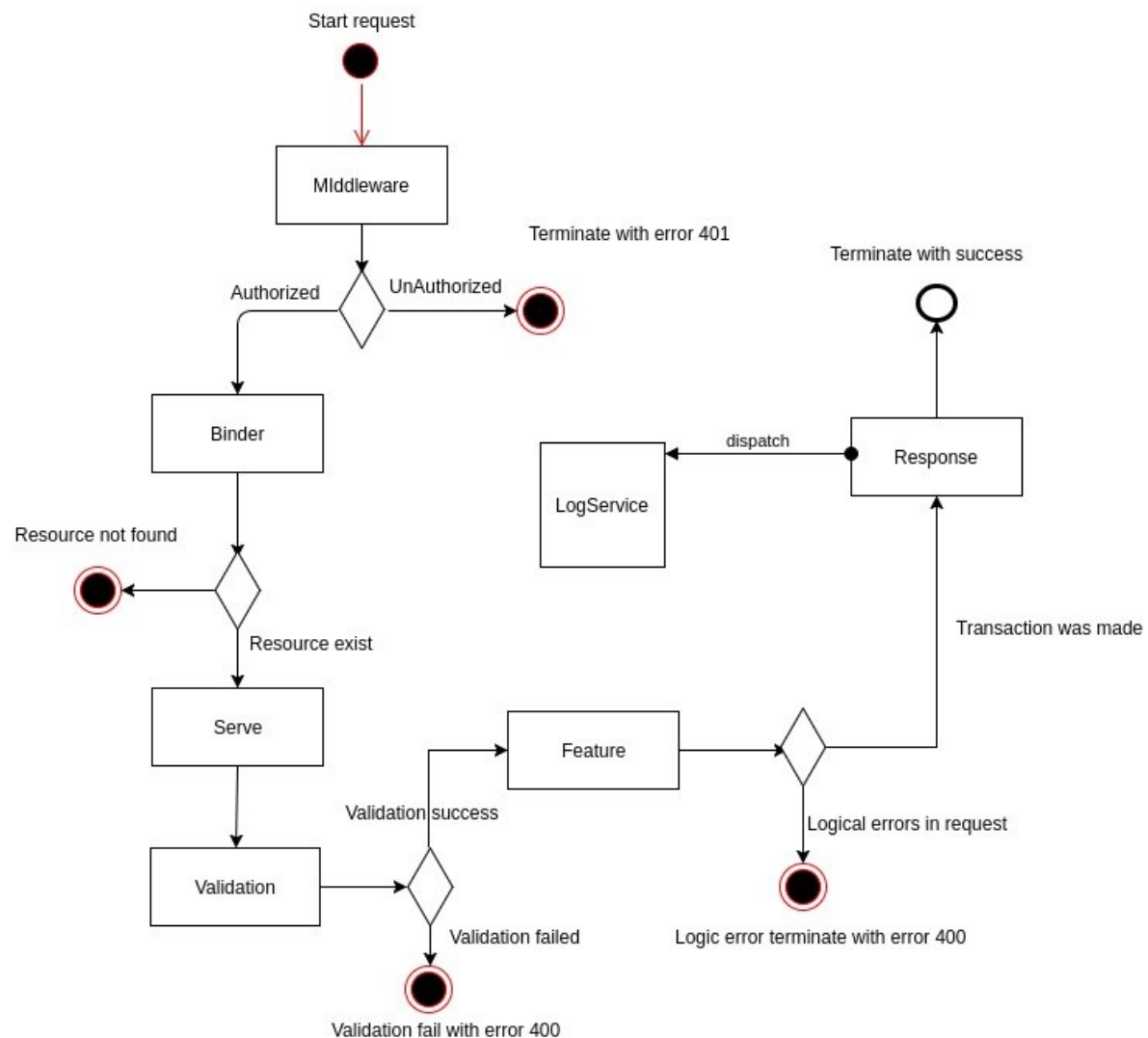
Accepted terms

Whenever a contact accepts a specific version of some terms, a record of this action will be created. Having acceptance separate from the subscription itself allows for nondestructive updates when it comes to accepting new versions of the terms, in the sense that all relations between subscriptions and media remain unchanged.

Software structure

The application is build using MCV combined with a layer object oriented design for the API design.

System Design



Subscription Types

Contact registry supports two media types by default, *SMS* and *email* media types, but with the flexibility to add more without limitations, the application evaluates media types.

When adding new media types, the new media type support needs to add as an array to `Api\Validation\MediaValidationRule::class` to ensure that any data that recorded contain the expected format and sanitized.

To add new media types to the validation need to follow laravel validator rules¹. To add new media types to the validation need to follow laravel validator rules, defining the media key will predefine the new media type and the value sets fields and validation rules.

```
'imap' => [  
    'server' => 'string|required',  
    'key' => 'string|required'  
];
```

Once the new media type is defined, any API consumer can create new media with the provided type.

Project structure

API directory

Related classes, services, and models to the API lives in this directory.

- **Contracts:** Any contract (interface) that should be implemented by models or classes to fulfill the requirements for other services in the application.
- **Controllers:** Any controller placed here must extend BaseController and implement Controller Interface, to fulfill API's requirements, to have access to the Serve method to call their respective feature and trigger validation and any other layer in the application.
- **Features:** This directory contains features representing actions to for their model. By convention, each feature should be inside a unique class and perform only one action. It also returns the model modified model or collection of models that will be transformed using the transformer property required in by the application.
- **Maps:** Mapping is used to indicate which resource should be used to validate request parameters or inputs.
- **Resources:** Resources must extend BaseResource and define the validation rules that apply to each possible input attributes in given endpoint. This validator array can also include and validate other resources, enums, and filters.

¹ <https://laravel.com/docs/5.4/validation#available-validation-rules>

- **Models:** Models represent domain-specific data, logic, and relations with other models.
 - **Transformers:** Transform models data for public consumption.
 - **Support:** Provides helper classes to the application to be used features or libraries.
 - **Validations:** Extends the validation rules to include more complex validation rules that could use the resources.
 - **Exceptions** that will be utilized only by the API, the exceptions extends from APIException to handle error messages in the application renderer.

App directory

The application folder has Laravel framework² standard structure with modifications to support Digist application requirements.

- **Exceptions** handler has been modified to support API responses and handle different environments.
- **Jobs** directory contains any job that can be triggered by to be the queue in the application to avoid overload of the system with processes that may take a long time to process.
- **Services** same functionality as services directory in API directory, but with a global scope to be used in API and app itself for private transactions.

Security

Basic auth

Basic authentication transaction allows a simple system to system connection and authentication method to the registry, using API tokens to identify requester and manage its permissions to determinate the access level to the data.

NemID Integration

Contact registry process request based nemID authentication adding support for municipality administrators to manage their information and write, update or delete information stored in the registry.

² <https://laravel.com/docs/5.4/structure#the-app-directory>

Requests

Request protocols

The application by default supports the following methods, and these methods will include the correct status code depending on the request made.

Type	Success code	Error codes
POST	201	400, 401, 403, 404, 500
GET	200	400, 401, 403, 404, 500
PUT/PATCH	200	400, 401, 403, 404, 500
DELETE	204	400, 401, 403, 404, 500

Status code definitions

- **Code 200:** OK
- **Code 201:** Created
- **Code 400:** Bad request
- **Code 403:** Forbidden
- **Code 404:** Not found
- **Code 401:** Unauthorized
- **Code 500:** Internal server error³

RestFul web service

POC system is a web service application that follows guidelines and recommendations on how to build RESTful⁴ application and services from Google⁵.

This web service is a stateless system that processes a request and generates a response based on this without ever saving any authentication locally. Therefore authentication headers need to be provided in each call done to the service.

Accepted methods

³ <http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml>

⁴ https://en.wikipedia.org/wiki/Representational_state_transfer

⁵ https://cloud.google.com/apis/design/naming_convention

Method	Resource	URL	Response
GET	*	/search	List<Contact>
	Search functionality provides a broad range functionality to meet contact registry requirements, for example: <ul style="list-style-type: none"> • Finding contacts in the registry • Filtering by subscription types • Filter by upcoming events like end date for subscription <i>notice:</i> multiple values need to be separated by (,) and information need to be trimmed and clean(0-9) only.		
GET	Contact	/contacts/{identifier}	Contact*
	<i>Get a more detailed source overview and relations.</i>		
POST	Contact	/contacts	Contact*
	<i>Create a new contact resource, and return newly created contact.</i>		
POST	Contact	/register	Contact*
	<i>Create a new contact resource with media and subscription.</i>		
GET	Media	/contacts/{identifier}/logs	List<Logs>
	<i>Get contacts log list.</i>		
GET	Media	/contacts/{identifier}/medias	List<Media>
	<i>Get contact media list</i>		
POST	Media	/contacts/{identifier}/medias	Media*
	<i>Create a new media resource for a contact. and return newly created media.</i>		
DELETE	Media	/contacts/{identifier}/medias/{media_id}	
	<i>Delete media, returns empty response on success.</i>		
PUT	Media	/contacts/{identifier}/medias/{media_id}	
	<i>Update media, returns updated media.</i>		
GET	Subscription	/contacts/{identifier}/subscriptions	List<Subscription>

	<i>Get contact subscription pagination ready list.⁶</i>		
POST	Subscription	/contacts/{identifier}/subscriptions	Subscription*
	<i>Create a new subscription for a contact and return newly created subscription.</i>		
DELETE	Subscription	/contacts/{identifier}/subscriptions/{subscription_id}	
	<i>Delete subscription.</i>		
GET	Subscription	/subscription-types	List<Subscription Types>
	<i>List of available terms for ordered by subscription type.</i>		
GET	Term	/terms	Contact
	<i>Create a contact including media and subscriptions.</i>		
POST	Media-Subscription	/contacts/{contact}/media-subscriptions	Media-Subscription
	<i>Create contact media subscription.</i>		
DELETE	Media-Subscription	/contacts/{contact}/media-subscriptions/{media_subscription}	
	<i>Delete contact media subscription.</i>		
GET	Term	/term/{term_id}	Term
	<i>Get single term.</i>		

* Include all relations for this resources

The application also generates documentation from models and features docs block, and this can access it with no security check within the application environment using basic authentication.

Documentation

The application does contain a json file with the basic documentation to display basic request to the api, but the user will need to have valid NEM-ID credentials to be able to

⁶ Pagination ready list includes next page, previous, current count of objects in request and total number of objects for given request.

access it, this do not prevent the user from looking at the sample data, but only prevent from making request to the api.

The documentation can be found under <http://localhost/api/documentation> while running the docker container, or their equivalent in any running server. But the application needs to load <http:// URL /api-docs.json> to be able to display the right mock data.

Responses

Hide fields

The system provides with a mechanism to hide any sensitive data, at the moment this functionality protects mobile and phone number, this information is hidden or shown based on the nemID and token's permissions.

Response Type

Responses are JSON objects, and there are two types of responses, one for any index request such as search and other that single objects. Index responses are a list of objects with the options, criteria and failed parameters included in the response object, and individual objects are the resource itself is returned as a JSON object.

Failed responses

Failed responses return JSON objects with a status, message and it does also include response status code in the headers, the application provides the option to return trace of errors when while in debug mode.

XML support

The POC at the moment do not have support for XML support, but this can be extended by adding all the required templates to the application and accept headers that will dictate what kind of response the requester is expecting.

Logs

To address the scalability issues that generating logs entry could create the application creates the logs using a queue system running in a different container that can be scale up and down depending on the application needs.

Every request that is processed by the application is queued to stored into the contact's log. Saving the unique identifier, type of request, old value and new value when present on the request.

Each log record is saved in a generic form. For example, when an update happens to a contact's media information the application will record the method (PUT), a named field

old_value containing number:old_number and a field new_value containing number:new_value and the unique identifier for the actor making the change to the media.

Commands

- cleanup is a command that is programmed to run each day at 2:00 to remove any contact deleted after five years and remove any relation that this contact can have in the contact registry.

Import original data

There is a command to read and import the original data provided by Eboks, this assumes data for bruger-adresser contain all the fields in the same table.

To be able to import the data the following steps:

1. Place data in storage folder
2. Make sure the data has been clean to remove special chars
3. run `php artisan db:reset; php artisan ds:import-file --contacts=storage/min/bruger_min.txt --addresses=storage/min/brugeradresser_min.txt --fritagelse=storage/min/fritagelse_min.txt`
4. All parameters are required

The application on a non server environment runs at ~6625 entries per minute.

Limitations

The **search** function contains a notable underperformance compared to the rest of the application. Having a higher latency and response time, this lack of performance is because of the need to look into a lot of records and calculate results for not found matches. The performance can be optimized by using technology such as elasticsearch. Elasticsearch will provide high performance and speed for the search functionality with minimal changes to rest of the application core functionality.

Subscriptions should include a validation system that prevents wrong media types to be attached to the subscription, at the moment medias that are emails can be added to NemSMS subscriptions. One possible solution could be to predefine which media types can be attached to a subscription, and this will facilitate the validation inside the create subscription feature.

Out of scope

Media confirmation

Implementation of media confirmation is one of the key features that this application is missing at the moment since the application is a stateless system this raises the issue of how media data verification process would work before being saved into the registry, to allow to maintain a clean database without corrupted or invalid information.

Therefore an extension to this application and handle the validation and verification of contact's data such as emails and SMS numbers or any other kind of media data that could apply in the future.

Send later

Also known as “Eftersendelse” and possible not something the contact registry should ever handle.

The contact registry should just end a subscription and whoever de-registers a specific subscription have the responsibility to send unsent documents by other means to avoid giving the responsibility to the contact register to store an address for one-time use.

CVR/CPR integration

CPR integration is missing for the lack of information how this process will work, but the implementation is a simple process once the following point gets cleared:

- CPR notifies registry or registry request from CPR?
- How often?

CVR integration does also need the same clarification as of the CPR, and adds the following:

- Which company status is considered to be closed and will register the company as permanently close and except digital post?
- Do this five years also apply for businesses or is only for citizens?

Performance

Tests are done under virtual environment restraining their performance. Therefore, the purpose this stress test is to provide a starting point or minimum performance requirements for any feature application development. And create an idea of how its performance can be in a real production environment, but these test environments do not take into consideration

any other possible scenarios that could make the application performance slow: firewalls, load balancers, server specifications, etc.

Testing servers

Both machines are running Docker containers, and SSD disks.

Machine one*

CPU@cores	2x-Xeon@2.1
RAM	612MB
Workers	4
Threat per core	1

Machine two*

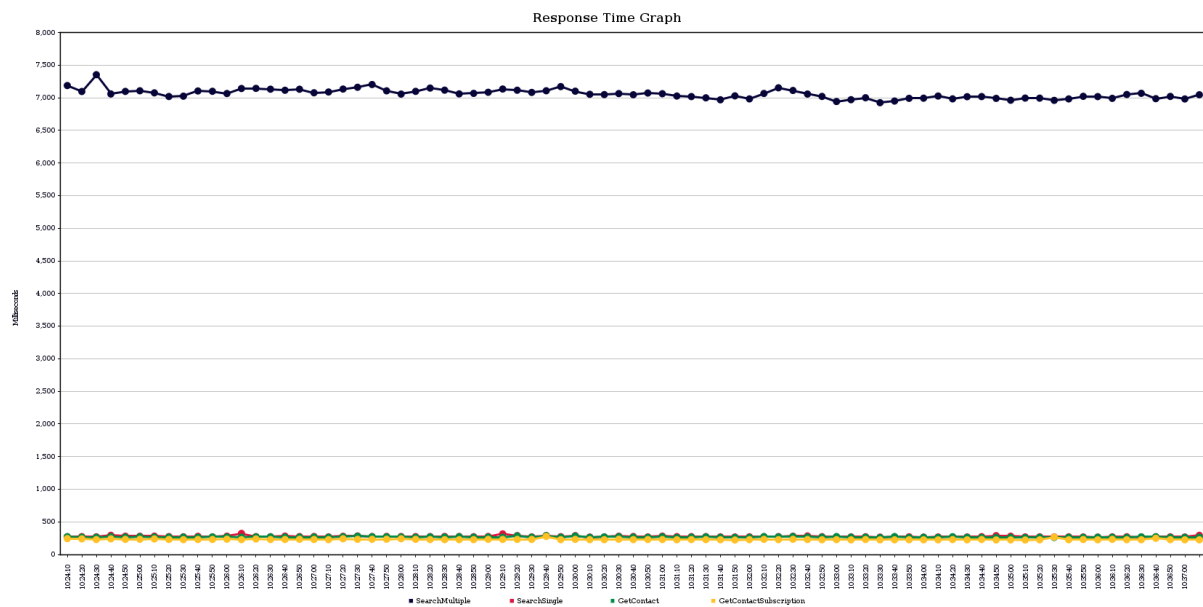
CPU@cores	i7-7600U @ 2.80GHz × 4
RAM	~15GB
Workers	4
Threat per core	2

* *Runs in virtual environments, it may underperformance compared to a dedicated server.*

Load samples

Functionality	Number Users	Request count
Search Multiple (a 39 identifiers batch)	2	100
Search single	2	100
Get Contact	2	100
Get Contact's subscription	2	100

Machine one results



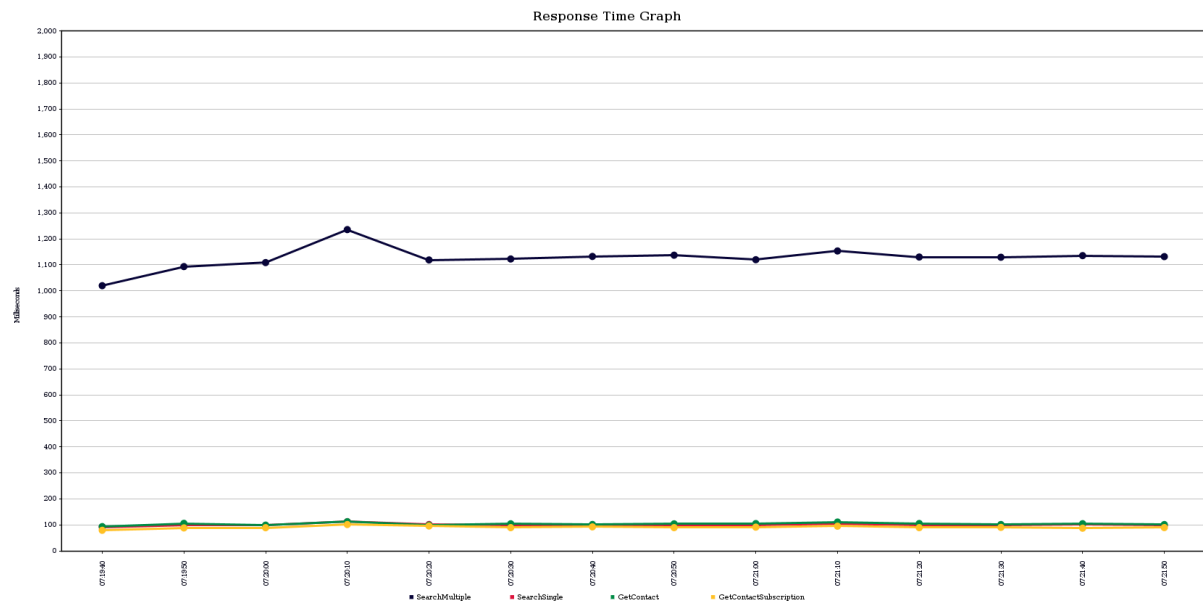
	Average	Median	Min	Max	Samples
Multiple Search	7059	7049	6894	7505	200
Single Search	276	272	264	376	200
Contact	267	266	258	312	200
C. Subscription	230	229	222	350	200

All results are in milliseconds.

Samples: amount of request in order to calculate values.

[HD Graphic](#)

Machine two results



	Average	Median	Min	Max	Samples
Multiple Search	1125	1122	1001	1220	200
Single Search	99	98	77	170	200
Contact	103	101	81	168	200
C. Subscription	90	88	67	141	200

All results are in milliseconds.

Samples: amount of request in order to calculate values.

[HD Graphic](#)

Results

The application performs okay considering the environment on which is running, but as mention before the search function needs to be implemented using elasticsearch or similar to provide the performance, speed, and scalability required by an application of this size.

Development setup

The application uses Laravel for its core functionality, laravel's documentation is enough to get started extending on top of this system, and any API specific requirements and documentation are explained in this document.

Docker

The host environment is required to have docker and docker-compose installed ([documentation](#)).

Requirements

- Docker 17.03.1-ce or greater
- Docker compose 1.13.0 or greater

Next steps

- Make sure the you are in the root folder of the application.
- Run the following commands:
 - `docker-compose up -d --build`
 - This command will build the application using the required containers and network initial setup.
 - By default this will create 3 containers that are connected via two internal networks.
 - `docker-compose exec app bash`
 - This command will grant access to the application terminal, once inside the container.
 - `copy('.env.example', '.env')`
 - `composer install`
 - `php artisan cache:clear`
 - `php artisan migrate`
 - `php artisan db:seed`
- In some cases there it may be needed to fix the folder permissions since the docker container can access it:
 - `chown -R www-data storage`
 - `chown -R www-data bootstrap/cache`
 - `chmod -R 0770 storage`
 - `php artisan cache:clear`
 - `composer dump-autoload`
- Finally visit the in the browser [localhost](#)

Automatic testing











All testing is done via PHPUnit⁷ using SQLite drivers for database transactions. The folder structure follows the same logic as the rest of the application dividing tests into features or unit depending on the type of test type or propose.

The application container installs PHPUnit and SQLite to facilitate local testing, but the project also includes CircleCI⁸ file with the right configuration to provide a continuous integration for testing and deployment to regular hosting services like Amazon services or Digital Ocean droplets.

- **Feature** contains all tests for the API endpoints including security, exceptions, validation and information access; these tests provide a way to verify that the result is what the user expects back, or the right exceptions are thrown when the validation fails. It also makes possible to test all scenarios described in the method. Each feature by default uses NemID payload with admin rights except on those test where test if the user can edit, delete or create their information in the contact registry.
- **Unit** test complexity on models or any other class that provides functionality help to the application methods.

Coverage

The application has almost 100% test coverage, testing each end point, exceptions and helper methods allowing to make sure that each feature can be changed or modified and warranty that each endpoint will always return the same result it did before the changes.

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	<div><div></div></div>	96.03%	460 / 479	<div><div></div></div>	90.14%	128 / 142	<div><div></div></div>	82.76%	48 / 58
 Controllers	<div><div></div></div>	95.71%	67 / 70	<div><div></div></div>	96.15%	25 / 26	<div><div></div></div>	83.33%	5 / 6
 Exceptions	<div><div></div></div>	100.00%	6 / 6	<div><div></div></div>	100.00%	3 / 3	<div><div></div></div>	100.00%	3 / 3
 Facades	<div><div></div></div>	100.00%	1 / 1	<div><div></div></div>	100.00%	1 / 1	<div><div></div></div>	100.00%	1 / 1
 Features	<div><div></div></div>	99.04%	207 / 209	<div><div></div></div>	95.56%	43 / 45	<div><div></div></div>	91.67%	22 / 24
 Maps		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0
 Models	<div><div></div></div>	83.72%	36 / 43	<div><div></div></div>	78.12%	25 / 32	<div><div></div></div>	50.00%	4 / 8
 Resources		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0
 Support	<div><div></div></div>	93.00%	93 / 100	<div><div></div></div>	81.82%	18 / 22	<div><div></div></div>	40.00%	2 / 5
 Transformers	<div><div></div></div>	100.00%	49 / 49	<div><div></div></div>	100.00%	12 / 12	<div><div></div></div>	100.00%	10 / 10
 Validation	<div><div></div></div>	100.00%	1 / 1	<div><div></div></div>	100.00%	1 / 1	<div><div></div></div>	100.00%	1 / 1

⁷ <https://phpunit.de/>

⁸ <https://circleci.com/>

Deployment requirements

Scripts to run deployments may change depending on the services that are being used to deploy the application and the server destination. But the following commands need to be executed and are required by the application to release new versions of the system.

- `php artisan down`
 - make the application enter maintenance mode, to prevent any user from experience loss of data.
- `php artisan composer-install --no-interaction`
 - install any new dependency or change to the required packages and libraries
- `php artisan migrate --force`
 - Add any changes to the database
- `php artisan optimize`
 - Optimize autoloading class and scripts to a production state
- `php artisan up`
 - Bring the application live again.

User stories

- **US1:** Citizens or company's subscriber can provide and register mobile and/or email to be able to receive digital post.
 - Email addresses are validated by known standards.
 - Phone number saved.
 - Contact can subscribe and unsubscribe at any given time.
- **US2:** Contact is able to edit contact information
 - Contacts are able to edit media
- **US3:** Except from digital post
 - Contacts with an exception will be omitted from the search result providing a way for authorities to know that this person can't receive notifications.
- **US4:** Search functionality allows authorities to check by CPR/CVR numbers if any contact exist and it has valid media subscriptions.
- **US5:** Read advisory information
 - Search functionally supports bulk requests of CPR/CVR numbers allowing to get requested information with fewer requests to digital post.
- **US6:** Search functionality provides a set of filters that can determinate if the CPR/CVR would have any event in the near future like subscription ending.

- **US7:** Each contact can access every request that has been made to their registry, changes and person making the changes identified by nemID unique request code.

Data analysis

Several issues were found in the data provided, the most persistent errors are the following.

Recommendations

- Remove all xFFFD chars in strings
- Clean data for white spaces
- Make sure each string does only contain number and letters
- The usage of uni2ascii to prevent any chart lost⁹

These commands are need based on the dataset received, but the optimal way to minimize the amount of data loss is to compare with another data set provided in another format that is not CVS, so there is more support for odds string formats and make possible to create better scripts to sanitize data.

Importing data sets

Contacts table

There are over 116k issues during clean up, and import of the data, the most common problems are an odd formatting or corrupted data present in the data dump when strings contain special characters that break the input.

Contact address data table

Category definitions:

- 1 - Email
- 2 - Mobile number
- 3 - Secure E-mail

Email issues (1)

There is --+480 broken email addresses, this is the result from some phone numbers being placed in the wrong address type, for example:

Type	Address
------	---------

⁹ Sample commands can be found under app repository cleanup.sh

1	42638001
1	JENSEN @ MAIL.DK
1	bent-larsen @ domain.dk
1	lars landmand @ gmail.com
1	-----
1	kristen.lars.com
1	dig@lxxsen.com ¹⁰
1	"gunnar larsen\" <erik.larsen@domain.dk>"

Mobile issues (2)

There is a total of +-10473 emails that are registered as mobile numbers:

Type	Address
2	kontakt@domainxxx.dk
2	pro-bit @subdomain.digstxxx.com ¹¹

Secure E-mail (3)

There is only 2 issues in this category with e-post.

Type	Address
------	---------

¹⁰ Full log file data_contact_addr_email_cat.csv

¹¹ Full log file contact_address_data_mobile_cat.csv

3	-
3	-

Appendix

Database table data sample

Subscription types

id	name
1	notification
2	reminder
3	digital_post

Contacts

id	type	identifier	created_at	updated_at	deleted_at
1	person	0307987289	2017-05-23 08:49:34	2017-05-23 08:49:34	null
2	company	10342309	2017-05-23 08:49:34	2017-05-23 08:49:34	null

Media

This needs to be explain where the 1 and 2 in the contact_id is coming from.

id	type	data	contact_id	created_at	updated_at	deleted_at
1	email	{"email": "rosenbaum.edyth@jacobs.net"}	1	2017-05-23 08:49:34	2017-05-23 08:49:34	null
2	sms	{"number": "10064888"}	2	2017-05-23 08:49:34	2017-05-23 08:49:34	null
3	digital_post_box	{"box": "40164943"}	1	2017-05-23 08:49:34	2017-05-23 08:49:34	null

Subscriptions

id	subscription_type_id	owner_contact_id	source_contact_id	started_at	ended_at	created_at	updated_at	deleted_at
1	1	1	1	2017-05-23	null	2017-05-23 08:49:34	2017-05-23 08:49:34	null
2	1	1	2	2017-05-23	null	2017-05-23 08:49:34	2017-05-23 08:49:34	null
3	3	2	2	2017-02-07	null	2017-02-01 11:49:34	2017-02-01 11:49:34	null

Media subscriptions

id	subscription_id	media_id
1	1	1
2	2	2
3	3	4

Terms

id	subscription_type_id	text	version	created_at	updated_at	deleted_at
1	1	Occaecati qui ut	0.1	2017-05-23 08:49:34	2017-05-23 08:49:34	null
2	2	Occaecati qui ut	0.2	2017-05-23 08:49:34	2017-05-23 08:49:34	null
3	2	Occaecati qui ut	0.3	2017-05-23 08:49:34	2017-05-23 08:49:34	null

Accepted terms

id	subscription_id	term_id	accepted_at	created_at	updated_at	deleted_at
1	1	1	2017-05-23 08:49:34	2017-05-23 08:49:34	2017-05-23 08:49:34	null
2	2	2	2017-05-23 08:49:34	2017-05-23 08:49:34	2017-05-23 08:49:34	null