Dette er en gruppeopgave. Du skal først oprette en gruppe.

Dit svar IKKE AFLEVERET

Afleveringsopgave 2 filer

Personal Test Data Generator

Design and implement unit, integration and end-to-end tests for an application that generates fake personal data for application tests. It must be developed in groups of 5 or 6 students. Please create the groups and write your names in the Excel sheet Groups – First Mandatory Assignment.xlsx in Microsoft Teams.

System under test

The application generates fake information for non-existing Danish persons with the following architecture:

- A backend that serves the requested information via an API
- A graphic UI that consumes the API

The backend contains either a series of functions or a class with a series of public methods that return:

- *Name* and *gender*. Compound information:
    - *First name*
    - *Last name*
    - *Gender*
    - Randomly extracted from the provided file person-names.json
- *CPR*. 10 numeric digits
    - The first six ones identify the date of birth in the format ddMMyy
    - The last four digits are randomly generated, with the following constraint:
        - For female persons, the final number is even
        - For male persons, the final number is odd
- *Date of birth*. Must match the date in the CPR

- *Address*. Compound information:
    - *Street*. A random assortment of alphabetic characters
    - *Number*. A number from 1 to 999 optionally followed by an uppercase letter (e.g., 43B)
    - *Floor*. Either “st” or a number from 1 to 99
    - *Door*. “th”, “mf”, “tv”, a number from 1 to 50, or a lowercase letter optionally followed by a dash, then followed by one to three numeric digits (e.g., c3, d-14)
    - *Postal code* and *town*. Randomly extracted from the provided database addresses.sql
    - More information at https://danmarksadresser.dk/regler-og-vejledning/adresser

- *Mobile phone number*
    - Format: Eight numeric digits
    - It must start by some of the following digit combinations: 2, 30, 31, 40, 41, 42, 50, 51, 52, 53, 60, 61, 71, 81, 91, 92, 93, 342, 344-349, 356-357, 359, 362, 365-366, 389, 398, 431, 441, 462, 466, 468, 472, 474, 476, 478, 485-486, 488-489, 493-496, 498-499, 542-543, 545, 551-552, 556, 571-574, 577, 579, 584, 586-587, 589, 597-598, 627, 629, 641, 649, 658, 662-665, 667, 692-694, 697, 771-772, 782-783, 785-786, 788-789, 826-827, 829
    - More information at https://digst.dk/media/1cllls3q/nummervejledning-2025.pdf

Functionalities. There must be one function/method and a corresponding API endpoint for each of the following functionalities:

- Return a fake CPR
- Return a fake first name, last name and gender
- Return a fake first name, last name, gender and date of birth
- Return a fake CPR, first name, last name and gender
- Return a fake CPR, first name, last name, gender and date of birth
- Return a fake address
- Return a fake mobile phone number
- Return all information for a fake person
  (CPR, first name, last name, gender, date of birth, address, mobile phone number)

- Return fake person information in bulk (all information for 2 to 100 persons)

It is at the group's convenience to choose:

- The programming language(s) and framework(s)
- The API format (e.g., REST)
- Whether a database other than MySQL/MariaDB will be used (a data migration would be necessary)
- The output format of both the functions/methods and the API (e.g., JSON, XML, CSV, formatted text, standard output)

Feel free to use my application code (either directly or as inspiration):

- Backend (MariaDB / PHP8): https://github.com/arturomorarioja/fake_info/
- Frontend (JavaScript / CSS3 / HTML5): https://github.com/arturomorarioja/js_fake_info_frontend

Testing

The following testing-related tasks must be implemented:

- Write unit tests and integration tests whenever it is considered appropriate
- Write API tests in Postman, Insomnia, ThunderClient or similar
- Design the test cases based on:
    - Black-box design techniques (manual analysis)
    - White-box design techniques (automated analysis with tools)
- Use static testing tools (beyond linters) to analyse and improve the code.
  Also for white-box analysis, if the chosen unit testing framework does not provide it
- Create a Continuous Integration job or pipeline to test the integration of the application
- Implement continuous testing by running all unit and integration tests in the CI process
- Design and write end-to-end test(s) in Cypress, Selenium WebDriver, Playwright or similar (not Selenium IDE)

Delivery

The assignment must be delivered to Fronter in one zip file by 27 October 2025, 23:59:

- The source code of the application
- The database creation and population script if it is different from the one provided
- The source code of the unit tests, integration tests in code, and end-to-end tests
- The CI configuration files
  (YAML, XML or whatever format the CI tool uses. A screenshot is also acceptable)
- A pdf file explaining how black-box design techniques were applied to the definition of test cases
  (e.g., definition of valid and non-valid partitions, boundary values, decision tables if applicable…)
- Screenshots of the static testing tool(s) used that show the value they brought to the testing activities

Presentation

On 28 October 2025 each group will make a presentation where they will show and explain the whole assignment to the rest of the class:

- Demo of the application at work
- The code of the application
- The code of the unit, integration and end-to-end tests
- The code of the API tests
- The criteria behind test case design:
  - Black-box techniques
  - White-box techniques
- The value brought by the static testing tools used (including white-box analysis, if appropriate)
- The CI job or pipeline
  - Walkthrough of the configuration
  - A demo of the process

Presentation time is limited to 8 minutes per group and all group members must present a part.

person-names.json ⋯

addresses.sql ⋯