DIGGIT

Job Simulation Report

BACK-END DEVELOPER • 26 JUNE, 2023 • DIGGIT.IO

CANDIDATE

Alexander Brandhaug
Seeking a Back-end Dev role

Email a.brandhaug@gmail.com

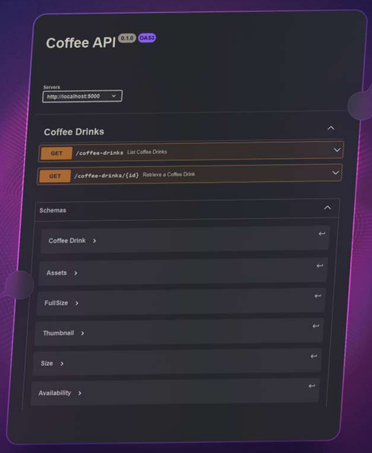
CHALLENGE

TITLE

Coffee API

SUMMARY

Build two brewing hot API endpoints.



SCORE

A **Great** ✨ Score

The candidate achieved a **96%** score. This is **36%** more than our estimated minimum to do well in a back-end developer job role.

** On the screen you can see what the candidate built.

EVALUATION

" This back-end developer is **beyond job ready**. Strengths: API implementation, code quality & package selection. Main area of improvement: version control."

Simen Fossnes
7 years of experience in Full-Stack Development

Scroll down
for more details

Score Details

Great **Quality** means the output scored high on **feature completion** and **qualitative measures**.

Feature Completion, also known as functional requirements, defines the function of something. In DIGGIT job simulations, they are made explicit in the form of tasks. Tasks can be high-level, as user stories. Or low-level, with more technical details.

Qualitative measures, also known as non-functional requirements, are used to assess the overall operation of a product, relative to expecations. In DIGGIT job simulations, they are made explicit in the form of quality assurance criteria. They can describe higher-level user expectations, or lower-level technical aspects.

Feature Completion

Initialize Project 90% | 2.7 / 3 points

Initialize the project inside the api/ folder, and use an well-organized folder structure.

Acceptance Criteria

- the API project is bootstrapped inside the api/ folder
- a well-organized folder structure is used

Implement Coffee API 95% | 7.6 / 8 points

The Coffee API relies on data about various Coffee Drinks. You can find this data as a json file in the /data folder. You need to provide this data through your endpoints.

Acceptance Criteria

- the API is implemented perfectly according to the Coffee API OpenAPI specification handoff
- modern software development principles are followed
- clear instructions on how to run the project is provided in an easy-to-find location (such as the README.md file found in the ./api/ folder)

Write Unit Tests 100% | 7 / 7 points

Write unit tests for the Coffee API

Tests should help ensure high quality, make maintenance & refactoring easier, and improve the overall developer experience.

Acceptance Criteria

- unit tests are provided, together with clear instructions on how to run them

Qualitative Measures

Software Package Selection 100% | 5 / 5 points

Have you chosen well-designed (e.g. easy to use & understand) highly adopted (e.g. big community, common issues raised & resolved already) and reliable (e.g. good performance, well maintained, no security vulnerabilities) and otherwise solid software packages?

Code Readability 92.5% | 3.7 / 4 points

Is the code easy to understand (i.e. simple naming & syntax) and well presented (i.e. consistent & well documented where needed)?

Readability is the ease with which a reader can understand your code. This includes things such as programmer comments, choice of loop structure, naming, and modern syntax features that improve readability.

Evaluation Details

Job simulations are evaluated in the same way that tasks would be on the job. Verified developers with many years of experience evaluate DIGGIT's job simulations.

Higher Quality: To make expectations more clear to candidates, we improve the quality to a level above normal. This also ensures higher evaluation consistency.

As you can see, the accuracy score is broken down into "Feature Completion" and "Quality Measures". These match closely with the differentiation between functional and non-functional requirements, respectively.

It's worth repeating that we utilize verified developers with high competency and many years of experience to evaluate our job simulations. This provides a much higher level of quality, compared to automated evaluation solutions.

Feature Completion

Initialize Project 90% | 2.7 / 3 points

Initialize your project inside the api/ folder, and use an well-organized folder structure.

" The API is initialized inside the 'api/' folder. Seems like a standard .NET starter has been used. The API project has a very clear folder structure - well done. I'm missing a separate commit for project initialization. This helps colleagues see what code was written, and helps whoever is going to review it. Naming for the API and test project could be more consistent. "

Simen Fossnes
7y Full-Stack Dev experience

Write Unit Tests 100% | 7 / 7 points

Write unit tests for the Coffee API.

" The unit tests are provided in a separate project. They're clearly written, and provide good test coverage. They're well named and easy to read. Really well done. Clear instructions on how to run the tests are provided in the readme file. Very nice. "

Simen Fossnes
7y Full-Stack Dev experience

Implement Coffee API 95% | 7.6 / 8 points

Implement the Coffee API, precisely according to the OpenAPI specification.

" Both endpoints work according to the spec. Data models adhere perfectly to the specification. Nice. The error codes are even better than expected based on the spec file. Also nice. The project is running on port 5212, whereas the spec file expects it to run on port 5000. Not so nice. Modern software development principles are indeed followed. The readme file includes clear instructions, with some room for improvement. If you're running the project purely from the terminal, you need to navigate into each project separately (e.g. instead of 'cd dark-roasted-coffee-api/api' you need to do 'cd dark-roasted-coffee-api/api/dark-roasted-coffee-api'). This might not be the case in the standard visual studio editor. Also, step 3 and 4 can be joined, since 'dotnet run' automatically takes care of the build step. "

Simen Fossnes
7y Full-Stack Dev experience

Quality Measures

Code Readability 92.5% | 3.7 / 4 points

is the code easy to understand (i.e. simple naming & syntax) and well presented (i.e. consistent & well documented where needed)?

" Overall, the project is well structured with highly readable code. Naming is good. Clear instructions documenting how to run the project and tests is provided. Very little unnecessary stuff. There's a few minor drawbacks that impact readability: A lot of files that should not have been commit has been, including the bin & obj folders. In other words, version control could improve. Also, the naming between the API and test projects should be more consistent: one use kebab case, and the other use pascal case. But all in all, it's a highly readable codebase. "

Simen Fossnes
7y Full-Stack Dev experience

Software Package Selection 100% | 5 / 5 points

Have you chosen well-designed, highly adopted, reliable and otherwise solid software packages?

" Well-designed, highly adopted and reliable packages have been selected. Reputable sources like Microsoft, Swashbuckle and Netwonsoft are the main providers. In addition, packages like Moq and xUnit are really solid for testing. Great package selection. "

Simen Fossnes
7y Full-Stack Dev experience

EVALUATION SUMMARY

Highly recommended
by the Evaluator

The candidate did well on "Software Package Selection", "Code Readability", "Implement the Coffee API", "Write Unit Tests" and "Initialize Project".

Highly recommended

" This back-end developer is **beyond job ready**. Strengths: API implementation, code quality & package selection. Main area of improvement: version control. "

Simen Fossnes
Chief Technology Officer
7y Full-Stack Dev experience

DIGGIT