Project NBD (20 points) – Assessment Rules

1. On the following pages, there are 4 projects to choose from, each requiring the use of a different database. You select and complete a single project.
2. The "Requirements" for the projects are formulated very minimally and leave a lot of room for interpretation – this is 100% intentional; I want to see what creative things you can do with them.
3. If you have a cool idea for a project of similar size to those presented – we can negotiate (but only in class, not remotely 😉).
4. The assessment criteria are below; I reserve the right to deduct points in the case of malicious compliance or other practices I do not like. I have no problem with using AI tools in building the project, but you should know what you did, build a sensible solution, etc.
5. Yes, some projects may be a bit harder than others – choose wisely.

Assessment Criteria:

- **Technical correctness and functional completeness (6 points)**
    - Does the application meet the minimum project requirements?
    - Implementation of the full set of required functions (CRUD, queries, logic specific to the given database).
    - Correct operation — no critical errors, the application runs and produces repeatable results.
    - Data consistency and predictable application behavior.
- **Proper use of NoSQL database (5 points)**
    - Was the chosen database used according to best practices?
    - Were the appropriate database features utilized?
- **Code and architecture quality (4 points)**
    - Is the code readable, logically divided, and do modules have clear responsibilities?
    - Is the API or user interface designed consistently?
    - Does the project follow good practices (e.g., error handling, data validation, separation of logic from the database layer)?
- **Documentation and runnability, quality of demonstration (5 points)**
    - README contains launch instructions (steps, dependencies, environment variables).
    - Description of data/index/query structure.

- Brief description of the project architecture.
- Sample test data or a database seeding script, quality and volume of data.

Allowed Programming Languages

- Java
- C#
- Python
- Ruby
- Rust
- JavaScript

If you want to use another language – contact me for permission

MongoDB — "Task Planner / To-Do List with Flexible Categories"

Project Description

A simple application for task management:

- Adding, editing, deleting tasks
- Tasks can have different fields depending on the type (e.g., deadline, priority, project)
- Searching and filtering tasks by category, priority, or date
- Simple aggregations: number of tasks in a project, average priority, etc.
- The application should be a web app; you can build an SPA, but server-side page generation is also acceptable

Suggestions:

- A tasks collection in MongoDB, flexible documents (different fields depending on task type)
- Aggregation pipelines (group, match, sort) for statistics

Redis — "User Session and Cart Management"

Project Description

A simple online store demo that allows:

- Creating and managing user sessions
- After logging in, a session with a unique identifier (token) is created
- The session automatically expires after a specified time (TTL)
- Ability to check if a session is active
- Real-time cart management
- Each user has a cart assigned to their session
- Ability to add, remove, and view products in the cart
- The cart expires with the user session
- The application should be a web app; you can build an SPA, but server-side page generation is also acceptable

Suggestions:

- Hash for storing cart information (product_id → quantity)
- String for session (session_id → user_id)
- TTL for automatic session and cart expiration
- Handling error cases (e.g., non-existent session, expired session)
- Use of cryptography (bcrypt?) is not required but appreciated

Neo4j — "Product Recommender"

Project Description

A simple tool for product recommendations based on friends' ratings, allowing:

- Registering users
- Adding products
- Adding "knows"/"follow" relationships
- Adding "recommends"/"discourages" or "rates" relationships (with scores in the latter case)
- Visualization of user connections
- Visualization of user recommendations
- Generating product recommendations based on graph queries (e.g., products recommended by at least X friends, products liked by people with similar recommendations to mine...)
- Implemented as a web or desktop application

Suggestions:

- Model with nodes of type *User* and *Product* and relationships like *FOLLOWS* / *KNOWS* / *RECOMMENDS* / ...
- The application does **not** have to allow user-mode operation – I even prefer a version where you can see and edit everything

Elasticsearch — "Restaurant Search Engine*"

** – you can easily change this to a search engine for something else*

Project Description

A server-side API plus a sample application (can be web or desktop) allowing:

- Adding restaurant descriptions (with some classification like offer, price category, size info, address, etc.)
- Adding user reviews
- Performing full-text searches (match, fuzzy, phrase)
- Filtering results (by categories and similar)
- Displaying result rankings

Suggestions:

- Create an index with appropriate mapping
- Load sample documents
- Queries like match, match_phrase, fuzzy, bool
- Endpoint for search with pagination and result sorting
- Optionally: highlights/promoted offers