Report Second Assignment
Healthbook - B. Secondary concepts

Deadline: 03.05.2018 18:00 via TUWEL

Group 13

Name	**Matriculation#**
GLAS Josef	08606876
MAITZ Alexander	01426069
MUSIC Ismar	01328053
~~PUHALO Stefan~~	~~01228354~~

B. Secondary concepts

B.1 REST versioning

The RESTful APIs [1] of our web application will change over time. Therefore we will introduce following versioning concept.

1/ How can clients request older versions?
Basically we thought about 4 Options:
1/ URI versioning
2/ Versioning through query parameters
3/ Versioning through custom headers → version entire API
4/ Versioning through content negotation → version single resource (Media type versioning)
1+2 affect URI

All approaches have pros and cons. For our project we would go for option 1.
Cons:
- larger URI footprint
- http caching
Pros:
- clear, consistent and stable API contract with clients, any change of our API contract with clients will be reflected by changed URIs
- no need to vary http header
- no versioning of media type

DETAILS:
We provide RESTful APIs following the general scheme
`scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]`
query string = `key1=value1&key2=value2 ...`

The URI is as follows: https://healthbook.at/api/<version>/<resource-name>
The most current version can be access with following permalink:
https://healthbook.at/api/<resource-name>

Example: If API v3.0 is the latest version, in addition to the permalink following aliases wiill be supported (behave identically):
https://healthbook.at/api/v3.0/<resource-name>
https://healthbook.at/api/v3/<resource-name>

In addition, API clients that still try to point to an old but supported API can do so by assessing corresponding URIs like:
https://healthbook.at/api/v2/<resource-name>
https://healthbook.at/api/v1.1/<resource-name>
https://healthbook.at/api/v1/<resource-name>

---

[1] REST (n.d.). In Wikipedia. Retrieved April 23, 2018
Web service APIs that adhere to the REST architectural constraints are called RESTful APIs. HTTP-based RESTful APIs are defined with the following aspects:
- base URL, such as http://api.example.com/resources
- an internet media type that defines state transition data elements (e.g., Atom, microformats, application/vnd.collection+json, etc.) The current representation tells the client how to compose requests for transitions to all the next available application states. This could be as simple as a URL or as complex as a Java applet.
- standard HTTP methods (e.g., OPTIONS, GET, PUT, POST, and DELETE)

2/ How does our architecture supports the simultaneous operation of multiple versions?
URI routing to point to a specific version of the API

3/ For which cases the support of outdated interface versions won't work?
In case of major semantic changes, eg changes requiring a new media type,

B.2 DSGVO – Protection of personal data (“the regulation”)

As of 25.5.2018 the “Datenschutzgrundverordnung” (DSGVO) will become effective in all EU member states. This regulation (EU 2016/679) is focusing “on the protection of natural persons with regard to the processing of personal data and on the free movement of such data...”
The regulation is applicable to our project and as “controller” and “processor” of personal data are required to comply.
Although we have considered many basic requirements in this respect (covering Art 6 “Lawfulness of processing” including encryption or pseudonymisation) many aspects have not been addressed properly so far. The following gives a short non-exhaustive overview.

1. Any new functions will be required to assure DSGVO compliance?
Right to data portability
It should be easy to transmit personal data between service providers. Further analysis needed in this respect.
Right to erasure (“right to be forgotten”) (Art. 17)
When an individual no longer wants their data processed and there is no legitimate reason to keep it, the data needs be deleted.
Right to withdraw (Art. 7, Art. 16)
Immediate amendments, extensions or even withdrawals shall be supported.
Data breach notification feature
Procedure needs to be completed within 72 hours after occurrence is experienced.
see also below

2. Is there any need to adapt the existing functionality?
Renewal of authorization
Although we share (process) data only after approval (“consent”) of the user (“data subject”), it is required to update regularly the authorization (“Zustimmungserklärung”)

3. Considering the fact that the application will be hosted in a cloud (Azure, Amazon Cloud Services, …), is there any impact on the operational process of the application?

Assuming we will offer our services in EU territory, we are obliged to comply with the regulation. This is also applicable to our providers (“Auftragsdatenverarbeiter”) which need to comply with the regulation as well. This is among others crucial when negotiating the Service Level Agreement. Outsourcing is addressed in Art 28 and Art 29.
Duties of processors are described in Art 27 para. 1, Art 30 para. 2, Art 31, Art 32 para. 1, Art 27 para. 1.
Chapter V (Art 44 to Art 50) describe the limitations for cross-border data transfer outside EU.

One current example:
Data breach notification duty
The provider (“Auftragsdatenverarbeiter”) needs to comply with data breach notification duty:
The user has the right to know when their personal data has been hacked, therefore we have to inform individuals promptly of serious data breaches. They will also have to notify the relevant data protection supervisory authority.

4. Is there any need to adapt the internal organization of our enterprise “Healthbook” with regard to the operational process?

Beside a clear documentation (logging) of all activities, it is crucial to o assure the regulation is implemented and monitored properly, therefore the designation of a “data protection officer” is obligatory.

Data compliance officer

This is required assuming we are “processing data on large scale”. Art. 37 and Art. 38 describe the position and the tasks of the officer, among others:

- inform and advise the controller or the processor
- provide advice where requested as regards the data protection impact assessment and monitor its performance
- cooperate with the supervisory authority and act as contact point

B.3 Ongoing data migration

The database schema for our application will change frequently. For example the schema will be extended due to additional functionality (data requirements).
Assuming the web application is already deployed, how can these database changes be applied? Changes shall be applied automatically during live operations without any data loss.

As we are using MongoDB, a schema-less database, we would foresee a versioning system for the documents. This allows to have documents of both versions at the same time. An application should support reading documents in all versions and update/write only the latest version.

There are various opportunities which would need further investigation but Vermongo would be our favorite.

Vermongo - simple versioning scheme for keeping older revision of MongoDB documents
https://github.com/thiloplanz/v7files/wiki/Vermongo

- a version number property is added to the document
- documents updated one-by-one (no bulk updates)
- update only possible with version number (similar to ho subversion works)
- outdated versions goes to shadow collection
- update performance slower

As an alternative approach we could evaluate a more conventional migration tool like *migrate-mongo* which also fit very well with our technology stack or *Mongobee*.

However, both could be a challenge from performance point of view and probably require a maintenance window for migrations (downtime). On the other side we could avoid version control and rely only one single version. Or maybe we decide to go for a mixed approach, for major change we use migrate-mongo, for minor changes we rely on version control.

migrate-mongo - a database migration tool for MongoDB in Node
https://www.npmjs.com/package/migrate-mongo

- create new migration script (in migration directory)
- edit content with instructions to change databases
- up/down implementation can use either callback-style or return a Promise
- migrate-mongo up (applies all pending migrations)
- migrate-mongo down (reverts only the last applied migration)

Mongobee – java tool which helps to manage changes in MongoDB and synchronize with application. Similar to Liquibase or Flyway but without using XML/JSON/YML files
https://github.com/mongobee/mongobee

- create ChangeLog containing multiple ChangeSets
- ChangeLog is a class annotated with @ChangeLog and containing methods annotated with @ChangeSet