Possible steps from Customer's idea to a working proven and accepted system

- A Database, data management and Back-end point of view v. 2022-09-23 by JV
- 1. Customer has the idea
- 2. Requirements engineering (Interviews, questionnaires, workshops, old system views and outputs, ...)
- 3. Database design
- 4. (ER diagram and Data dictionary, iff you want, limited real life model)
- 5. **DB diagram**, database model trying to emulate the limited real life model with a DB
- 6. Testing the logic of the database from the database diagram, e.g. if there is a loop, two ways to go from A to B, then there must be two different kind of connections between A and B. Same thing cannot be said twice.
- 7. **Data dictionary**, explains concepts, **common language**/used terminology, and also details, like data types and value domains. E.g. Are we using feet or meters for the flight altitude?
- 8. Choice of PK:s? Do we use 1.real life data keys like PIN or surrogate keys: 2. auto-generated id:s. (Or 3. GUID:s?)
- 9. And Alternate keys (AK, UNIQUE) and CHECKs/Triggers, all based on business rules and "design decisions".
- 10. Foreign key policies (DCA, UNA, etc...) based on the business rules: Deleting Order, what to do with OrderRows?
- 11. Testing the model by asking questions: "Who are the project managers for projects that have open positions for workers?" and checking if we can find the answer.
- 12. Normalization rules used to test that the database is, at least, in the 3NF. Happens often automatically for profs.
- 13. Creating the database with Git maintained SQL DDL scripts (or the more modern code-first approach, which has the benefit of writing the model schema only once, but also the drawback of cumbersome data insertion).
- 14. Testing the database with few lines in each table to check the table structure and foreign key logic.
- 15. Testing the database with initial **technical test data 5-10-20-10**, written as version management maintained SQL DML scripts. Test data always needs to have variation: Categories without Ideas, categories with 1, with many.
- 16. Concurrency design? E.g. row version verification to ensure nobody has modified same data in the meantime.
- 17. Testing the database with the most important business case originated SQL queries, again in version management and shared/updated
- 18. Create a SSH tunnel, to provide both secure pipe and database server name not specified in the case
- 19. Create the backend project (e.g. Node.js, Express, Knex, mysql 'driver'). All DB etc. settings defined only once!
- 20. Define the REST API service routing with Express.
- 21. How to do income request data validation, including possibly also search criteria get:s?, with joi?
- 22. Implement asynchronous HTTP Request handler with Express
- 23. Implement the database operations that use Knex's chained database query builder functions asynchronously like: .orderBy('lastName').orderBy('firstName).then(... & other functions e.g. for outer joins and NULL handling.
- 24. Knex operation (input and) return values. Read also from model project code. Important topic:
 - a. insert (req has: JSON excluding the id) => id(s) of newly created row(s)=object(s) returned as JSON array
 - b. select (req has: e.g. id or AK) => JSON object based on select/join (e.g. Member, Idea with its Category)
 - c. delete (req has: e.g. id or AK) => number of rows affected (=deleted)
 - d. update (req has: JSON including the id) => number of rows affected (=1 = 1 updated)
- 25. In case of an error we will get an error object containing a message, but more importantly the SQL "error code". 0 = no probs. Basically anything else is a problem of some kind. Should we send specific error objects to frontend or just generic HTTP status codes? To give more info on UI: yes. To be more secure and tell hackers less: no.
- 26. Test the REST API service with PostMan
- 27. Create the frontend (Redux and AJAX temporarily handling part(s) of the DB model in Frontend!) and test it
- 28. Leads into need for more backend services...
- 29. There are three different basic situations from the programming point of view. Important topic:
 - a. standalone table that has an id like Category Id generated if new. No foreign key saved to table
 - b. table with an id and foreign key(s) to others, like Idea Id generated if new. Selected foreign key saved
 - c. table without an id, linkage table, PK is a composite key of mainly foreign keys to other tables, like **Comment by a Member** for an Idea The foreign keys given in the request, will be saved to the
- 30. Possibly transactional features, where one user action causes many lines to be updated in the database?
- 31. While developing the final UX, we need usability test data 5-500-1000-200 to test also paginating and searches.
- 32. When time to test and tweak performance, we might need performance test data 5-100 000 2 000 000-5 000