IMSE Guidelines SS2022

(Updated: 02/2022)

General remarks

- **Common sense is the golden rule!** If this isn't working for you - ask in the forum for help!

- If you have questions that are not strictly related to a personal matter of yours **use the forum**.

- Check the kick off slides to get the full picture of the requirements.

- You can write either in German or in English.

- Documentation must be uploaded as a single PDF. Only the source code (and if necessary images, but no binaries or libraries should be included) for the implementation is to be uploaded as a single ZIP file.

- If a deadline is missed, points will be taken (per milestone) as explained below.

  ```
  Within the 1. 24 hours 1 point,
  Within the 2. 24 hours additional 2 points (total of 3),
  Within the 3. 24 hours additional 3 points (total of 6),
  Within the 4. 24 hours additional 4 points (total of 10),
  ...
  ```
  you get the gist. Submissions uploaded later than 7 days after the deadline will not be considered anymore.

Milestone 1

- **1.1.2: Entity-Relationship model (Chen notation)**
  When describing your application domain, be precise about how things are expected to work. Make sure that the things described in the application domain and/or use cases are actually working with your model and verify that your model meets the requirements (number of entities, relationship types, …) as defined in the kickoff slides on the course page.

- **1.2. Use-Case Design**
  Each team member has to design 2 use cases, of which one must be **clearly indicated** as **a main use case of your business** (meaning this is the use case you are going to implement). Please make sure that each main use case **creates some sort of data** (meaning it inserts/updates into the database). Also, indicate **which team member was responsible** for which use cases.

  Examples: A library could have two of the following use cases as its main use cases "renting a book", "returning a book", "register a new book", "register a new user", "add book to user favourites", … and whatever you can think of that creates some sort of data.

  - **Detailed textual description**
    A use case description has to include at least the following elements: objective, short description, precondition(s) (e.g. successful execution of other use cases or a specific system state), expected execution (set of activities), postcondition(s) in case of success and error. Verify that each mentioned data element is actually present in the model (i.e. do not refer to information that is not part of 1.1.2).

  - **Graphical representation of dynamics**
    Illustrate your use cases using only "Activity diagrams" (see https://en.wikipedia.org/wiki/Activity_diagram) or "BPMN diagrams" (see http://www.bpmn.org/). Also, check the BPMN Cheat Sheet downloadable on the course page.

    Concepts that are likely to be relevant for your assignment:
    - *Swim lanes*
    - *Message/Control flow*
    - *Start-/Endpoints*
    - *Decision vs Activity (especially the number of outgoing connections)*
    - *Parallels and Joins, or how and when to join control flows in general*

Maybe your use cases require only a subset of the concepts I mentioned, or maybe it requires a concept that's not listed above, this depends on what and how you're modelling, but for most applications this should be sufficient.

Hint: *When designing your diagrams, use swimlanes to identify where you may need some sort of API and also don't forget that* ***data is read and written*** *from/to the database.*

- **1.3. Reporting**

  Imagine your company starts doing some data analysis to gather useful insights into how your company's business is evolving. Each team member has to define one report/data analysis use case. Please make sure that they …
  - … include **at least 3 different entities** (remember that joining/bridging tables resulting from m:n relations are no entities!).
  - … define a **field to be sorted** on.
  - … define a **field to be filtered** on.
  - … include **data written** by your **main use cases**. In other words, the data presented by the report should change when the according main use case is executed.

  For these reports/ data analysis use cases, you do not need to make a graphical representation nor a lengthy textual description. Instead list the included entities and their included attributes. For example:

  Find active users**?** A report about all users who rented a book within the last year, including the most recently rented book (title, author and ISBN) of each user and sorted by its last name.
  - Use entities: Books, User, Author
  - Filtered by: rental date (last year)
  - Sorted by: Number of rentals in the last year
  - Columns: "User name", "User address", "User age", "Title of most recent rental", "Author name of most recent rental", "Number of rentals in the last year"

Milestone 2

2.1. The RDBMS Part (Phase 1)

- **2.1.1. Configuration of Infrastructure**
  Ideally, all **compile work (if any) is done inside a container** during build-time together with pulling in all the dependencies, allowing your build-process to be completely agnostic about its host. Make further sure that your docker-compose file does not expose unnecessary ports and that each container is completely self-contained (i.e. does not depend on its host environment).

  For HTTPS: **self-signed certificates are fine**.

- **2.1.3. Data import**
  The DBFilling script generates all the data needed for proper execution of your main use cases as well as the reporting. Implement a button in your GUI that starts the filling process of the SQL database (and only the SQL database!). Delete all data first and fill it with (mostly) randomised data again.

- **2.1.4. Implementation of a Web system**
  Every team member needs to implement her/his **main case** and the according **report**. The **aesthetics of your GUI** implementation will be considered when grading the assignment. For example, GUIs that depend heavily on the usage of ID/PK fields will lead to a deduction of points.

  You **do not need to implement a fully featured login system**, but it is a requirement to be able to choose which user should be used (given there are any users in your system).

2.2. NoSQL Design (Phase 2)

Upload a PDF describing the following aspects of your Milestone 2 submission.

- **Logical/Physical database design of your RDBMS**

- **NoSQL design (collections, documents, ...)**
  Write down the **reasons for designing the NoSQL collection/documents** the way they are. *E.g. one could argue that rentals as subdocuments of users increase performance of the user's* `rental history query` *by saving a join. Since the rental history of a user is requested very frequently this is the most beneficial design option. On the contrary it is less performant for querying the rental history of a book as it would require the complete user collection to be iterated for this report. But since the rental history of a book is de facto immutable, and we do expect updates on a books rental history happens only every few days (average rental duration) we decided to*

have the rental history redundant inside the book document too. Thus we can query the user rental history equally easy as the rental history of a specific book.

Hint: *Including example documents of each collection makes it easier to highlight how the data is structured/referenced in the model.*

- **NoSQL indexing**
  Propose **how data should be indexed** in MongoDB to be **beneficial for your application**. *E.g. Since in the reporting use case, only users who rented a book within the last year should be considered, a time-to-live index on* `user.rentals.start_date` *is beneficiary to identify rentals within the last year, and since it is a TTL, it will not grow infinitely.*

- **Show and compare the SELECT statements of both main and elaborate use cases from to MongoDBquery statements**
  Include the SQL query statement of each **report** in the documentation. *E.g. for reporting:* `SELECT users.name, books.title, rentals.from, rentals.to FROM users, books, rentals …`
  Note: that your reporting use cases (**and only the reporting use cases**) are supposed to be expressed as a single statement each.

  Include how you implemented the **aggregation pipeline for the reporting use case** in using the syntax from MongoShell. Keep in mind that for efficiency reasons (which are considered during grading) it is recommended to filter (e.g. *$match, $project*, …) first and aggregate (i.e. *$lookup, $aggregate*, …) only relevant data afterwards.

NoSQL Implementation (Phase 3)

- **3.1. NoSQL Database design**
  NoSQL collections must be designed to support the same functionality as the RDBMS schema (i.e. no semantic information is to be lost). Implementing only entities relevant for the main use cases is considered incomplete and will result in deduction of points.

- **3.2. Data migration**
  The **data migration** is not a data recreation(!) nor simultaneous write operations into both databases. Add a button in the GUI to start the data migration. (Note: it is OK if the NoSQL data is wiped before data is migrated). Also, going back and forth between databases is not required.

  Hint*: Check if all foreign- and primary keys defined in the RDBMS are still relevant in the NoSQL design, and if so, explain why.*

Final Presentation

General Remarks

- Have your project and presentation prepared on your laptop
- Both parts should be presented in not more than **20 minutes!**
- Upload a PDF version of your slides (there is a dedicated upload service on the course page)
- A presentation is **mandatory** for a positive grade

Implementation

Have your stack (with empty DB) up and running **before** we start the timeslot.

1. Fill DB
2. Every team member presents her/his use case and report/data analysis
3. Migrate data from RDBMS to MongoDB
4. Every team member presents her/his use case and report/data analysis

Presentation

Basically "*summarise your report*"

- Compare your **RDBMS model** with your **NoSQL design** (collections and documents)
- Performance implications of the chosen NoSQL design for the reporting use cases (indexing, embedding, …)
- Show and compare the SQL Select statements to the MongoDB query statements.