Home Assignment

This assignment was carefully designed to provide you with the opportunity to demonstrate your skills relevant for the Full Stack Developer role and to get a feeling of what this role is about.

The assignment consists of 3 mandatory parts and 1 optional part. Each part assesses different skills: back-end, front-end, bundling all parts as a single app. Each part is designed to take you up to 1 hour in the ideal conditions: you are familiar with all the required tools, you are focused and there are no distractions. In case you need to consult with the docs of a library/framework, it should take you no more than 2 hours. If you feel that the whole assignment will take you more than 6 hours, your skill set may not match our requirements, so please consider whether the effort is worth the invested time.

Successful candidates should complete all 3 mandatory parts. The solution should be submitted as a public repository under your Github/Gitlab account. Any build/run instructions should work in a Linux-based environment.

Part I

Build a web server implementing the API described below to run a wiki. In this case, a wiki is basically a key-value storage. You can store any content under a certain name. Later you can retrieve the latest stored content for a particular name.

API

Operation	HTTP Request	HTTP Response

List stored articles	`GET /articles/`	`HTTP/1.1 200 OK`
		`Content-Type: application/json`
		
		The payload is a JSON array listing the names of all stored articles

Store an article	`PUT /articles/:name`	If a new article was created:
		`HTTP/1.1 201 Created`
		No payload
		
		If an existing article was updated:
		`HTTP/1.1 200 OK`
		No payload

Read an article	`GET /articles/:name`	If the article is not found: HTTP/1.1 404 Not Found No payload If the article is found: HTTP/1.1 200 OK Content-Type: text/html The payload is the latest content stored under this name

Notes:

- Article name can be any non-empty valid Unicode string
- Article content can be any valid Unicode string
- No articles are persisted between server restarts

Allowed Technologies

Programming Language: GO
External Libraries: none

Additional Acceptance Criteria

- The server can be built using `` `go build` `` (or `` `go run` ``)

- The solution contains an appropriate level of unit testing that is run using `go test`
- All steps to build and run the server are documented in a README file
- When the server is running locally it should be accessible using:
    - `curl http://localhost:9090/articles/`
    - `curl ‘http://localhost:9090/articles/rest_api’`
    - `curl -X PUT http://localhost:9090/articles/wiki -d ‘A wiki is a knowledge base website’`

Part II

Build a single page application for managing a Markdown-based wiki. The app should use the webserver written in Part I as a store for the articles.

Requirements

Page	URL	Description
Index page	`/`	• Displays the names of all stored articles • Every article name is a link to the article's viewing page
Article viewing page	`/:name`	• Displays the article name as a header at the top of the page • Displays a visible edit button at the top of the page leading to the article's edit page • Displays the article's content rendered as HTML • Displays `No article with this exact name found. Use Edit button in the header to add it.` message instead of the article's content if the article is not found
Article edit page	`/edit/:name`	• Displays article name as a header at the top of the page • Has means to edit the article's content as plain text • Has means to preview the article's content rendered as HTML before saving it • Displays Save and Cancel buttons

Note about rendering the content of the articles. Articles' content is assumed to be a markdown text valid according to CommonMark spec. If the content is not a valid CommonMark markdown, the result of rendering it to HTML is undefined.

Allowed Technologies

Programming Language: Javascript
UI Frameworks: React or VueJS
External Libraries: anything you can find in npm registry

Additional Acceptance Criteria

- The entire solution can be built using `npm install && npm build`
- The solution contains an appropriate level of unit testing that is run using `npm run test`
- All steps to build and run the solution are documented in a README file

Part III

Dockerize the wiki app you have built in Part I and in Part II.

Requirements

- The entire solution can be built using a single command
- The build command should generate a docker image called “md-wiki” with the tag “2019”
- The generated docker image should include both the back-end and the front-end
- All steps to build and run the solution are documented in a README file
- When the docker image is started as follows:
  ```
  docker run -ti -p 8080:8080 md-wiki:2019
  ```
  the SPA from Part II is available at http://localhost:8080/ .
- Nginx should be used to act as a reverse proxy so that the SPA and web server appear as one web site on the local machine

Allowed Technologies

- Docker
- Nginx as a reverse proxy
- Any other open source tools/technologies

Part IV (Optional)

Configure a CI (Continuous Integration) pipeline for the project. Please note this part is optional, but candidates submitting it will get extra points that can compensate for minor issues made in previous parts.

Requirements

- The pipeline should build and run tests for Part I
- The pipeline should build and run tests for Part II
- The pipeline should fail if any of these steps fail

Allowed Technologies

- You can use one of the following CI/CD services: Travis, CircleCI, Github Actions, Gitlab CI/CD.