Local Web Server Documentation

Technical Assessment Interview

Tendable

April 2022

Contents

Abstract

This is the documentation for the local web server provided to you for your technical assessment interview. This is simply a way to document the endpoint behaviour and JSON models. It is not necessary that you use all of these in your project.

1 General Guidance

Running the web server is really easy. In the terminal you need to **cd** into the main directory of the web server and then run **python run.py**. This will launch the server on **localhost:5001**

2 Endpoints

2.1 Basic Authentication

Authentication is not required to access the other endpoints. The following endpoints exist purely as a tool for you to develop the login task.

Note: There is no persistence of registrations between runs of the web server - every time you stop and run the server again all the registered users will be lost.

- **POST** `/api/register` This endpoint registers a user so that it's later available for the login endpoint. The server expects the json to contain an "email" field and a "password" field.

  Example Body:

  ```
  {
    "email": "johnd@email.com",
    "password": "dogsname2015"
  }
  ```

  Returns:

  - **200** if the registration was successful (both the email and password fields were provided)
  - **400** if the registration was unsuccessful due to one or both fields missing from the JSON
  - **401** if the user already exists

- **POST** `/api/login` This endpoint verifies user credentials against already registered users.

  Example Body:

```
{
  "email": "johnd@email.com",
  "password": "dogsname2015"
}
```

Returns:

- **200** if the login was successful (the user was previously registered with the /api/register endpoint)
- **400** if the email or password fields are missing from the JSON
- **401** if the user does not exist or the credentials are incorrect

2.2 Inspections

- **GET** `/api/inspections/start`

  This endpoint returns a new inspection (no selected answers). Note that this endpoint returns the same new inspection every time, except for the inspection id which increments every time the endpoint is called.

  Returns:

  - **200** with an inspection body attached

- **GET** `/api/inspections/submit`

  This endpoint expects a json body which will be submitted (saved into a file in /inspections at the root of the webserver). Note that this endpoint does not do any validation against the json format, with the exception of having an inspection object and an id inside the inspection object.

  Returns:

  - **200** if the inspection was saved
  - **500** if there was an error (such as the inspection not having an id)

2.3 Other available endpoints

- **GET** `/api/generate_random_inspections/<count>`

  This endpoint generates random inspections that can later be interacted with via other endpoints. This is generally expected to be run from the web browser and it only needs to be run once. Subsequent calls to this endpoint will overwrite the previously generated inspections.

  Returns: 200

Example usage:
`localhost:5001/api/generate_random_inspections/10`
This will generate 10 random inspections.

- **GET** `/api/random_inspection`

  This endpoint returns a random inspection from the generated inspections, if they have been generated using the endpoint above.

  Returns:

  - **200** with an inspection body if the inspections have been generated with the `generate_random_inspections` endpoint
  - **404** if the inspections have not already been generated

  Example usage: `localhost:5001/api/random_inspection`

- **GET** `/api/inspections/<inspectionId>`

  This endpoint returns the inspection with the specified id from the generated inspections, if they have been generated using the endpoint above.

  Returns:

  - **200** with an inspection body if the inspection was found (was already generated with the `generate_random_inspections` endpoint, and it wasn't deleted).
  - **404** if the inspection could not be found

  Example usage: `localhost:5001/api/inspections/3`

- **DELETE** `/api/inspections/<inspectionId>`

  This endpoint deletes an inspection with the specified id, if it exists.

  Returns:

  - **200** if the inspections was found and deleted successfully
  - **404** if the inspection could not be found

  Example usage: `localhost:5001/api/inspections/3`

3 Models

3.1 Inspection Model

3.1.1 Inspection Type Model

```
"inspectionType": {
    "id": 1, (integer)
    "name": "Clinical", (string)
    "access": "write" (string enum, possible values: none, read, write)
}
```

3.1.2 Area Model

```
"area": {
    "id": 1, (integer)
    "name": "Emergency ICU" (string)
},
```

3.1.3 Answer Choice Model

Note that in the case of Not Applicable answer choices (or N/A), the answer choice id is always -1

```
{
    "id": 1, (integer)
    "name": "Yes", (string)
    "score": 1.0 (double, value between 0 and 1)
}
```

3.1.4 Question Model

Note that the selectedAnswerChoiceId is an optional integer. When this value is null it means that no answer has been selected

```
{
    "id": 1, (integer)
    "name": "Is the drugs trolley locked?", (string)
    "answerChoices": [AnswerChoiceModel], (array of answer choice models)
    "selectedAnswerChoiceId": 1 (optional integer)
}
```

3.1.5 Category Model

```
{
    "id": 1, (integer)
    "name": "Drugs", (string)
    "questions": [QuestionModel], (array of question models)
}
```

3.1.6 Survey Model

```
{
    "id": 1, (integer)
    "categories": [CategoryModel], (array of category models)
}
```

3.1.7 Inspection Model

```
{
    "id": 1, (integer)
    "inspectionType": InspectionTypeModel,
    "area": AreaModel,
    "survey": SurveyModel
}
```

3.1.8 Full Inspection Model Example

This is an example inspection that could be returned by the endpoints. When an inspection is returned, it is passed as the value of the "inspection" key.


```
{
    "inspection": {
        "id": 1,
        "inspectionType": {
            "id": 1,
            "name": "Clinical",
            "access": "write"
        },
        "area": {
            "id": 1,
            "name": "Emergency ICU"
        },
        "survey": {
            "id": 1,
            "categories": [
                {
                    "id": 1,
                    "name": "Drugs",
                    "questions": [
                        {
                            "id": 1,
                            "name": "Is the drugs trolley locked?",
                            "answerChoices": [
                                {
                                    "id": 1,
                                    "name": "Yes",
                                    "score": 1.0
                                },
                                {
                                    "id": 2,
                                    "name": "No",
                                    "score": 0.0
                                }
                            ],
                            "selectedAnswerChoiceId": 1
                        }
                    ]
                }
            ]
        }
    }
}
```

3.2 Authentication Model

The server expects these fields for both the login and register endpoints. There is no validation being done against the format of the value for these fields.

```
{
    "email": "johnd@email.com" (string),
    "password": "dogsname2015" (string)
}
```

3.3 Error Model

Some endpoints can return error codes such as 400, 401, 404. Generally when this happens, a JSON body is attached with an error message.

```
{
    "error": "Invalid user or password" (string)
}
```