What to check where in full-stack architecture?

1. Frontend 2. Backend 3. Database (DBMS)

Usability – Performance – Security

Characteristics of Frontend

- **Web browser** on a mobile device or PC. Or a **native application** ().
- We could have X 000 - X 000 000 users, thus counting all devices **a lot of processing power**
  - Of course with bad programming it's always possible to ruin performance
- Extremely **fast and smooth interaction** with the user is possible
  - Versatile UX components, drag-and-drop, …
- User is slow though (second(s)) compared to computers. And user can e.g. take coffee break for 20 mins
- (Frontend device might be off the network too)

What to do in Frontend?

- Input validation (e.g. illegal characters, number and date formats, min-max) (USABILITY AND PERFORMANCE)
  - Always better do error-prevention than error-handling (e.g offer user only the allowed options) (USABILITY)
- Offer login screens, handle role-based navigation differently for public surfer / logged in user / logged in admin so that that certain features only offered for some users. But, this is only for (USABILITY AND PERFORMANCE), not a security check!
- Not sending any requests to backend if frontend can detect any problem! (PERFORMANCE)
- Security credentials/tokens etc. sent to Backend (implementing the Frontend part of security checking)

- Request forging might by-pass our Frontend checks totally!!!
  - => do not trust any checks done in Frontend. They still have to be done for USABILITY and PERFORMANCE.

Frontend – **Internet** – Backend

- Protecting the data with encryption (**HTTPS**, certificates, …)
- Man-in-the-middle? Wifi eavesdropping? Session hijacks?

Characteristics of Backend

- **REST API backend** in case of Full-stack
- Easy to put many instances of the very same backend code/binaries running parallel in the cloud (with a load balancer)
  - => **power not so much a problem** as in with databases
  - Though often we only have one instance of backend (then could be performance bottleneck)

What to do in Backend?

- Always re-check everything what Frontend (supposedly Frontend, but could be also black-hat hacker) has sent! Data validation, security checks, authorization.
  - SECURITY and PERFORMANCE
- Don't start any database operations if backend could detect the problem before! (PERFORMANCE AND SECURITY)
- Service-level authorization (and also authentication with tokens, as we want to be stateless).
- In case of error what to return to the caller? (Supposedly our Frontend, but could be black-hat hacker)
  - Informative option: all info about why failed: e.g. 441 - "Field 'name' is missing". Maybe good in development phase
  - Secure option: just "200=ok" or "400=prob" without revealing any internal structures. And for unauthorized or broken requests just: DROP, which means stop handling the request, no response to the client. Good when testing over and deployment to the Internet has happened.

- **Do not trust** the service calls and data that seem to come from the frontend!
- Protect Database by checking **whatever you can** in the Backend services! Hide the database address and username(s)!

Backend – **Internet** – Database (DBMS)

- Protecting the data with encryption (HTTPS, certificates, …)
- Man-in-the-middle?
- By-passing backend and attacking database directly?
- We use SSH tunnel here for encryption/protection

Characteristics of Database

- **Usually there is one data model**, that has to keep its integrity no matter what
- => We **usually have only 1 database instance**, which is a performance bottle-neck
  - It is possible to use multiple DB instances, with so called distributed databases. But they come with a lot of costs (performance costs, as well as money).
- Database is to be bothered only after all other means of checking have been passed!

What to do in Database?

- Checks of many kind. Basically still re-checking everything, although here usually pre-defined in the DDL definitions.
- PKs, UNIQUEs, NOT NULLs, FKs, CHECK conditions, TRIGGERs
- Logical checks that could not be checked fully without the whole data model
- Object- and user-based (sometimes also role-based) authorization usually needs to be checked (again) using database
- RDBMSs are also really fast in searching, sorting, filtering, combining, calculating, … data
- RDBMSs are really good in resolving concurrency problems
- Just think that
  - each DB operation has to be as valid as possible
  - Each DB operation has to be wrapped in one transaction, possibly prepared, then transaction started, run fast, committed or rolled back

- **RDBMS is really powerful. But do not jam it with problems that can be detected without database.**

Front end, front-end, or frontend?

English language evolves in phases. Officially we might be on the second phase, front-end.

But I am lazy to change in 2025 so I sometimes write frontend ☺

No, actually in some technical names frontend is just better.