Starting the Frontend (UX) design and the Backend REST API web service design simultaneously

What does it mean: Designing e.g. the most business value bringing **view (a)** first, along the **backend REST API web service**(s) **(b)** it needs, and also either starting, updating, or validating the data storage or **database design (c)**.

Why to do that simultaneously on this course? Why not just frontend and user experience (UX) design first?

- Our approach is technical correctness and good (technical) practices. Architecture as a whole enlightens us on why to do things a certain way
- For easier learning, as you'll need to think about the whole architecture once, before you start creating other views. While doing a) or b) first without the other, you'll most likely do a lot of changes later
- To keep a) b) and c) in synch. Remember that usually some other team members are reading the code and documentation in the same time. And often different parts share the same services or same fractions or subparts of the database.

Process:

- Take one view having the biggest business value (or in many cases: the easiest one to implement and fastest to test/build the new architecture)
- Design it fast as wireframe (with real tools like InVision, or just on paper, or with square boxes in Paint or Excel or something)
- Think what web services it will need to get its data and to do its updates. **Describe those Web services** (see later)
- Check / Update / Expand the database to support the functionality and content.
- Implement the design for that one view with the chosen technology, e.g. as React components
- Complete other parts later, like AJAX to backend, backend service, database queries…

How to describe/define web services (creating a document telling how to implement them later):

General about the service:

- What does it do? For what use, for what not.

Input (mainly in the request):

  - URL
  - (((mapped to internal service name / function name)))
  - Http verb: GET/POST (DELETE/UPDATE/PUT)
  - Request parameters (mainly for GET), POST form data, (((cookie text))), request header data
  - (Protocol, like http, https)

Output to the response:

  - Http Status code (inside the response headers)

- 200 OK, 2XX OK with additional info
- 3XX redirects
- 400 request error, 404 resource not found, 409 conflict=e.g. trying to insert user with existing email (AK), 4XX request somehow wrong or cannot be served because of the request.
- 500 generic server-side (backend) error, 504 upstream connection prob / gateway error

- response body (data), e.g. JSON:
  - the object(s) frontend needs to show/edit
  - sometimes in error case, info about which fields were missing/incorrect
- other response headers
- (of course the JSON data might have URLs pointing at more resources, like images even on a DIFFERENT server)