# Heroku.com - запуск сайтов на node.js, php и тд.

## Порядок установки

1. Регистрируемся на heroku.com
2. Активируем письмо на е-мейле, вводим пароль
3. Скачиваем их клиент под ОС. Ссылка на скачивание должна быть на сайте
4. Проверить, что клиент установлен: в cmd пишем

heroku --help

1. Теперь в системе надо ввести логин и пароль, в установленное приложение heroku:

heroku login

## Запуск node.js через heroku

В **server.js** вместо порта создаем переменную, и начинаем ее слушать:

const port = process.env.PORT || 3000;

...

app.listen(port, () => {

console.log(`Server is up on port ${port}`);

});

В **package.json** в "scripts" дописываем:

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"start": "node server/server.js",

},

Под "scripts" дописываем engines с установленной версией ноды

"engines": {

"node": "7.5.0"

},

Теперь в cmd, из-под директории, в которой находится файл, можно запускать Node.js через скрипт:

npm start

Теперь можно создать соединение heroku. В cmd пишем:

heroku create

Это создаст удаленный репозиторий в heroku. После этого необходимо добавить все файлы, кроме папки node-modules в **git.**

**Важно!** heroku собирает проект на месте, потому у проекта должен быть 1 уровень вложенности (server.js должен быть сразу в папке проекта или нужно в "start" указать правильный путь к server.js) и нет необходимости пушить node-modules, т.к. heroku сразу заинсталлит их самостоятельно

**Важно!** Все модули должны быть заинсталлены с использованием --save (для того, что бы сохранились в dependencies)

После добавления в гит, запушим их в heroku:

git push heroku master

Heroku добавит файлы и заинсталлит модули. После этого, можно сразу открыть страничку в дефолтном браузере. Пишем

heroku open

Если будет ошибка, можно посмотреть, что именно произошло. Пишем

heroku logs

## Пишем свои переменные в окружение Heroku

Например есть **config.json** файл, в котором хранится переменная кличевого слова к заданию пароля при создания юзера (salt)

выглядит примерно так:

{

"test": {

"PORT": 3000,

"MONGODB\_URI": "mongodb://localhost:27017/TodoAppTest",

"JWT\_SECRET": "SLsNlvc23f1MNls23"

}

}

Так выглядит **config.js**:

const env = process.env.NODE\_ENV || 'development';

if(env === 'development' || env === 'test'){

// "require" automatically convert JSON to JS object

let config = require('./config.json');

let envConfig = config[env];

// make array from object

Object.keys(envConfig).forEach((key) =>{

process.env[key] = envConfig[key];

})

}

В Heroku переменная **process.env.NODE\_ENV === 'production'**

А в наших моделях везде используется ключевое слово **process.env.JWT\_SECRET** что бы сгенерировать или считать пароль, например в модели user.js

UserSchema.methods.generateAuthToken = function () {

let user = this;

let access = 'auth';

// userId already taken, auth - just word in DB,

// process.env.JWT\_SECRET - secret from config

let token = jwt.sign(

{\_id: user.\_id.toHexString(), access},

process.env.JWT\_SECRET

).toString();

user.tokens.push({access, token});

// save token in DB and return result to server.js to futher chaining (.then)

return user.save().then(() => {

return token;

})

};

Что бы такое приложение нормально работало на сервере heroku, нужно сохранить переменную JWT\_SECRET внутри Heroku

Получаем текущеие переменные:

heroku config

>>> MONGODB\_URI: mongodb://heroku\_db64tsc5:ol4cbvoj9a2kdodivl5664j44...

Задать переменную:

heroku config:set JWT\_SECRET=7JsnwDlnmb234Lsd34

Удалить переменную:

heroku config:unset JWT\_SECRET