Core Design Decisions

I choose Dropwizard[1] - a Java framework for developing RESTful web services – as the basis for this project. Dropwizard is framework with the goal to provide a reasonable interconnection of several established java libraries. Therefore, MS1 currently uses:

- Jetty for HTTP serving.
- Jersey for REST modeling.
- Jackson for JSON parsing and generating'.
- Logback for logging.

MS1 provides authentication services through a RESTful API serving json web tokens. JWT generation is implemented the jose.4.j library[2]. In the coming weeks user account creation and authorization services will be implemented.

API Endpoints:

Currently:

/AUTHENTICATE/	supporting GET-requests using HTTP Basic Authentication[3] and if successful serving a session token for the authenticated user

Coming:

/AUTHENTICATE/	supporting GET-requests with a pre-shared-key encrypted JWT containing login information and if successful serving a session token for the authenticated user
/USER/	supporting PUT and POST-requests to create new user accounts
/AUTHORIZE/	supporting PUT, POST, DEL to manage a user's privileges

JWTs will be checked using a pre-shared HMAC-Key. This means during login each user receives a session token. This token will be send with each user request to any of the microservices part of the ToDo-List-App. Validity of the tokens can be checked at the receiving service without contacting MS1 again. This ensues there is no direct communication between microservices without being orchestrated by MS3. In practice this solution leads to improved scalability of the microservice architecture.

Currently the API supports HTTP and HTTPS as well as the generation of unencrypted and fully encrypted session tokens.

[1] http://www.dropwizard.io/1.2.0/docs/
[2] https://bitbucket.org/b_c/jose4j/wiki/Home
[3] RFC 2617

Deployment

See README.md for details on building, configuring and starting MS1

The service can be deployed as a fat jar with jetty embedded. During startup the application has to be provided with a config.yaml file:

```
logging:
  # The default level of all loggers. Can be OFF, ERROR, WARN,
INFO, DEBUG, TRACE, or ALL.
  level: INFO
  loggers:
    # Overrides the level of above and sets it to DEBUG.
    at.ac.univie.dse: DEBUG

#Server configuration.
server:
    applicationConnectors:
        - type: http
          port: 8080
        - type: https
          port: 8085
          keyStorePath: ms1/dse.keystore
          keyStorePassword: dse2017
          validateCerts: false

dse2017:
  secret: gyswgQL+9ts5xc1XX47Y0xxI7ReNuh/w
  # can be jws for unencrypted jwt's with signature or
  # can be jwe for fully encrypted tokens
  tokenType: jws
```

1 - Example of configuration file: config.yml

Ideally, I would like to offer the option to deploy it as a Docker Container. Not only should this be a convenient way to start the application, but also provide an easy way to test the microservice architecture on a single machine.

```
INFO  [2017-11-16 22:05:07,665] org.eclipse.jetty.server.handler.ContextHandler: Started
i.d.j.MutableServletContextHandler@1fb2eec{/,null,AVAILABLE}
INFO  [2017-11-16 22:05:07,676] org.eclipse.jetty.server.AbstractConnector: Started
application@70d63e05{HTTP/1.1,[http/1.1]}{0.0.0.0:8080}
INFO  [2017-11-16 22:05:07,680] org.eclipse.jetty.server.AbstractConnector: Started application@e11ecfa{SSL,[ssl,
http/1.1]}{0.0.0.0:8085}
INFO  [2017-11-16 22:05:07,681] org.eclipse.jetty.server.AbstractConnector: Started
admin@485e13d7{HTTP/1.1,[http/1.1]}{0.0.0.0:8081}
INFO  [2017-11-16 22:05:07,681] org.eclipse.jetty.server.Server: Started @3210ms
DEBUG [2017-11-16 22:05:16,467] at.ac.univie.dse.auth.BasicAuthenticator: successfully authenticated user: Testuser
DEBUG [2017-11-16 22:05:16,472] at.ac.univie.dse.auth.TokenHandler: building token of type jws
0:0:0:0:0:0:0:1 - - [16/Nov/2017:22:05:16 +0000] "GET /authentication HTTP/1.1" 200 224 "-" "insomnia/5.10.1" 165
```

2 - Example of the console output generated by MS1

Testing and Presentation

For testing and demonstrating the functionality of MS1 the Insomnia REST client was used[4]. A file containing supports RESTrequests, which can be imported into insomnia can be found at:

ms1/src/test/Insomnia_requests.json

Instructions on using MS1 via cURL can be found in README.md

Tokens received as response can be validated via www.jwt.io (see p.4)

```
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS change cipher, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-SHA256
* ALPN, server did not agree to a protocol
* Server certificate:
*  subject: CN=localhost
*  start date: Nov  6 18:15:47 2017 GMT
*  expire date: Feb  4 18:15:47 2018 GMT
*  issuer: CN=localhost
*  SSL certificate verify result: self signed certificate (18), continuing anyway.
* Server auth using Basic with user 'Testuser'
> GET /authentication HTTP/1.1
> Host: localhost:8085
> Authorization: Basic VGVzdHVzZXI6dGVzdHB3
> User-Agent: insomnia/5.10.1
> Accept: */*
> Accept-Encoding: deflate, gzip

< HTTP/1.1 200 OK
< Date: Thu, 16 Nov 2017 22:05:16 GMT
< Content-Type: application/json
< Vary: Accept-Encoding
< Content-Length: 224
```

3 -Example of a Request Timeline generated by Insomnia

Current State of Implementation

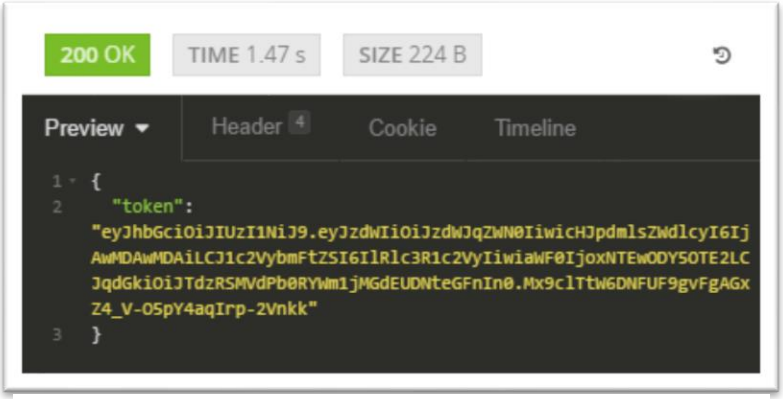
MS1 currently exposes one API Endpoint using http basic authentication (/auth/BasicAuthenticator.java). An Authenticator class accepting JWTs already exists but isn't used at the moment.

Token generation is currently done in a single class file (/auth/TokenHandler.java) which decides on generating encrypted or unencrypted tokens via a simple If-statement. Ideally this would be managed via a factory pattern.

The management of user accounts is currently performed by using a BiMap<Integer, User> object in an implementation of the DAO. Ideally this will be done by using Hibernate and the H2 database. Both supported by the application framework dropwizard.

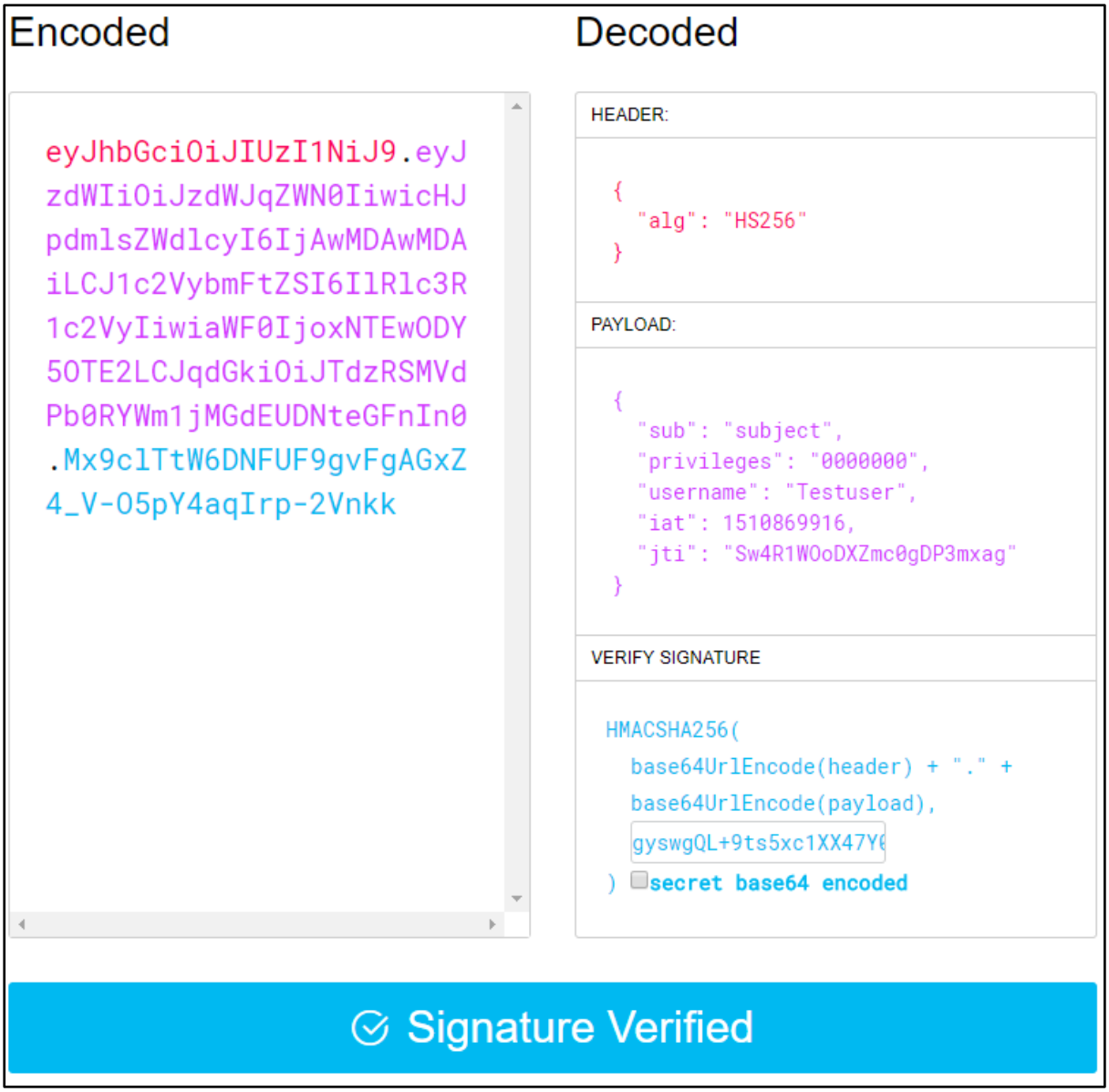
Missing API endpoints as specified on p.1 must be added under /resources/ to provide user management and authorization services.

Available project specific configuration options (ms1Configuration.java) for MS1 in config.yml currently are the pre-shared Key for token encryption or signing and specifying if MS1 should serve encrypted or unencrypted JWTs. Ideally this will be expanded as needed.



4 - Example Service Response as shown by Insomnia

[4] https://insomnia.rest/



5 - Looking at a JWT generated by MS1 via www.jwt.io If the token should be encrypted the PAYLOAD-field will show random characters.