Useful technologies in software projects

Adam Petříček

Introduction

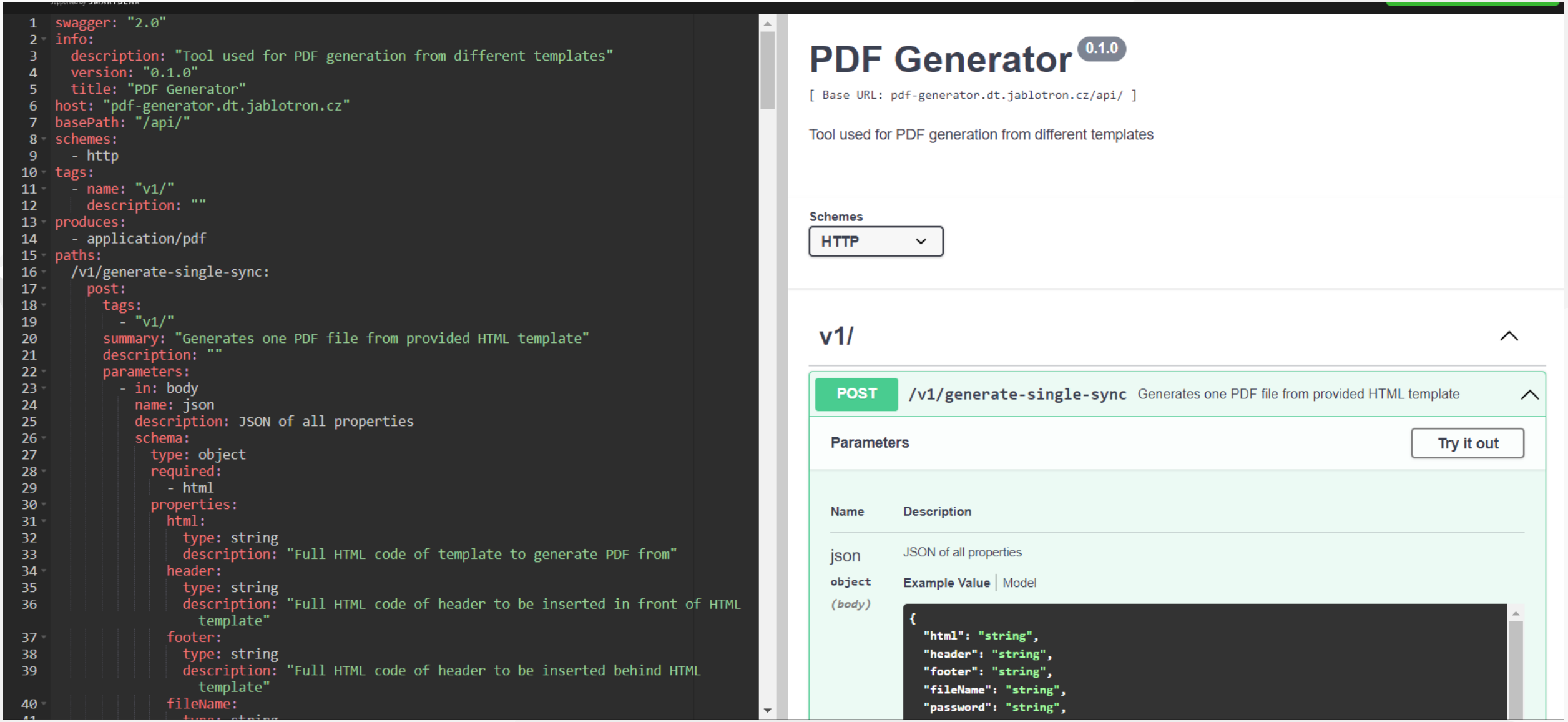
- not mandatory, but useful
- any software project
- enhancing best practices
- used to:
  - improve code quality
  - save time
  - eliminate mistakes
  - make collaborative projects easier

Code documentation

```php
/**
 * Sum the given two numbers and return the result
 * You can sum integers and floats
 *
 * @param int|float $num1
 * @param int|float $num2
 * @param mixed $third_param
 * @return int|float
 */
function sum($num1, $num2,$third_param)
{
    return $num1 + $num2;
```

- docs comments
  - can be used for generating wiki
  - code is more understandable by other people
  - necessity for good open-source projects

- API endpoint documentation — **Swagger**
  - code formatted as YAML / JSON
  - generates nicely looking API structure
  - endpoints can be tested


swagger: "2.0"
info:
description: "Tool used for PDF generation from different templates"
version: "0.1.0"
title: "PDF Generator"
host: "pdf-generator.dt.jablotron.cz"
basePath: "/api/"
schemes:
- http
tags:
- name: "v1/"
description: ""
produces:
- application/pdf
paths:
/v1/generate-single-sync:
post:
tags:
- "v1/"
summary: "Generates one PDF file from provided HTML template"
description: ""
parameters:
- in: body
name: json
description: JSON of all properties
schema:
type: object
required:
- html
properties:
html:
type: string
description: "Full HTML code of template to generate PDF from"
header:
type: string
description: "Full HTML code of header to be inserted in front of HTML template"
footer:
type: string
description: "Full HTML code of header to be inserted behind HTML template"
fileName:
PDF Generator
0.1.0
[Base URL: pdf-generator.dt.jablotron.cz/api/]
Tool used for PDF generation from different templates
Schemes
HTTP
v1/
POST
/v1/generate-single-sync
Generates one PDF file from provided HTML template
Parameters
Try it out
Name
Description
json
object
(body)
JSON of all properties
Example Value
Model
{
"html": "string",
"header": "string",
"footer": "string",
"fileName": "string",
"password": "string",

Static code analysis

- Prettier, Linters (language-specified)

1. **catching bugs** - Linters
   - no unused, undeclared variables
   - no class reassigning
   - force return in class getter
2. **code formatting** - Prettier
   - normalized number of spaces
   - limit characters in one line
   - keyword spacing

- code formatting useful in collaborative projects

```
if (condition1)
{
    //...
}else if (condition2) {
    //...
}  else
{
    //...
}


_____________________________


if (condition1) {
    //...
}
else if (condition2) {
    //...
}
else {
    //...
}
```

Code testing

- writing program that expects designated result for each case that can happen in your code
- example - **testing a function that moves robot by some distance forward**
  - 1. case: `distance = 10`
    - you expect robot to move by 10 units
  - 2. case: `distance = "hello"`
    - you expect program to throw exception
  - 3. case: `distance = 0`
    - you expect robot to stay in place
- if all cases match the expectations, tests succeed
- rewriting tests everytime you change your code
- unit tests, endpoint tests

Generated code coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #
All files	76.92	69.23	68.42	80.65	
src	100	100	100	100	
index.js	100	100	100	100	
src/application	66.67	50	33.33	80	
App.js	66.67	50	33.33	80	58
src/application/About	50	100	0	50	
index.js	50	100	0	50	4
src/application/Users	0	100	0	0	
index.js	0	100	0	0	3-4
src/design-system/Alert	0	100	0	0	
index.js	0	100	0	0	3
src/design-system/GlobalStyles	100	50	100	100	
index.js	100	50	100	100	21
src/design-system/Message	100	100	100	100	
index.js	100	100	100	100	
src/design-system/Toggle	100	83.33	100	100	
index.js	100	83.33	100	100	12
src/utilities	83.33	100	83.33	85.71	
addOne.js	0	100	0	0	1

Docker

- OS-level virtualization
- isolates software to containers - **containerization**
- used for having the same working environment on different machines

1. you create **dockerfile** (set of instructions) for the app where you specify all the needed files, libraries, etc.
2. on different machine, you create **docker container** based on the instructions - where your app will be running
3. you don't have to worry about mismatched versions of libraries, etc.

```dockerfile
FROM dregistry.jablotron.cz/docker-images/node:lts-alpine

RUN mkdir -p /opt/node-app

WORKDIR /opt/node-app

RUN apk add --no-cache \
    libstdc++=8.3.0-r0 \
    qpdf=8.4.2-r0 \
    libxrender \
    libxext \
    ca-certificates \
    fontconfig \
    ttf-dejavu \
    ttf-droid \
    && update-ms-fonts \
    && fc-cache -f \
    && rm -rf /tmp/* \
    && apk del .build-deps

COPY . .
RUN npm install

CMD ["npm", "run", "dev"]
```

Git CI

- Continuous Integration
- GitHub and GitLab each with their own implementation
- way to automate routine tasks like testing, builds or deployment with every **push**
- CI can be set to do different tasks depending on commit tag
- you can generate **artifacts** for download - results of CI (code coverage, exe files)
- previously mentioned technologies can be automated using CI

Git CI - practice example

- *setup*:

1. creating tests, static code analysis, containerization
2. adding these tasks to the CI

- *when you push to Git*:

1. if static code analysis or tests fail, commit gets rejected
2. when everything passes, app build starts (Docker)
3. after app is built, you can deploy it to production server automatically

```yaml
test:npm:
  tags:
    - docker
    - linux
  stage: test
  image: dregistry.jablotron.cz/it/pdf-generator:node-extended-latest
  services:
    - docker:stable-dind
  before_script:
    - chmod +x ./.gitlab-ci-dotenv-setup.sh
    - ./.gitlab-ci-dotenv-setup.sh
  script:
    - cp -f .env.testing .env
    - npm install
    - npm run test
    - npm run lint:nofix
  artifacts:
    expire_in: 1 week
    paths:
      - src/__coverage__/
```

```
build:production:
  rules:
    - if: '$CI_COMMIT_TAG !~ /^$/ && $CI_COMMIT_TAG !~ /^prebuild-/ && $CI_COMMIT_TAG !~ /^preview-/'
  tags:
    - docker
    - linux
  stage: build
  needs: ["test:audit", "test:npm"]
  image: docker:stable
  services:
    - docker:stable-dind
  before_script:
    - chmod +x ./.gitlab-ci-dotenv-setup.sh
    - ./.gitlab-ci-dotenv-setup.sh
    - rm -f .env.testing
  script:
    - echo "$CI_BUILD_TOKEN" | docker login -u gitlab-ci-token --password-stdin $CI_REGISTRY
    - docker build . -t $CI_REGISTRY/$CI_PROJECT_PATH:$CI_COMMIT_TAG --no-cache
    - docker push $CI_REGISTRY/$CI_PROJECT_PATH:$CI_COMMIT_TAG
    - docker tag $CI_REGISTRY/$CI_PROJECT_PATH:$CI_COMMIT_TAG $CI_REGISTRY/$CI_PROJECT_PATH:latest
    - docker push $CI_REGISTRY/$CI_PROJECT_PATH:latest
```

Any questions?

Questions

1. Which of the mentioned technologies seems the most useful to you? Why?
2. What other practices / technologies do you use for improving your software projects?
3. Do you believe that well written code needs comments? Why?
4. How would you describe some of your biggest bad habits while writing code? What can you do to get rid of them?

Thanks for you attention