BACKEND ASSESSMENT

V4.0 23-12-2021

BACKEND WEB DEVELOPMENT ASSESSMENT

Congratulations, you have made it to the next phase of DTT's application procedure!

This phase will consist of a backend web development assessment made in **PHP**. During your time at DTT, you will work for strong international brands, including Greenpeace, Randstad, ING, Philips, Rabobank, FOX Sports, and KPMG. Through this assessment , you can demonstrate that you are capable of contributing to DTT and her clients. DTT promises her clients a detail orientated approach, a high quality standard for deliveries, and a work ethic that goes that extra mile: we look for applicants that understand these values.

This assessment gives us a better understanding of how you work, and provides you with insight into what you can expect when working at DTT.

Keep in mind:

I. Log the hours you spend on the project in the provided Excel document, provide a detailed description of your activities to help us in the grading process.

II. We strongly advise you to complete the **user stories** within 20 hours, and to adjust your scope based on this advisory deadline. Time taken for **bonus points** will not be included in the final time evaluation.

THE ASSESSMENT

We challenge you to develop an API that stores and retrieves information about catering facilities. For this assessment , it is important that you understand the concept of CRUD. Your project should be **straight-forward** and you should keep things **simple**. The code should be secure, properly structured, well commented and object-oriented (OOP).

Functional requirements:

I. MySQL/MariaDB should be used for storage of the data.
II. Both the request body and response data have to be in JSON format.
III. The API should follow the REST architectural style (with correct HTTP Status Codes).

User stories

DTT uses user stories as a baseline for all our (technical/design) documentation. Make sure your assessment adheres to the user stories and conditions of satisfaction as stated below.

For the minimal viable product, the following user stories should be implemented:

User story 1: As a developer, I want to create a database for persistent storage of the data.

- *The data types should be chosen appropriately.*
- *Primary and foreign keys should be used correctly.*
- *Junction tables should be used correctly.*
- ***Schema requirements:***
  - ii. *A **Facility** has a name and creation date.*
  - iii. *A **Tag** has a name.*
  - iv. ***Tag** names MUST be unique and re-usable amongst different facilities.*
  - v. *A **Location** has a city, address, zip code, country code and phone number.*
  - vi. *A **Facility** can have multiple **Tags**.*
  - vii. *A **Facility** always belongs to one **Location**.*

User story 2: As a user I want to be able to manipulate the **facilities** within my API.

- *I want to be able to **Create** facilities and their tags.*
- *I want to be able to **Read** facilities and their tags.*
- *I want to be able to **Update** facilities and their tags.*
- *I want to be able to **Delete** facilities and their tags.*

User story 3: As a user, I want to be able to search for facilities with a **query string.**

- *I want to be able to search facilities by the **facility** name, **tag** name, or **location** city.*
- *I want to be able to apply the search criteria in a **single** API call that also*

allows partial matches.

Bonus points for a good impression (all optional)

I. Include API documentation to make it easier for us to assessment (example: Postman collection).
II. Implement pagination for the results (extra bonus for cursor pagination).
III. Implement 'employees' of the facilities to further display your understanding of database table relationships.
IV. Add an authentication mechanism to secure the API endpoints.
V. Include configuration that would allow the API to run as a Docker container(s).
VI. Write unit/integration tests using PHPUnit framework.
VII. Have a fun idea? We'd love to see it :)

These points are completely optional and do not count towards your 'total time'. We do not negatively judge an assessment that does not include bonus features, but differentiating your assessment does increase your chances to get invited for a follow-up.

Limitations

I. The API should be written in plain PHP **without** relying on 3rd party frameworks (Laravel, Symfony, etc).
II. It is not allowed to make use of API generators.
III. When consulting an online source for the generation, it would aid our grading process if you'd refer to this in your hour logs. We are fine with a tutorial being used as long as the assessment deviates/adds enough for us to grade your ability to create something of your own.

Delivery

When you are satisfied with your API, **zip** your project (including your hour log, documentation and database dump) and send a WeTransfer link to apply@d-tt.nl. .

If you worked with a GIT repository, please follow the steps below:

I. Send the link to the Git repository to *apply@d-tt.nl* and make sure to explain that you're done with the project. Also, please invite *apply@d-tt.nl* as a contributor with Admin privileges
II. Make sure that the repository includes the full project source code.
III. Make sure that the repository includes a copy of any other deliverables e.g.:

A. your completed hours log.
B. your documentation.
C. your database dump.
D. etc.

Our review

We review your assessment on different criteria. Functional requirements are graded on their completeness and the chosen method of implementation. The quality and structure of code are graded on (amongst other factors) consistency, clarity, optimisation, extensibility, reusability, and single-responsibility. The hour log is graded on its completeness, language, and level of detail. Your code will undergo the most scrutiny during our review, make sure it is well-structured and commented accordingly.

Amongst other factors, the following points are essential to us:

I. **Standards:** Adhere to PSR standards and implement them in a consistent manner.
II. **Clarity**: You should strive to make your code easy to understand without relying on comments as a crutch. Use comments to explain more complex parts of the code.
III. **Single responsibility**: Is the code maintainable, isolated, generalised, reusable. Always avoid duplicate code.

Adherence to these criteria indicates an approach to work we greatly value.

Questions?

If you have any questions regarding the requirements of the assignment, or if you are stuck on a problem for too long, please don't hesitate to contact us.

More DTT

Feel free to have a look at all our apps at: https://www.d-tt.nl/en/apps.