API Lifecycle Exercises

Version 1.0 23.4.2023

Thomas Bayer

predic8 GmbH
Koblenzer Str. 65

53173 Bonn

Content

1. Preparation 3
1.1. Visual Studio Code 3
1.2. Node (Optional) 3
1.3. Oasdiff (Optional) 3
1.4. Projects 4
1.4.1. rfq-api 4
1.4.2. rfq-server 4
1.4.3. API Gateway 4
2. JSON 5
3. JSON Schema 6
4. YAML 9
5. OpenAPI 11
6. git 17
7. Linting 24
8. Github Actions 28
9. Github Action to for Linting 30
9.1. Badge 32
10. Protect Main Branch 34
10.1. New Branch 36
10.2. Pull Requests 39
10.3. Run Actions on Merge Request 40
11. Security Checks 43
11.1. Making the OpenAPI more secure 46
11.2. Additional Properties (Optional) 51
12. Samples (Optional) 52
13. Server Implementation 53
13.1. Implement 57
13.2. Preparation(Instructor only) 59
14. API Gateway 60
15. API Diff 62
16. Troubleshooting 64
16.1. github 64
16.2. Azure 64

1. Preparation

1.1. Visual Studio Code

Install VSC from:

`https://code.visualstudio.com`

1.2. Node (Optional)

Check if you have a decent node installation.

```
› node –v
v19.6.1
```

Or install the latest LTS version:

https://nodejs.org/en

1.3. Oasdiff (Optional)

Install oasdiff on local computer.

`https://github.com/Tufin/oasdiff`

1.4. Projects

1.4.1.rfq-api (Instructor only)

Delete *tubayer/rfq-api* repository at github.

1.4.2.rfq-server (Instructor only)

Steps:
1. Checkout:

```
https://github.com/predic8/rfq-server
```

2. Delete *getQuote* Method in RESTController of repo *predic8/rfq-server*

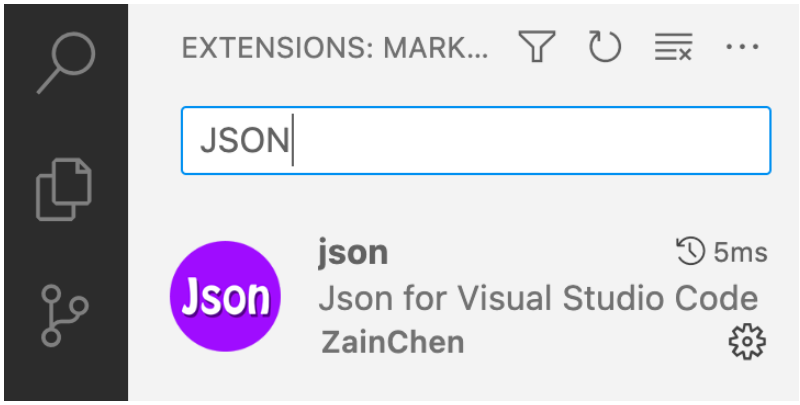
1.4.3.API Gateway (Instructor only)

Steps:
1. Checkout:

```
https://github.com/predic8/api-lifecycle-gateway
```

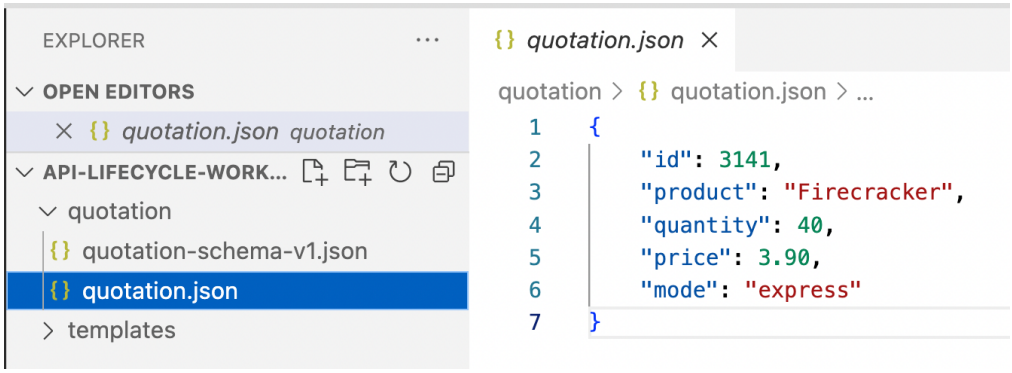
2. JSON

Exercise: JSON Plugin
1. Check if the JSON plugin is installed.



Marketplace: https://marketplace.visualstudio.com/items?itemName=ZainChen.json

Exercise: JSON Sample
1. Unzip the *api-lifecycle-workshop-XXXX.zip* archive.
2. Look at the *file quotation/quotation.json* in VSC.



3. JSON Schema

Exercise: JSON Sample

1. Look at the file *quotation/quotation-schema-v1.json*.

```
{
    "type": "number"
}
```

2. Add a reference to the JSON schema to *quotation.json*.

```
{
    "$schema": "quotation-schema-v1.json",
    "id": 3141,
    "product": "Firecracker",
    "quantity": 40,
    "price": 3.90,
    "mode": "express"
}
```

3. Have a look at the error message.
4. Change the type in the schema to *object*.
5. Have a look at *quotation.json* again. There should be no error anymore.

Exercise: Properties

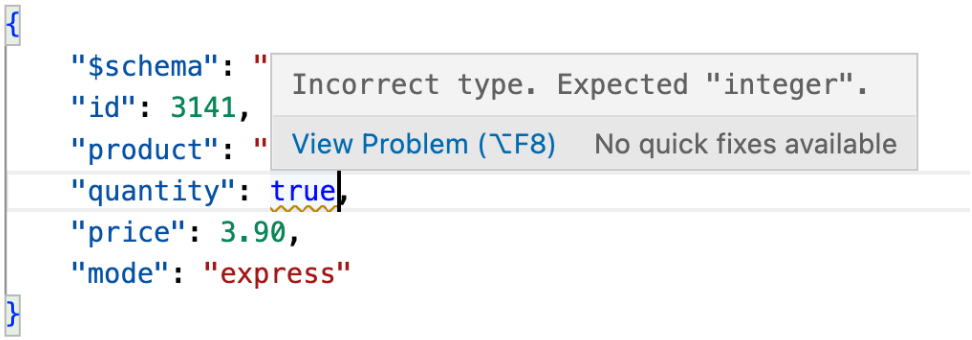
1. Have a look at *templates/quotation-schema-v1.json*.

```
{
    "type": "object",
    "properties": {
        "id": {
            "type": "integer"
        },
        "product": {
            "type": "string"
        },
        "quantity": {
            "type": "integer"
        },
        "price": {
            "type": "number"
        },
        "shipment": {
            "type": "string"
        },
        "mode": {
            "type": "string"
        }
    }
}
```

2. Copy *templates/quotation-schema-v1.json* into the *quotation* folder.

Exercise: Validation

1. Change the type of some properties of the quotation and have a look at the error message.

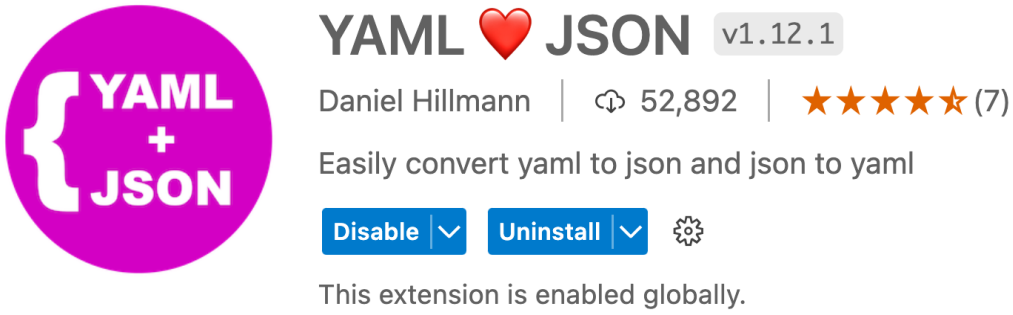


2. Make the file valid again.

4. YAML

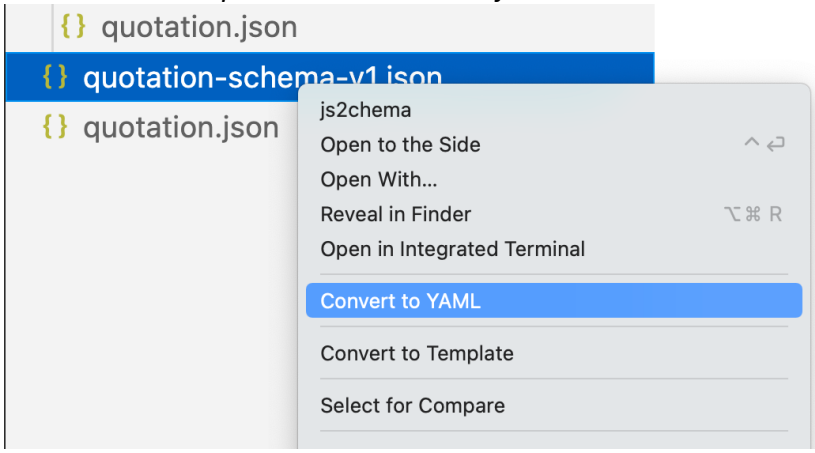
Exercise: JSON 2 YAML

1. Install the YAML to JSON plugin.



Marketplace: https://marketplace.visualstudio.com/items?itemName=hilleer.yaml-plus-json

2. Convert the *quotation-schema-v1.json* to YAML.



3. Have a look at the file *quotation-schema-v1.yaml*.

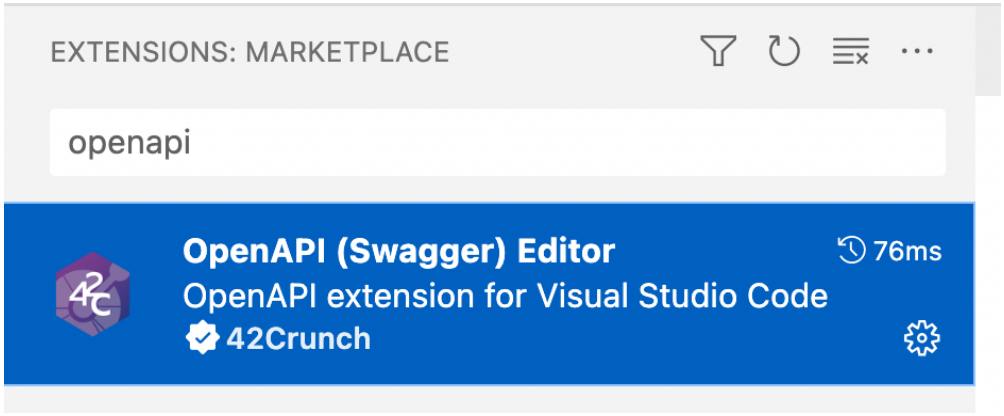
```yaml
type: object
properties:
  id:
    type: integer
  product:
    type: string
  quantity:
    type: integer
  price:
    type: number
  shipment:
```

```
    type: string
  mode:
    type: string
```

4. Fix the reference in *quotation.json*.

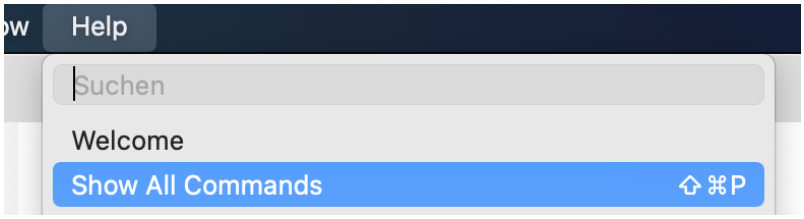
5. OpenAPI

Exercise: OpenAPI Plugin
Install the OpenAPI plugin from 42Crunch.

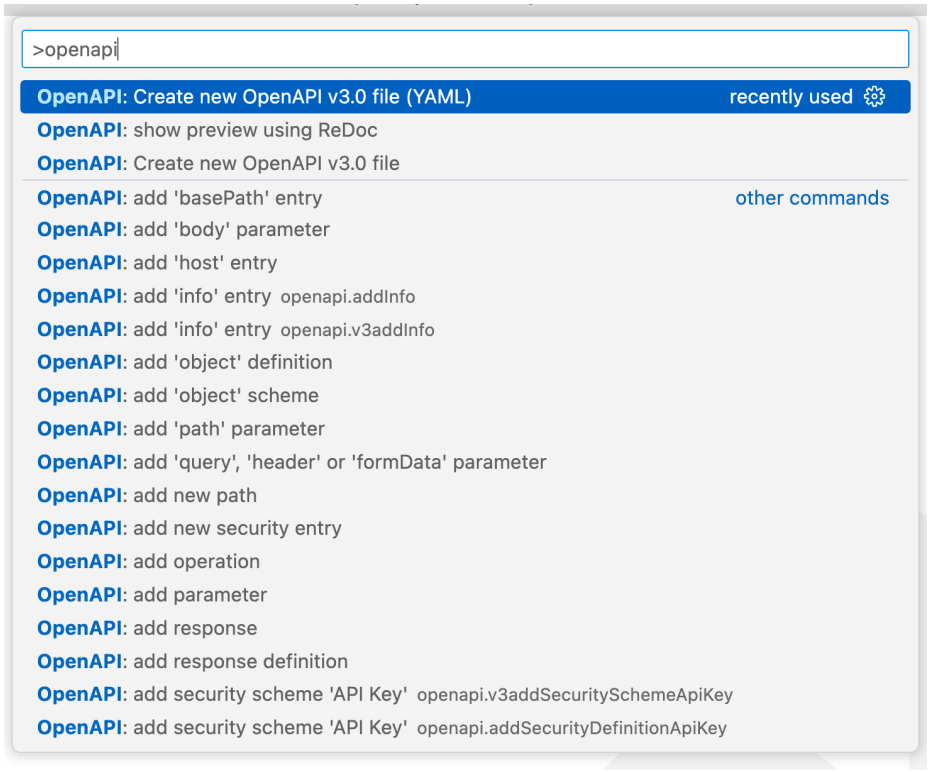


Exercise: Create OpenAPI Document

1. Press CTRL+SHIFT+P or *Help/Show All Commands*.



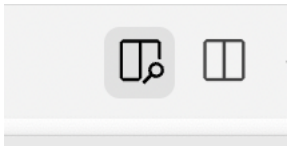
2. Execute the *Create new OpenAPI v3.0 file (YAML)* command



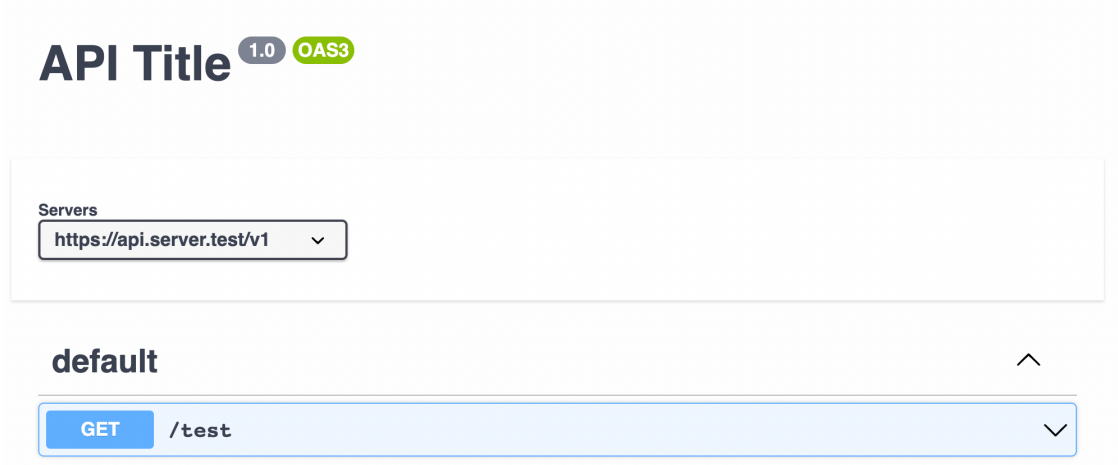
3. Save *as rfq-api-v1.oas.yml*.
4. Change the *title* to *Quotation API*.

```yaml
openapi: '3.0.3'
info:
  title: Quotation API
  version: '1.0'
servers:
  - url: https://api.server.test/v1
paths:
  /test:
    get:
      responses:
        '200':
          description: OK
```

5. Click at the preview symbol.



6. Have a look at the Swagger UI.



Exercise: Type Description

1. Add a *components* section at the end of the file.

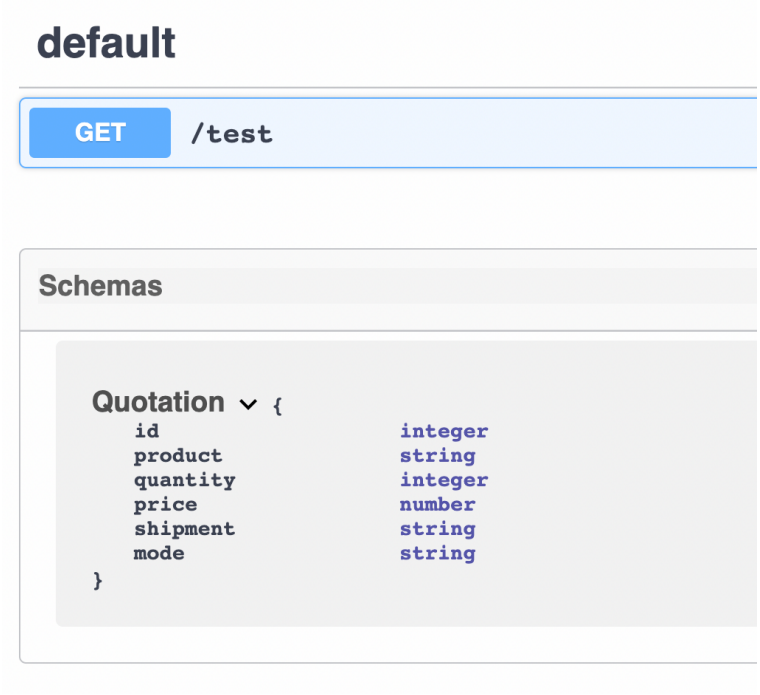
```yaml
openapi: '3.0.3'
info:
  title: Quotation API
  version: '1.0'
servers:
  - url: https://api.server.test/v1
paths:
  /test:
    get:
      responses:
        '200':
          description: OK

components:
  schemas:
    Quotation:
```

2. Copy the content of the file *quotation-schema-v1.yaml* into the OpenAPI document. Pay attention to the right indentation.

```yaml
components:
  schemas:
    Quotation:
      type: object
      properties:
        id:
          type: integer
        product:
          type: string
        quantity:
          type: integer
        price:
          type: number
        shipment:
          type: string
        mode:
          type: string
```

3. Have a look at the Swagger UI.



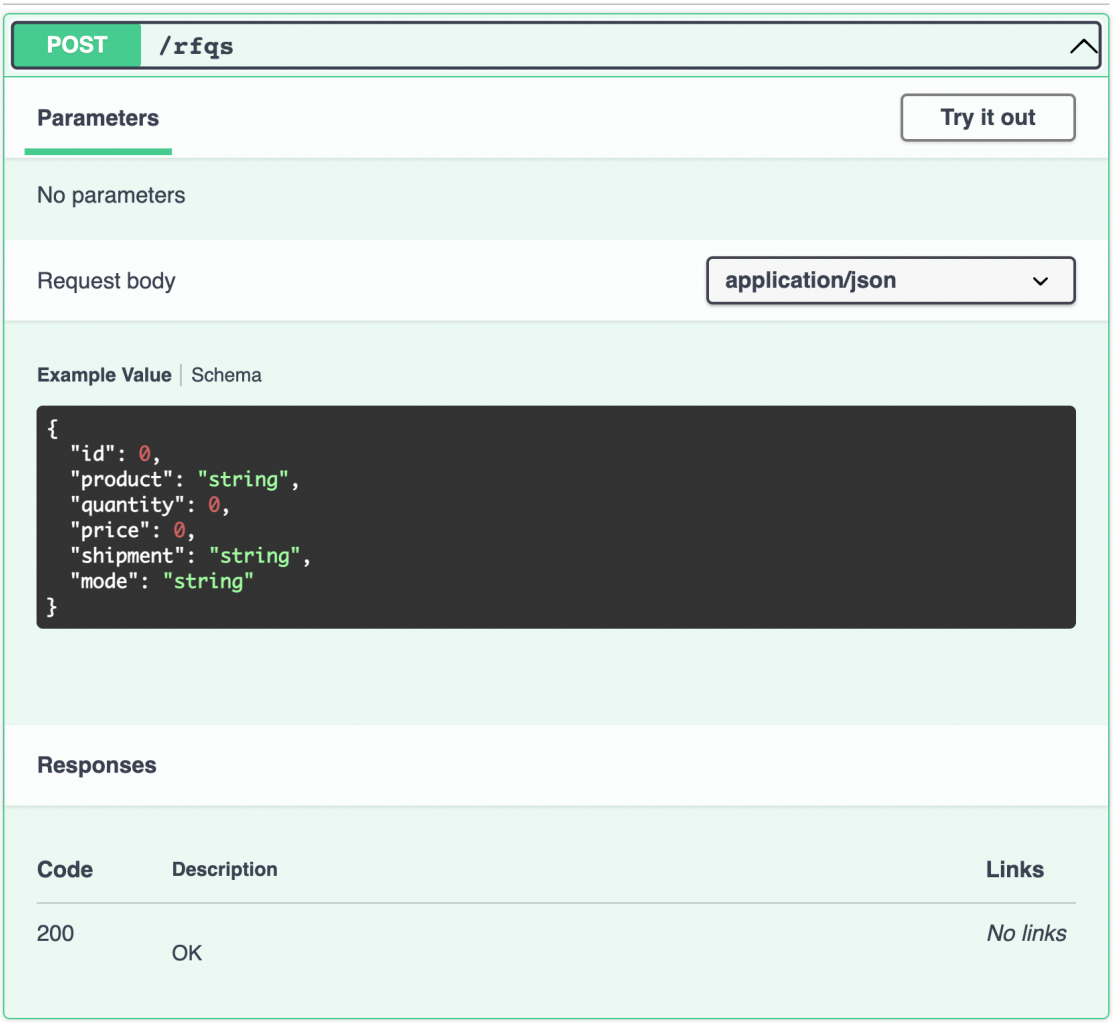
4. Change the path and the method in the OpenAPI document.

```
paths:
  /rfqs:
    post:
      responses:
        '200':
          description: OK
```

5. Use the *Quotation* type to describe the request body.

```
paths:
  /rfqs:
    post:
      requestBody:
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/Quotation"
      responses:
        '200':
          description: OK
```

6. Have a look at the Swagger UI.



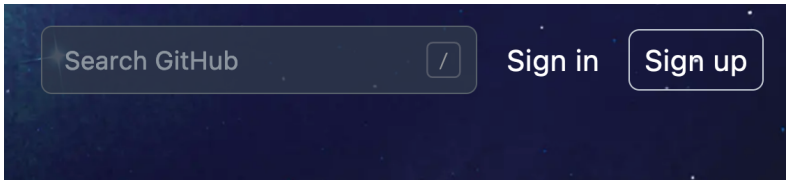
6. git

Exercise: github Account

1. Go to:

https://github.com/

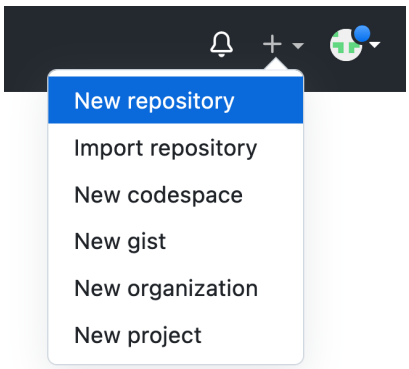
2. Click on **Sign up**



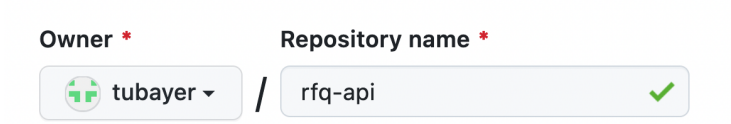
3. Complete registration process.

Exercise: New Repository

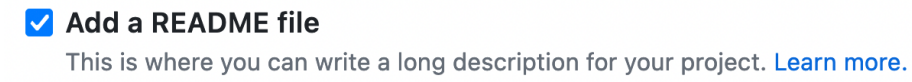
1. Create a new repository at github.



2. Use *rfq-api* as name.



3. Make a check at *Add a README file*.



4. Click on *Create repository*.

Exercise: git Command line Client Installation (If needed)

1. Install git:

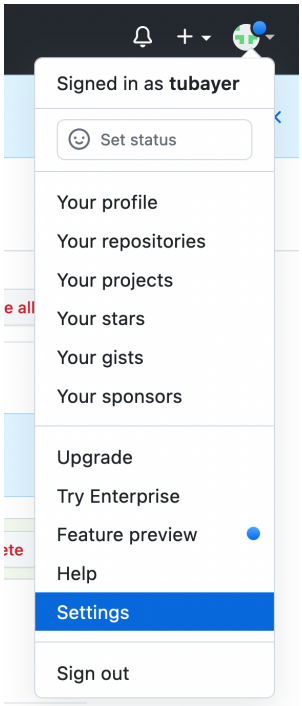
`https://git-scm.com/downloads`

2. Test your Installation.

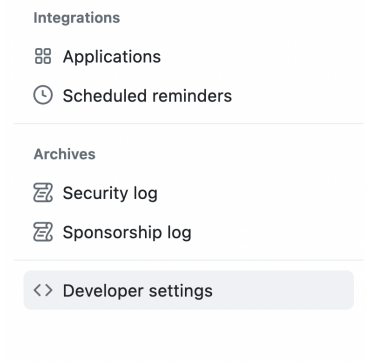
```
❯ git version
git version 2.39.2
```

Exercise: Access Token for Authentication in the CLI

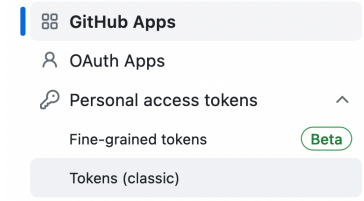
1. Go to https://github.com
2. Click on your avatar at the upper right corner and then on *Settings*.



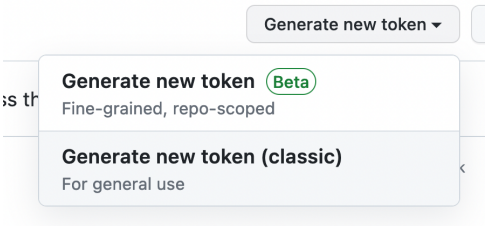
3. Click on Developer Settings at the lower left corner.



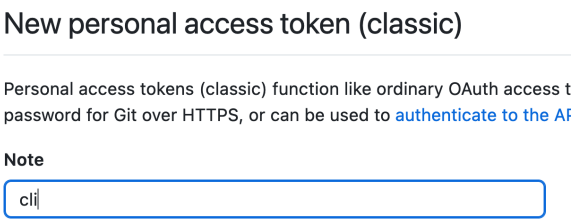
4. Click on *Personal access tokens* and then on *Tokens (classic).*



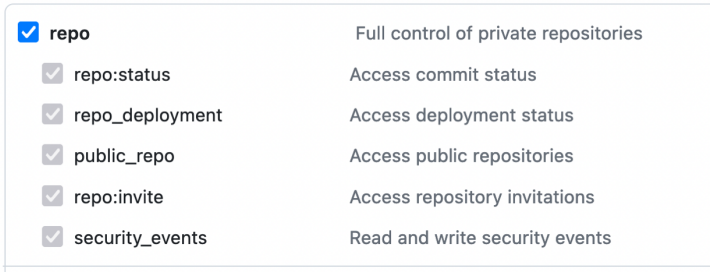
5. Click on *Generate new token* then on *Generte new token(classic)*



6. Use *cli* as Note.



7. Activate the checkboxes for the following permissions.



8. Click on *Gerate token* at the bottom.
9. Copy the token and store it on a save place (e.g. password safe).

Exercise: Clone Repository

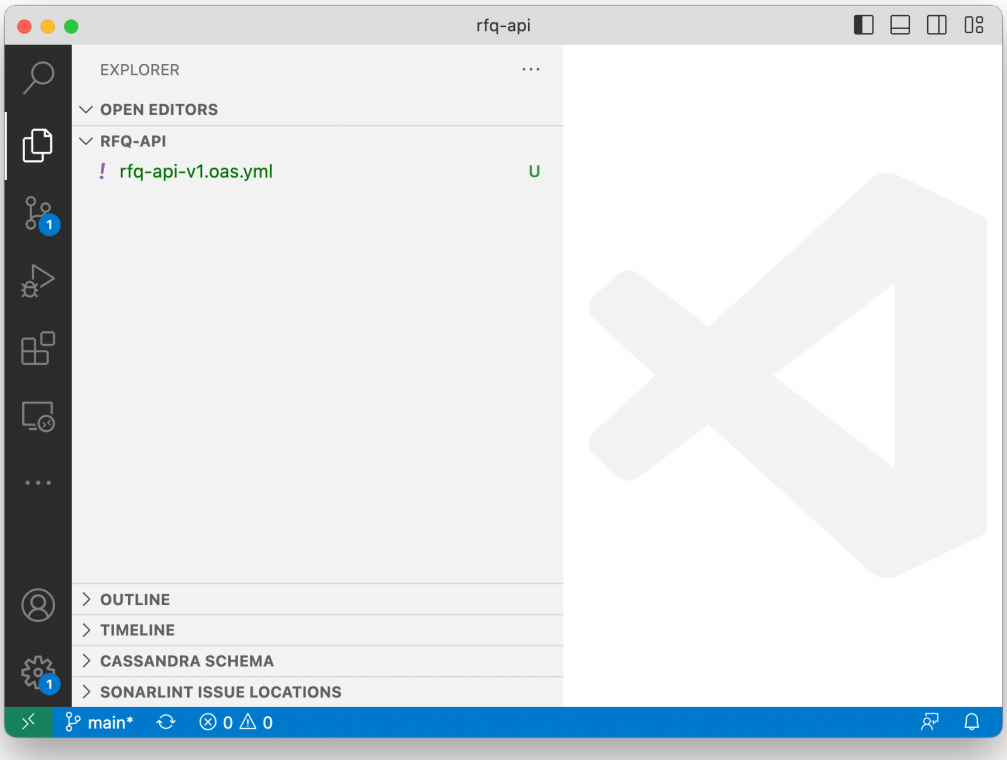
1. Open a terminal in the *api-lifecycle-workshop-XXX* folder.
2. Clone your repository.

```
git clone https://<<Username>>:<<Token>>@github.com/tubayer/rfq-api
```

Exercise: Add OpenAPI to Repository

1. Copy the OpenAPI document from the previous exercises into the *rfq-api* working directory.

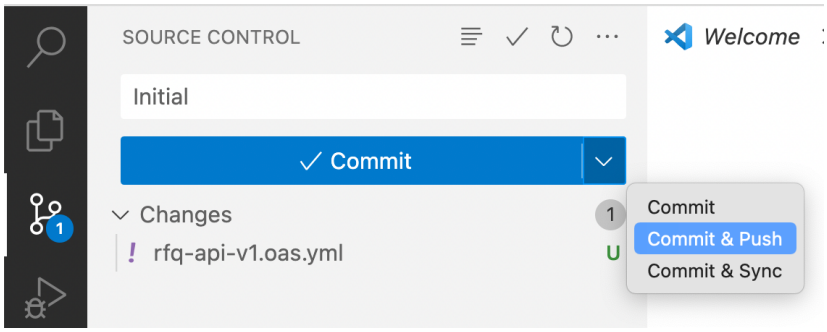
2. Close all VSC Windows and open the folder *rfq-api*.



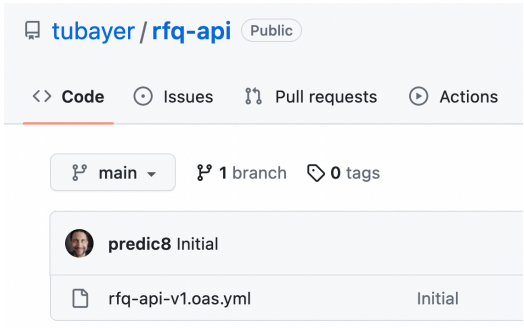
Exercise: Push to github

1. Click on the git symbol on the left side.

2. Type in *Initial* as commit message. Click on the little arrow next to the *Commit* button and then click on *Commit & Push*.



3. Have a look at your repository at github and check if the file was added.



Troubleshooting:

- Use https://cli.github.com/
- vi .git/config

7. Linting

Exercise: Install Node (If needed)

Install Node from https://nodejs.org/en.

Hint: You do not need Node to continue with the exercises. If you cannot install it, do not waste time!

Exercise: Install Spectral Linter

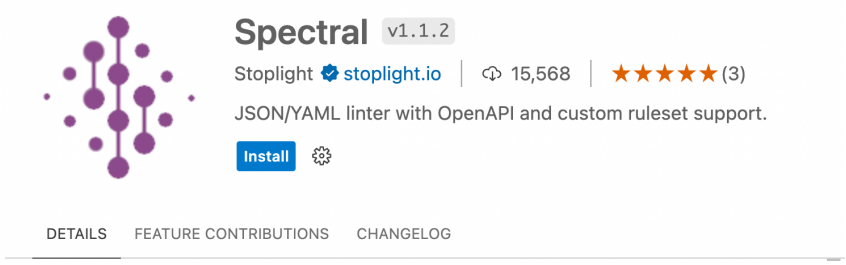
1. Look at:

```
https://github.com/stoplightio/spectral
```

2. Install spectral

```
npm install -g @stoplight/spectral-cli
```

3. Install Sectral Plugin into VSC:



4. Have a look at the file *rfq-api-v1.oas.yml*

```yaml
openapi: '3.0.3'
info:
  title: Quotation API
```

Validation schema for OpenAPI Specification 3.0.X.

`Info object must have "contact" object.` spectral(info-contact)

`Info "description" must be present and non-empty string.` spectral(info-description)

View Problem (⌥F8) No quick fixes available

```yaml
      requestBody:
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/Quotation"
      responses:
        '200':
          description: OK

components:
  schemas:
    Quotation:
      type: object
      properties:
        id:
          type: integer
        product:
          type: string
        quantity:
          type: integer
        price:
          type: number
        shipment:
          type: string
        mode:
          type: string
```

5. Hover over underlined words and have a look at the errors and warnings.

6. Fix the errors:

```yaml
info:
  title: Quotation API
  description: Get a quote online
  version: '1.0'
  contact:
    email: bayer@predic8.de
servers:
  - url: https://api.server.test/v1
tags:
  - name: Quote
paths:
  /rfqs:
    post:
      tags:
        - Quote
      description: Request a quoute
      operationId: getQuote
```

Exercise: Rules File

1. Create the file *.spectral.yml* in *rfq-api* with the following content:

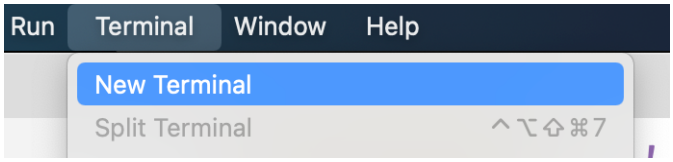
```yaml
extends:
  - spectral:oas
```

Hint: Make sure you spell the filename correct with a dot at the beginning:

```
.spectral.yml
```

Exercise: Running Spectral from the Command line

1. Open a new Terminal in VSC.



2. Run spectral.

```
❯ spectral lint rfq-api-v1.oas.yml
No results with a severity of 'error' or higher found!
```

3. Change *operation**Id*** to *operation**id*** with a lowercase **i** in the OpenAPI document.

```
  operationid: getQuote
```

4. Save.
5. Run the Spectral linter again.
6. Look at the error message.

Hint: Do not correct the error now. Just leave it.

8. Github Actions

Exercise: First Action

1. Create a folder named *.github* inside the *rfq-api* folder. You can use VSC for that by clicking on the new folder icon.

Hint: Pay attention to the correct spelling with a dot at the beginning.

```
.github
```

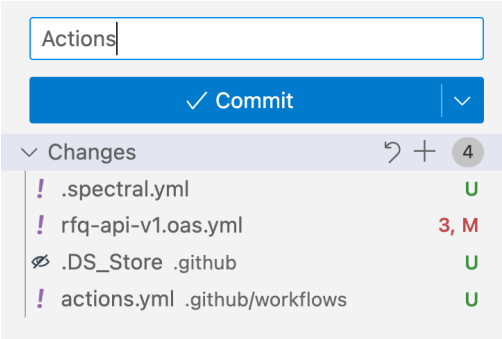
2. Create a second folder named *workflows* inside the *.github* folder.
3. Copy the file *actions.yml* from the templates folder into the *.github/workflows* folder.

```yaml
on:
  - push

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Building..."
```

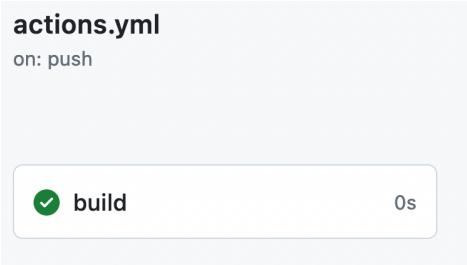
Listing: actions.yml

4. Commit and Push the modifications. Use *Actions* as a commit message.

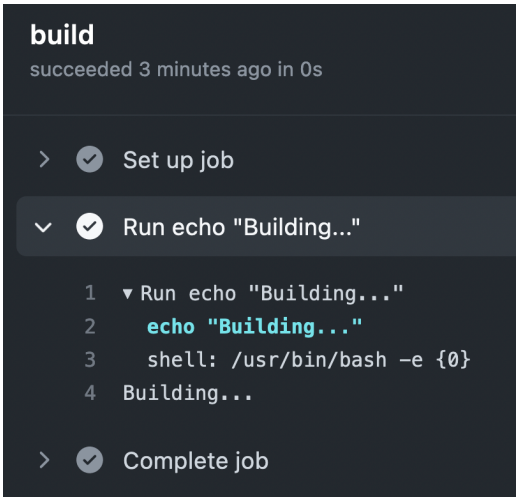


5. Got to the Repository at *github.com*
6. Click at *Actions*. Then click on the workflow

7. Click on build.



8. Have a look at the output.



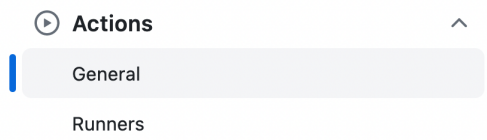
9. Linting OpenAPI with Github Action

Exercise: Workflow permissions

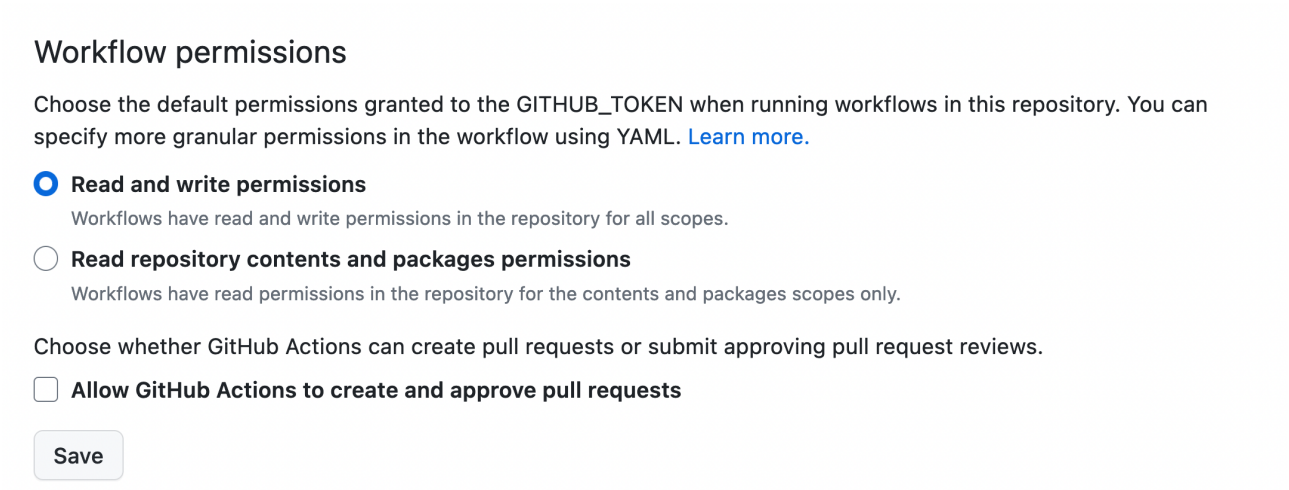
1. Go to the *rfq-api* repository on the *github.com* site.
2. Click on *Settings*.

3. Click on *Actions* on the left.

4. Click on General



5. Choose **Read and write permission** on the bottom.



6. Do not forget to click on *Save* at the very bottom.

Exercise: Action with Spectral Linting

1. Modify the *actions.yml* as follows:

```yaml
name: OpenAPI Checks

on:
  - push

jobs:
  Checks:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - run: npm install -g @stoplight/spectral-cli
      - run: spectral lint rfq-api-v1.oas.yml
```

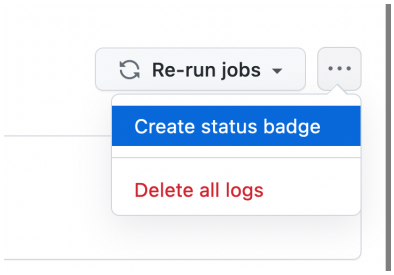
Hint: Do not forget the name property!

2. Commit and push with message *Linting*.
3. Look at the execution of the action at the github website.
4. Look at the error message.

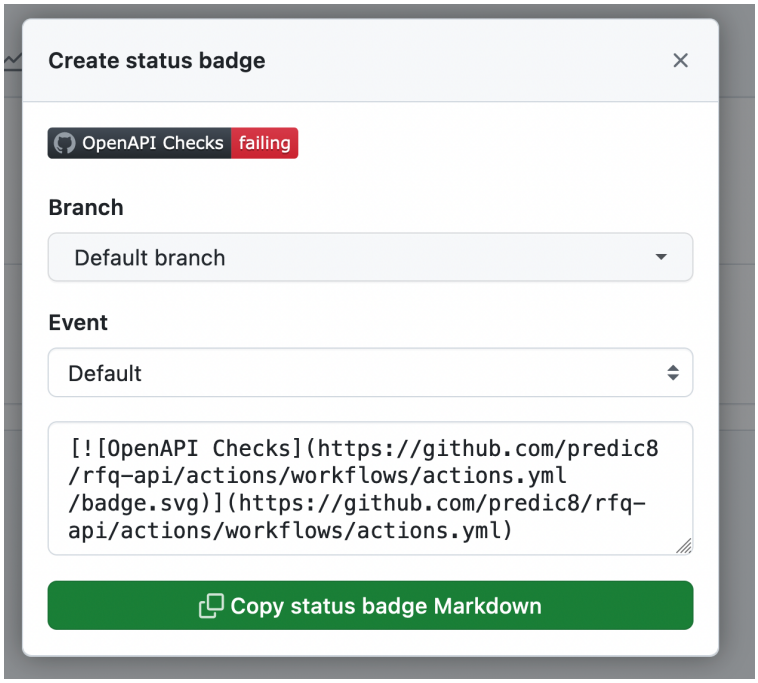
9.1. Badge

Exercise: Add Badge

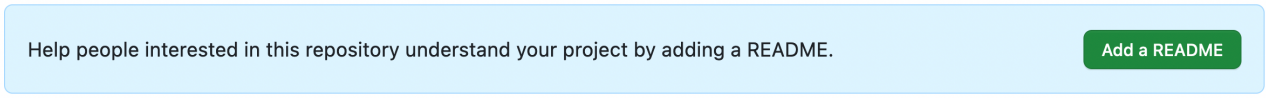
1. Go to the webpage with the workflow result.
2. Click on the three dots at the right and choose *Create status badge*.



3. Click on Copy status badge Markdown.



4. Make sure your project has a *README.md*. If not add one by clicking on Add a *README* on the homepage of the repo. Do not forget to pull!

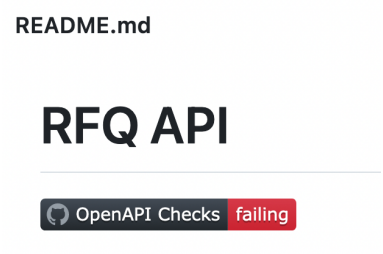


5. Copy the markup for the badge into the *README.md*.

```
# RFQ API

[![OpenAPI Checks](https://github.com/predic8/rfq-api/actions/workflows/actions.yml/badge.svg)](https://github.com/predic8/rfq-api/actions/workflows/actions.yml)
```

6. Commit/Push with message *Badge*.
7. Have a look at the homepage of your repo at github.

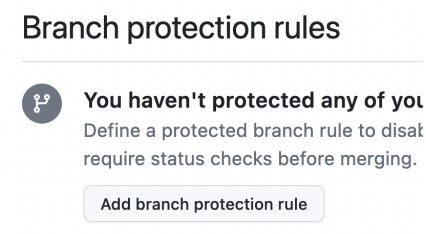


8. Fix the OpenAPI document.
9. Commit/Push with *operation**I**d*
10. Check the homepage again and be proud of your badge! Click reload if it is still red.

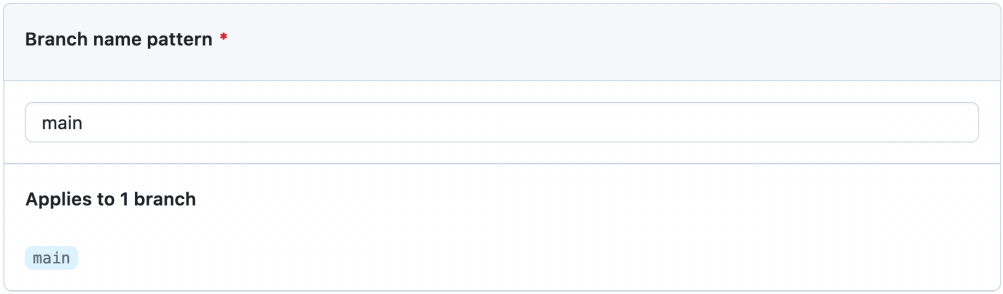
10. Protect Main Branch

Exercise: Protect Main Branch

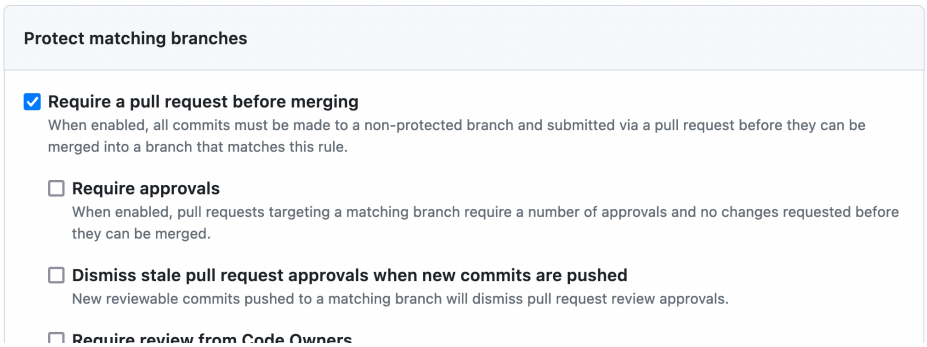
1. Go to the site of the repo then *Settings/Branches*.
2. Click on *Add branch protection* rule.



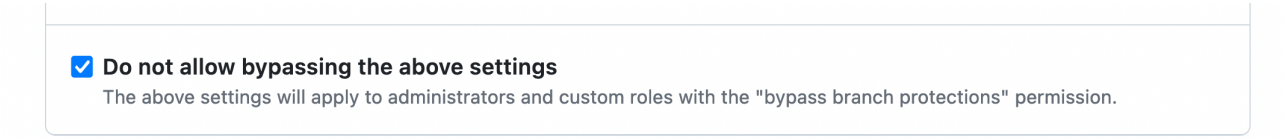
3. Type *main* as *Branch name pattern*.



4. Check *Require a pull request before merging* and uncheck *Require approvals*.



5. Check also *Do not allow bypassing the above settings* further down.



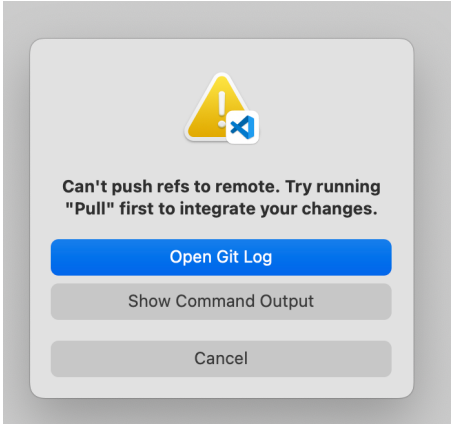
6. Click on *Create*.

Exercise: Try to Push

1. Add an *About* section to the *README.md* in VSC.

```
## About
```

2. Commit/Push with *About*.



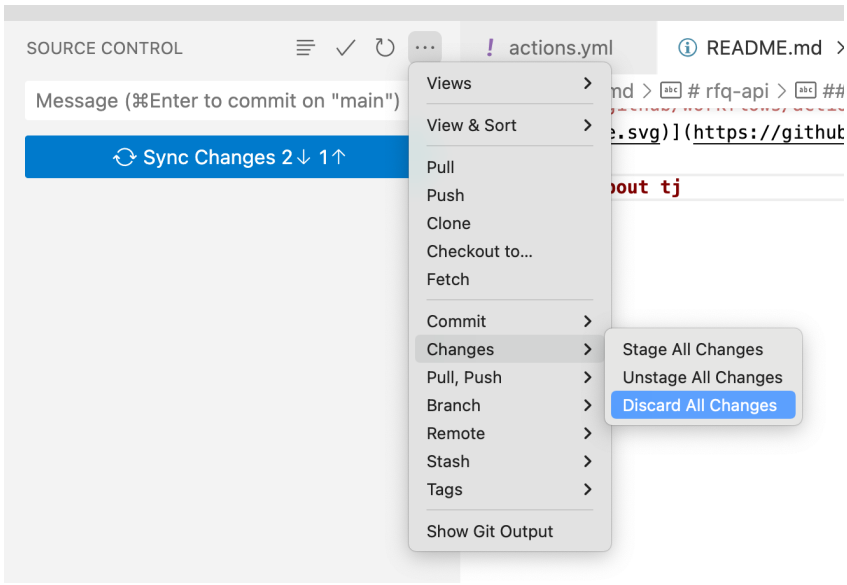
3. Click on *Show Command Output*

```
> git push origin main:main
remote: error: GH006: Protected branch update failed for refs/heads/main.
remote: error: Changes must be made through a pull request.
To https://github.com/predic8/rfq-api.git
 ! [remote rejected] main -> main (protected branch hook declined)
error: failed to push some refs to 'https://github.com/predic8/rfq-api.git'
```

10.1. New Branch

Exercise: Undo Changes.

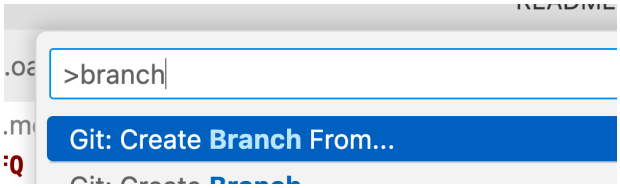
1. Go to the git view in VSC.
2. Click on the *dots/Changes/*Discard All Changes



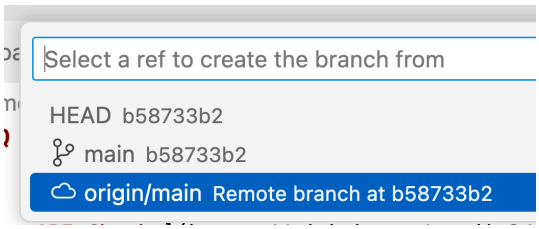
Exercise: New Branch

1. Click on *main* at the lower left corner of VSC.

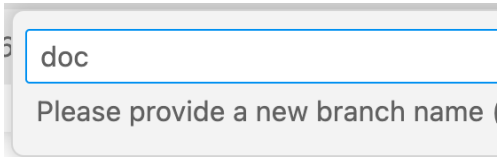
2. There should a popup appear at the top.
3. Choose *Git: Create Branch From …*



4. Select *origin/main Remote branch …*



5. Specify *doc* as branch name and press Enter.



6. Have a look at the lower left corner.

7. Have a look at *README.md* there should be no modification.
8. Make the modification again:

```
## About this API
```

9. Commit/Push with *docs*.

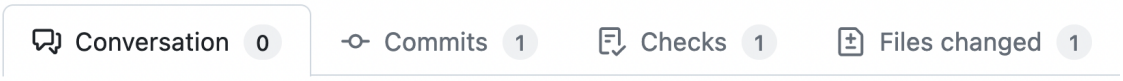
10.2. Pull Requests

Exercise: Pull Request

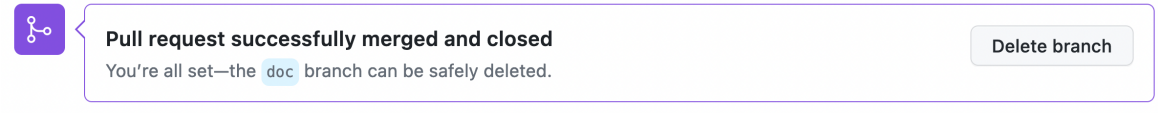
1. Go to the Repository at github. You can see there are now two branches.

2. Click on *Compare & pull request*.

3. Click *on Create pull request*.
4. Look at the *Checks* and *changed Files*.



5. Then click on *Merge Pull Request* and *Confirm Merge*.
6. Klick on *Delete branch*.

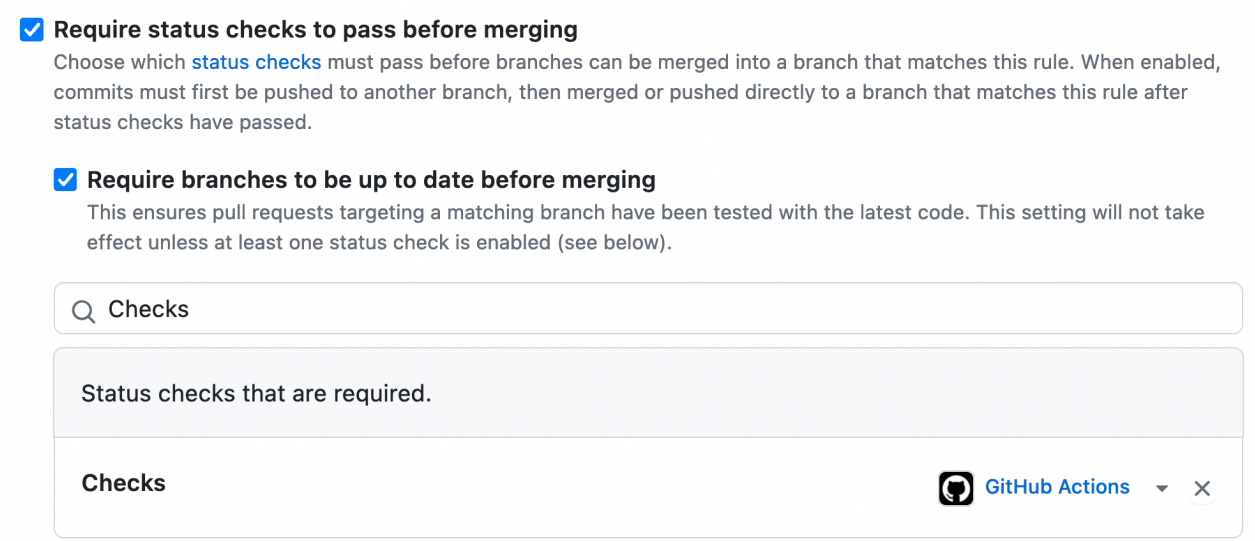


7. See if the *README.md* was changed.

10.3. Run Actions on Merge Request

Exercise: Settings

1. Got to the repository at *github.com*
2. Click on *Settings/Branches*
3. Edit rule for *main-Branch*.
4. Check additionally the box at *Require status checkes to pass before merging*. Then tick *Require branches to be up to date before merging* and select *Checks*.



5. Save.

Exercise: Change Trigger

1. Create a new branch *validate* of *origin/main*.
2. Edit *actions.xml* and change the trigger to *pull_request*.

```
name: OpenAPI Checks

on:
  - pull_request
```

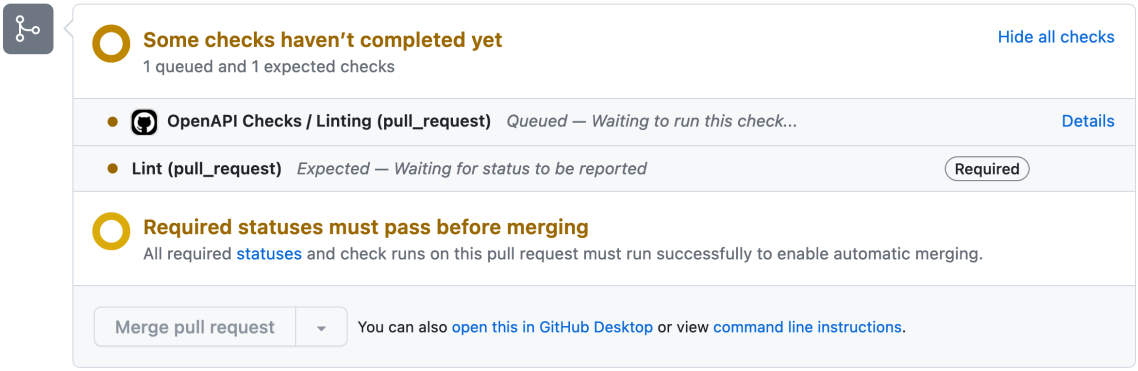
3. Commit/Push with name *validate*.
4. Create and merge a pull request on the github site.

Exercise: Test Validation

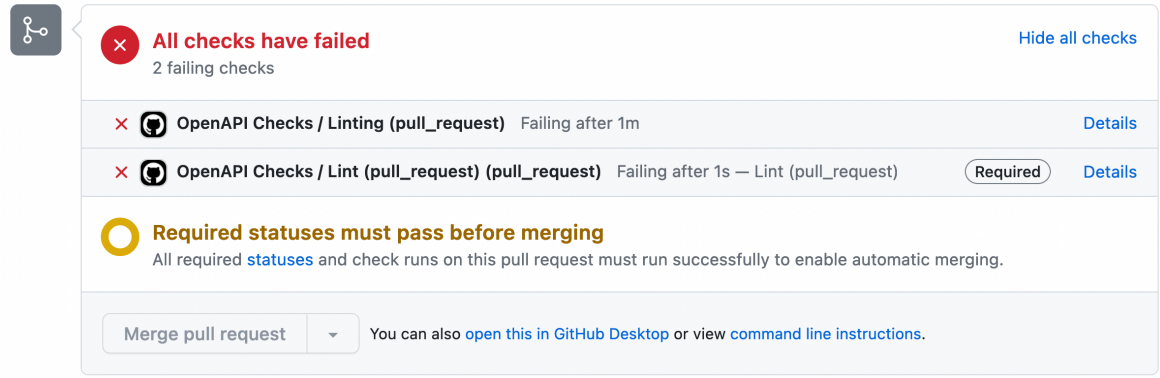
1. Create a new branch *operationid* from *origin/main*.
2. Change the fieldname of *operation**I**d* to *operation**i**d* with a lowercase **i**.

```
operationid: getQuote
```

3. Commit/push with *operation**i**d*.
4. Create a pull request at *github.com.*
5. Now we must wait till the status checks complete.



The checks should fail and the *main*-branch should remain untouched.



Exercise: Fix Branch

1. Correct the spelling of *operation**I**d*.
2. Push to the same branch.
3. Create and merge a pull request.

11. Security Checks

Exercise: Extend

1. Create a new branch from *origin/main* named *OWASP*.
2. Look at:

```
https://github.com/stoplightio/spectral-owasp-ruleset
```

Hint: Google github spectral owasp

3. Extend the file *.spectral.yml* with the OWASP ruleset.

```
extends:
  - spectral:oas
  - "@stoplight/spectral-owasp-ruleset"
```

Exercise: Run OWASP Ruleset local (Optional)

1. Install the ruleset:

```
npm install --save -D @stoplight/spectral-owasp-ruleset
```

2. Run the following command in a terminal:

```
spectral lint rfq-api-v1.oas.yml
```

3. Only display errors.

```
spectral lint rfq-api-v1.oas.yml -D error
```

Exercise: Adopting the github Workflow

1. Add the dependency for the OWASP ruleset.

```yaml
name: OpenAPI Checks

on:
  - push

jobs:
  Checks:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - run: npm install -g @stoplight/spectral-cli
      - run: npm install --save -D @stoplight/spectral-owasp-ruleset
      - run: spectral lint rfq-api-v1.oas.yml -D error
```

2. Commit the changes, push, and create a pull request named *OWASP*.
3. Look at the output of the Action.
4. Have a closer look at the error messages.

Annotations
10 errors

Checks: rfq-api-v1.oas.yml#L14
`owasp:api2:2019-protection-global-unsafe This operation is not protected by any security scheme.`

Checks: rfq-api-v1.oas.yml#L25
`owasp:api4:2019-rate-limit All 2XX and 4XX responses should define rate limiting headers.`

Checks: rfq-api-v1.oas.yml#L33
`owasp:api4:2019-integer-format Schema of type integer must specify format (int32 or int64).`

Checks: rfq-api-v1.oas.yml#L33
`owasp:api4:2019-integer-limit-legacy Schema of type integer must specify minimum and maximum.`

Checks: rfq-api-v1.oas.yml#L35
`owasp:api4:2019-string-limit Schema of type string must specify maxLength, enum, or const.`

Checks: rfq-api-v1.oas.yml#L35
`owasp:api4:2019-string-restricted Schema of type string must specify a format, pattern, enum, or const.`

Checks: rfq-api-v1.oas.yml#L37
`owasp:api4:2019-integer-format Schema of type integer must specify format (int32 or int64).`

Checks: rfq-api-v1.oas.yml#L37
`owasp:api4:2019-integer-limit-legacy Schema of type integer must specify minimum and maximum.`

Checks: rfq-api-v1.oas.yml#L41
`owasp:api4:2019-string-limit Schema of type string must specify maxLength, enum, or const.`

Checks: rfq-api-v1.oas.yml#L41
`owasp:api4:2019-string-restricted Schema of type string must specify a format, pattern, enum, or const.`

Question: How many errors are there?

11.1. Making the OpenAPI more secure

Hint: You'll find the final OpenAPI in *templates* with the name *rfq-api-v1-owasp.yml*

Exercise: Fixing

1. Stay in the *OWASP* branch.
2. Edit the file *rfq-api-v1.oas.yml*.
3. Change:

```
        id:
          type: integer
```

to:

```
        id:
          type: string
          format: uuid
```

4. Lint again, if you have a local installation of Spectral.
5. How many errors did you get?
6. Add restrictions to the fields of the Quotation:

```
      properties:
        id:
          type: string
          format: uuid
        product:
          type: string
          maxLength: 20
        quantity:
          type: integer
          format: int32
          minimum: 0
          maximum: 1000
        price:
          type: number
          minimum: 0
        shipment:
          type: string
        mode:
          type: string
```

7. Lint again, if you have a local installation of Spectral.

8. Specify the shipment terms:

```yaml
shipment:
  type: string
  enum:
    - FOB
    - DAT
```

9. Add a pattern for mode.

```yaml
mode:
  type: string
  pattern: "[A-Z]{3}"
```

10. Lint.
11. **Question:** What is still wrong with mode?
12. Add *maxLength*.

```yaml
mode:
  type: string
  pattern: "[A-Z]{3}"
  maxLength: 3
```

13. Lint again.
14. Add a pattern to product.

```yaml
product:
  type: string
  maxLength: 20
  pattern: '(\w+\W*)*'
```

15. Lint again.

16. Now the following errors should be left.

```
owasp:api2:2019-protection-global-unsafe
This operation is not protected by any security scheme.
paths./rfqs.post

owasp:api4:2019-rate-limit
All 2XX and 4XX responses should define rate limiting headers.
paths./rfqs.post.responses[200]

error  owasp:api4:2019-string-limit
Schema of type string must specify maxLength, enum, or const.
components.schemas.Quotation.properties.id
```

Exercise: Customize Ruleset

1. Add the following to the *.spectral.yml*:

```
extends:
  - spectral:oas
  - "@stoplight/spectral-owasp-ruleset"

rules:
  "owasp:api4:2019-rate-limit": warn
```

2. Lint again.
3. Change also the level to *warn* for the following rule:

```
"owasp:api2:2019-protection-global-unsafe"
```

Hint: The following exercise can be skipped by setting the level to warn.

Exercise: Customize Rule

1. Go to:

https://github.com/stoplightio/spectral-owasp-ruleset

2. Look at the file */src/*ruleset.ts.
3. Search for: *2019-string-limit*
4. Copy the rule from the following file in the *template* folder:

`string-restricted-rule.yml`

5. Paste it into your Spectral rules. The *.spectral.yml* should now look like this.

```
extends:
  - spectral:oas
  - "@stoplight/spectral-owasp-ruleset"

rules:
  "owasp:api4:2019-rate-limit": warn
  "owasp:api2:2019-protection-global-unsafe": warn
  "owasp:api4:2019-string-limit":
    message: "Schema of type string must specify maxLength!"
    description: "String size should be limited to mitigate resource exhaustion attacks.
This can be done using `maxLength`."
    severity: error
    given: '$..[?(@ && @.type=="string")]'
    then:
      function: schema
      functionOptions:
        schema:
          type: object
          oneOf:
            - required: ["maxLength"]
            - required: ["enum"]
            - required: ["const"]
```

6. Run the linter and look at the result.

7. There should be one error left:

```
owasp:api4:2019-string-limit
Schema of type string must specify maxLength!
components.schemas.Quotation.properties.id
```

8. Add the option for format.

```
    schema:
      type: object
      oneOf:
        - required: ["maxLength"]
        - required: ["enum"]
        - required: ["const"]
        - required: ["format"]
          properties:
            format:
              type: string
              enum: ["uuid","date","date-time"]
```

9. Run the linter locally and look at the result or run it at github.
10. Commit/push with *uuid*.
11. Look if the Action is green.
12. Finally merge the *OWASP* pull-request.

11.2. Additional Properties (Optional)

Exercise: Additional Properties Rule

1. Add the rule from the file *additional-properties-rule.yml* to your *.spectral.yml*. Do not forget to create a new branch first.

```
additional-properties:
  description: Additional properties must be forbidden in every object
  given: '$..[?(@ && @.type=="object")]'
  severity: error
  then:
    function: schema
    functionOptions:
      schema:
        type: object
        required:
          - additionalProperties
        properties:
          additionalProperties:
            const: false
```

12. Samples (Optional)

Exercise: Add Samples

1. Create new branch from *origin/main* with name *examples*.
2. Add examples:

```
product:
  type: string
  maxLength: 20
  pattern: '(\w+\W*)*'
  example: Firecracker
quantity:
  type: integer
  format: int32
  minimum: 0
  maximum: 1000
price:
  type: number
  minimum: 0
  example: 1.99
shipment:
  type: string
  enum:
    - FOB
    - DAT
mode:
  type: string
  pattern: "[A-Z]{3}"
  maxLength: 3
```

3. Commit/push and merge.

13. Server Implementation (Optional)

Exercise: Project Setup

1. Check out:

`https://github.com/predic8/rfq-server`

2. Open the project in your IDE.
3. Have a look at:

https://github.com/OpenAPITools/openapi-generator

4. Can you find your programming language?
5. Have a look at *generate.sh* file in the *rfq-server* project.

```
openapi-generator generate -g spring  -i
https://raw.githubusercontent.com/predic8/rfq-api/main/rfq-api-v1.oas.yml  \
 --additional-properties=interfaceOnly=true;artifactId=rfq-server \
 -o .
```

6. Note where the OpenAPI comes from.
7. Run *generate.sh*.

Exercise: Inspect generated Code: Model

1. Have a look at:

```
org.openapitools.model.Quotation
```

2. Look how the properties are generated from the OpenAPI:

id:

```
        id:
          type: string
          format: uuid
```

```
@Valid
@Schema(name = "id", requiredMode = Schema.RequiredMode.NOT_REQUIRED)
public UUID getId() {
  return id;
}
```

product:

```
        product:
          type: string
          maxLength: 20
          pattern: '(\w+\W*)*'
```

```
@Pattern(regexp = "(\\w+\\W*)*") @Size(max = 20)
@Schema(name = "article", example = "Firecracker", requiredMode =
Schema.RequiredMode.NOT_REQUIRED)
public String getArticle() {
  return article;
}
```

quantity:

```yaml
        quantity:
          type: integer
          format: int32
          minimum: 0
          maximum: 1000
```

```
@Min(0) @Max(1000)
@Schema(name = "quantity", requiredMode = Schema.RequiredMode.NOT_REQUIRED)
public Integer getQuantity() {
  return quantity;
}
```

shipment:

```yaml
        shipment:
          type: string
          enum:
            - FOB
            - DAT
```

```
public enum ShipmentEnum {
  FOB("FOB"),

  DAT("DAT");
…
```

```
@Schema(name = "shipment", requiredMode = Schema.RequiredMode.NOT_REQUIRED)
public ShipmentEnum getShipment() {
  return shipment;
}
```

Exercise: Inspect generated Code: API and Controller

1. Have a look at *RfqsApi* and *RfqsApiController* classes.

Exercise: Test Server

1. Open:

http://localhost:8080/swagger-ui/index.html

2. Send a Post request using Swagger UI and have a look at the response code.

13.1. Implement (Optional)

Exercise: Implement Server

1. Overwrite *getQuote* in *api.RfqsApiController*:

```
@Override
public ResponseEntity<Void> getQuote(Quotation quotation)  {
    System.out.println("quotation = " + quotation);
    try {
        return ResponseEntity.created(new URI("/rfqs/45324")).build();
    } catch (URISyntaxException e) {
        throw new RuntimeException(e);
    }
}
```

2. Run *generate.sh* again.
3. Look at *RfqsApiController* again.
4. Look at the option *interfaceOnly* in *generator.sh*.
5. Look at *.openapi-generator-ignore*.

Exercise: Test Server Implementation

1. Open:

http://localhost:8080/swagger-ui/index.html

2. Send Post using Swagger UI and have a look at the response code.
3. Have a look at the console output of the server in your IDE.

Exercise: Change Interface and run the OpenAPI Generator again (Optional)

1. Create a new branch named *rename*.
2. Edit the OpenAPI in the *rfq-api* project. Change the name of the property *product* into *article*.
3. Push and Merge.
4. Make sure the change is visible on github.
5. Run *generate.sh* again in the *rfq-server* project.
6. Build and run the project.
7. Send a POST again and check if it is still working.

Exercise: Commit and Push the Server (Instructor)

1. Look at *actions.yml* in the *rfq-server* project.
2. Commit & Push (no branching)
3. Look at:

https://github.com/predic8/rfq-server/actions

4. Have a look at the output of the build job.

Exercise: Azure Deployment (Instructor)

1. Log into Microsoft Azure cloud.
2. Look at *rfq-server* Overview.
3. Check *Last Deployment*
4. Open:

https://rfq-server.azurewebsites.net.

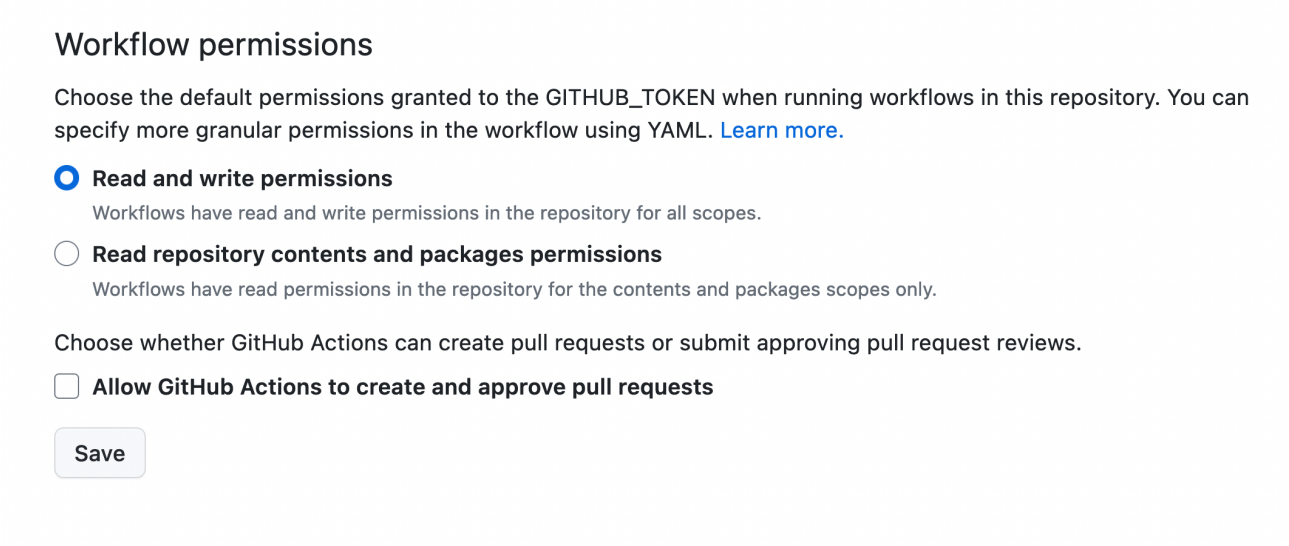
Maybe you must wait a minute or two.

5. Send POST and look at the status code.

13.2. Preparation(Instructor only)

When setting up the project from scratch check the settings in this section.

Give Actions write-Access to packages:



14. API Gateway

Exercise: Project

1. Checkout the project:

```
https://github.com/predic8/api-lifecycle-gateway
```

2. Have a look at all the files:
   - proxies.xml
     - Configuration from OpenAPI
     - Health endpoint
   - Dockerfile
     - Membrane Download
     - Add proxies.xml configuration.
     - Add OpenAPI from *rfq-api* project.
   - actions.yml
     - Dockerbuild with artifact
     - Azure deployment

3. Have a look at the server URL of the OpenAPI in the *rfq-api* project.

Exercise: Deployment

1. Make the *README.md* dirty.
2. Commit/Push
3. Have a look at the running actions at github.
4. Have a look at the *packages* registry at github.
5. Have a look at the output of *Deployment center* at Azure.

Exercise: Check Health and Test

1. Have a look at the health API in the *proxies.xml* file.
2. Call the endpoint:

```
https://api-lifecycle-gateway.azurewebsites.net/
```

3. Check if the result is OK.
4. Have a look at the deploy APIs.

https://api-lifecycle-gateway.azurewebsites.net/api-doc

5. Click on the *title* at the left side.
6. Submit a post.

15. API Diff

Exercise: OpenAPI Diff

1. Look at:

```
https://github.com/Tufin/oasdiff
```

2. Create new branch *diff* in the *rfq-api* project.
3. Make a copy of the OpenAPI file in the *rfq-api* project.

```
cp rfq-api-v1.oas.yml rfq-api-v1-ref.oas.yml
```

4. Change in *rfq-api-v1.oas.yml* the fieldname *product* to *article*.
5. Run:

```
oasdiff -base rfq-api-v1-ref.oas.yml -revision rfq-api-v1.oas.yml
```

6. Add the option for breaking changes.

```
oasdiff -check-breaking -base rfq-api-v1-ref.oas.yml -revision rfq-api-
v1.oas.yml
```

Exercise: Add OpenAPI Diff to the Pipeline

1. Modify the Pipeline:

```
jobs:
  Checks:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-go@v4
        with:
          go-version: '^1.20.3'
      - run: go install github.com/tufin/oasdiff@latest
      - run: |
          oasdiff -fail-on-diff -base rfq-api-v1-ref.oas.yml -revision rfq-api-
v1.oas.yml
      - uses: actions/setup-node@v3
      - run: npm install -g @stoplight/spectral-cli
      - run: npm install --save -D @stoplight/spectral-owasp-ruleset
      - run: spectral lint rfq-api-v1.oas.yml -D error
```

2. Commit and push with name *diff*.
3. Create pull request.
4. Look at the output of the pipeline.

```
components:
  schemas:
    modified:
      Quotation:
        properties:
          added:
            - article
          deleted:
            - product
```

Hint: Check if *oasdiff* returns a non-zero code in the case of an error. If you add the *-check-breaking* option, this might not be the case.

16. Troubleshooting

16.1. github

Passed checks are ignored.

Run Action again or close pull request and reopen it again.

No idea

Look at the github status:

https://www.githubstatus.com/

16.2. Azure

Logs

https://api-lifecycle-gateway.scm.azurewebsites.net/api/logs/docker