**КТ № 3. NPM. Package.json. Работа с пакетами.**

**Время на выполнение практической работы:** 4 часа.

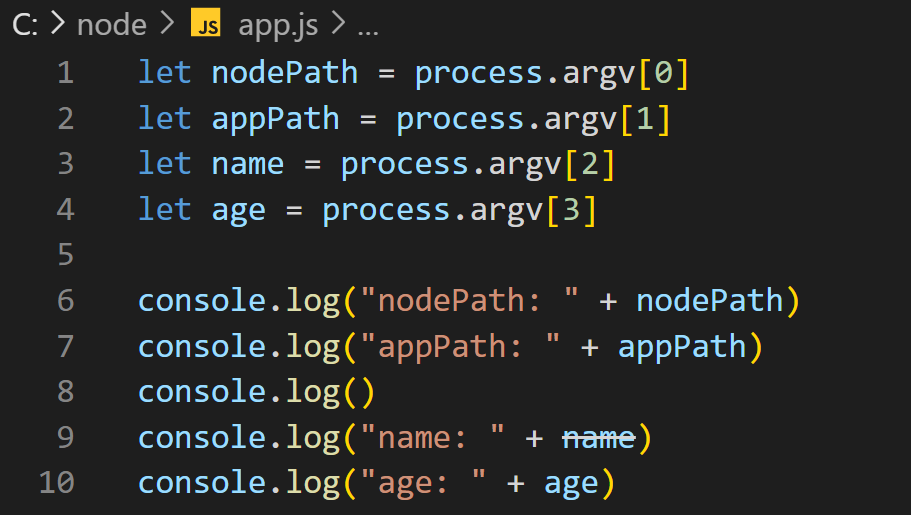
**Теоретические сведения к выполнению практической работы:**

**Передача параметров приложению.**

При запуске приложения из терминала/командной строки мы можем передавать ему параметры. Для получения параметров в коде приложения применяется массив process.argv. Это аналогично тому, как в языках C/C++/C#/Java в функцию main передается набор аргументов в виде строкового массива.

Первый элемент этого массива всегда указывает на путь к файлу node.exe, который вызывает приложение. Второй элемент массив всегда указывает на путь к файлу приложения, который выполняется.

К примеру, определим следующий файл app.js:

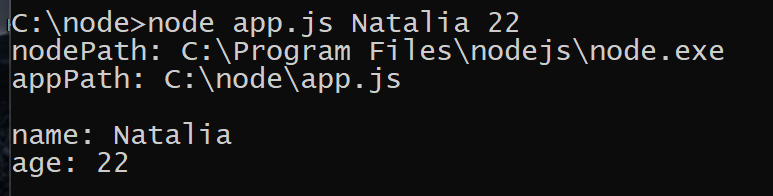


В данном случае мы ожидаем, что приложению будут переданы два параметра: name и age.

Теперь запустим приложение с помощью следующей команды:



В данном случае "Natalia" и "22" - это те значения, которые помещаются соответственно в process.argv[2] и process.argv[3]:



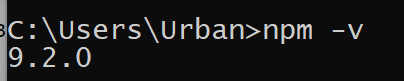
**NPM. Package.json. Установка модулей. Определение команд**

Кроме встроенных и кастомных модулей Node.js существует огромный пласт различных библиотек и фреймворков, разнообразных утилит, которые создаются сторонними производителями и которые также можно использовать в проекте, например, express, grunt, gulp и так далее. И они тоже нам доступны в рамках Node.js. Чтобы удобнее было работать со всеми сторонними решениями, они распространяются в виде пакетов. Пакет по сути представляет набор функциональностей.

Для автоматизации установки и обновления пакетов, как правило, применяются систему управления пакетами или менеджеры. Непосредственно в Node.js для этой цели используется пакетный менеджер NPM (Node Package Manager). NPM по умолчанию устанавливается вместе с Node.js, поэтому ничего доустанавливать не требуется. Но можно обновить установленную версию до самой последней. Для этого в командной строке/терминале надо запустить следующую команду:



Чтобы узнать текущую версию npm, в командной строке/терминале надо ввести следующую команду:



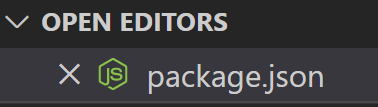
Для нас менеджер npm важен в том плане, что с его помощью легко управлять пакетами. К примеру, создадим на жестком диске новую папку modulesapp.

Если в дальнейшем нам больше не потребуется express, то мы его можем удалить следующей командой:

npm uninstall express

**Файл package.json**

Для более удобного управления конфигурацией и пакетами приложения в npm применяется файл конфигурации package.json. Так, добавим в папку проекта modulesapp новый файл package.json.



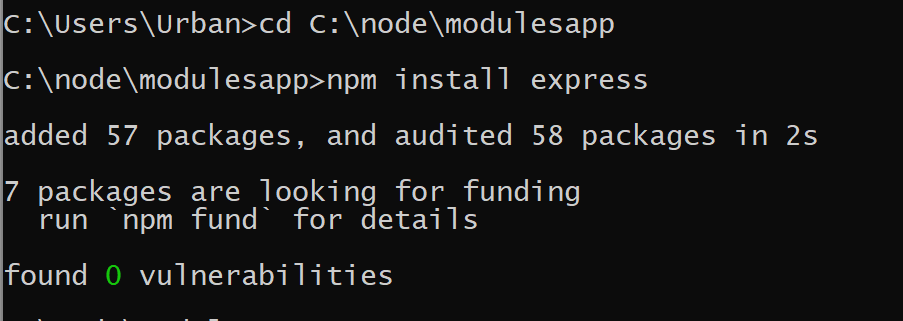


Здесь определены только две секции: имя проекта - modulesapp и его версия - 1.0.0. Это минимально необходимое определение файла package.json. Данный файл может включать гораздо больше секций. Подробнее можно посмотреть в документации.

Далее для примера установим в проект express. Express представляет легковесный веб-фреймворк для упрощения работы с Node.js. В данном случае мы не будем пока подробно рассматривать фреймворк Express, так как это отдельная большая тема. А используем его лишь для того, чтобы понять, как устанавливаются сторонние модули в проект.

Для установки функциональности Express в проект вначале перейдем к папке проекта с помощью команды cd. Затем введем команду





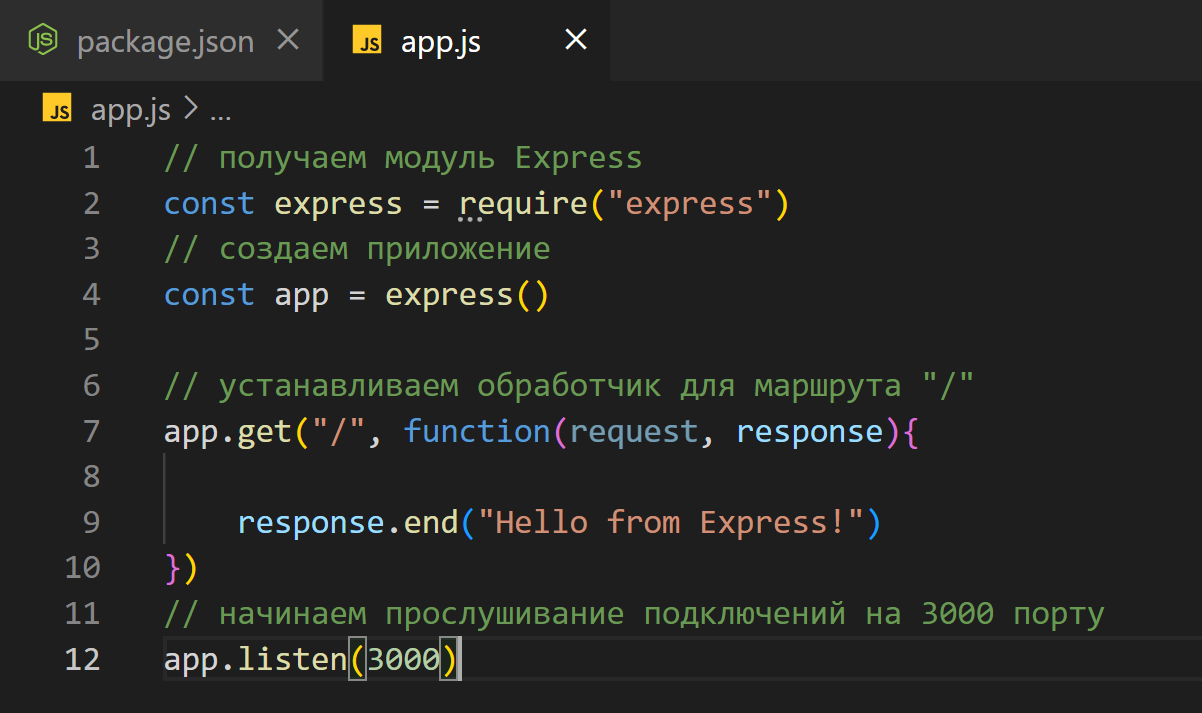
После установки express в папке проекта modulesapp появится подпапка node\_modules, в которой будут хранится все установленные внешние модули. В частности, в подкаталоге node\_modules/express будут располагаться файлы фреймворка Express.

И после выполнения команды, если мы откроем файл package.json, то мы увидим информацию о пакете.



Информация обо всех добавляемых пакетах, которые используются при работе приложения, добавляется в секцию dependencies.

Используем добавленный пакет express и для этого определим файл простейшего сервера. Для этого в корневую папку проекта modulesapp добавим новый файл app.js



Первая строка получает установленный модуль express, а вторая создает объект приложения.

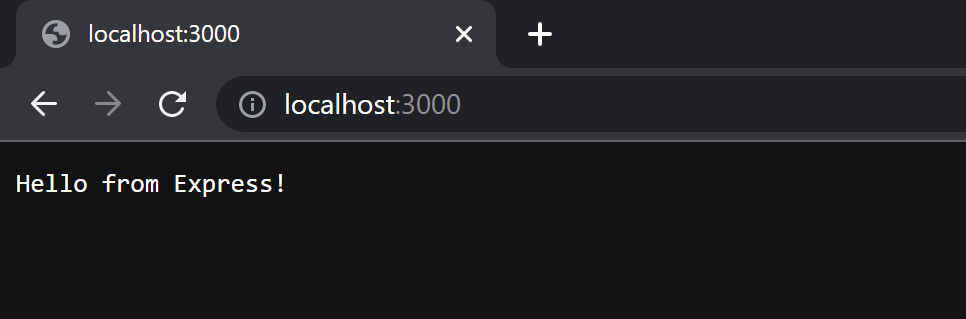
В Express мы можем связать обработку запросов с определенными маршрутами. Например, "/" - представляет главную страницу или корневой маршрут. Для обработки запроса вызывается функция app.get(). Первый параметр функции - маршрут, а второй - функция, которая будет обрабатывать запрос по этому маршруту.

И чтобы сервер начал прослушивать подключения, надо вызвать метод app.listen(), в который передается номер порта.

Запустим сервер командой node app.js



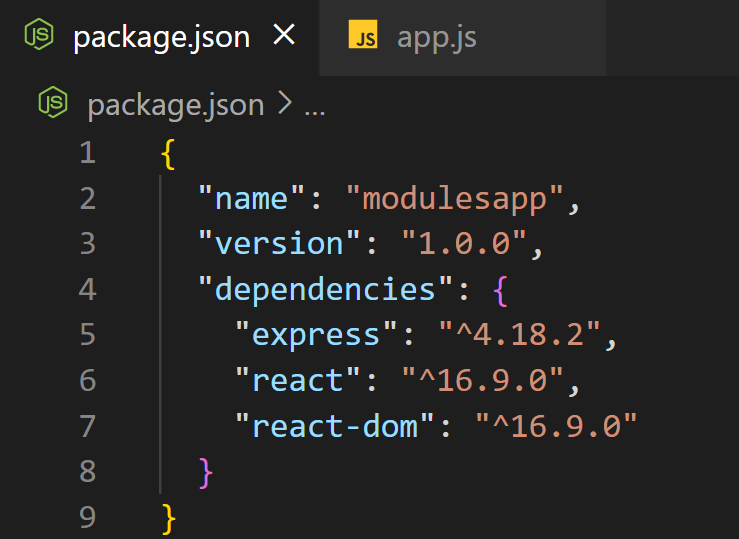
И в адресной строке браузера введем адрес <http://localhost:3000/>



**Добавление множества пакетов**

Файл package.json играет большую роль и может облегчить работу с пакетами в различных ситуациях. Например, мы планируем использовать множество пакетов. Но вводить для установки каждого пакета в консоли соответствующую команду не очень удобно. В этом случае мы можем определить все пакеты в файле package.json и потом одной командой их установить.

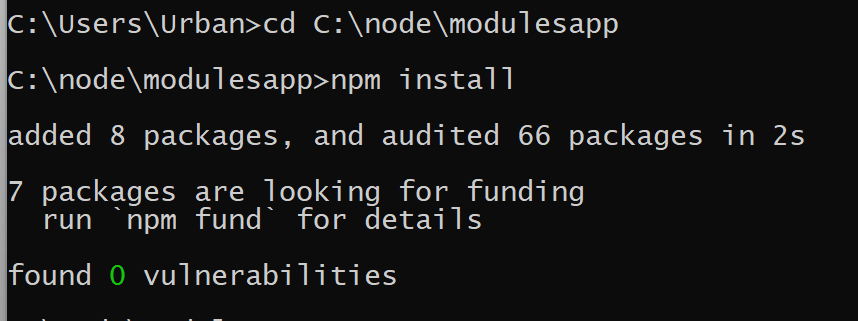
Например, изменим файл package.json следующим образом



Здесь добавлены определения двух пакетов, которые представляют библиотеку React.

Затем для загрузки всех пакетов выполнить команду





Эта команда возьмет определение всех пакетов из секций dependencies и загрузит их в проект. Если пакет с нужной версией уже есть проекте, как в данном случае express, то по новой он не загружается.

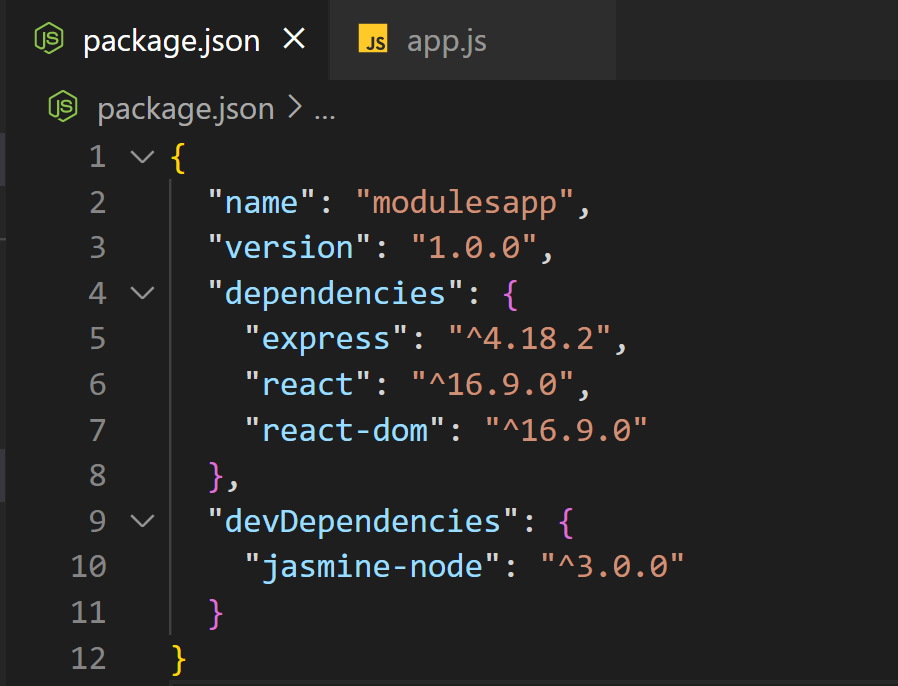
**devDependencies**

Кроме пакетов, которые применяются в приложении, когда оно запущено и находится в рабочем состояни, например, express, то есть в состоянии "production", есть еще пакеты, которые применяются при разработке приложения и его тестировании. Такие пакеты, как правило, добавляются в другую секцию - devDependencies.

Например, загрузим в проект пакет jasmine-node, который используется для тестирования приложения



Флаг --save-dev указывается, что информацию о пакете следует сохранить именно в секции devDependencies файла package.json



**Удаление пакетов**

Для удаления пакетов используется команда npm uninstall. Например:



При этом не важно, где располагается информация о пакете - в секции dependencies или devDependencies, пакет удаляется из любой из этих секций.

Если нам надо удалить не один пакет, а несколько, то мы можем удалить их определение из файла package.json и ввести команду npm install, и удаленые из package.js пакеты также будут удалены из папки node\_modules.

Например, изменим файл package.json следующим образом



Здесь больше нет определения никаких пакетов. И введем команду



Причем мы также можем одновременно некоторые пакеты добавлять в package.json, а некоторые, наоборот, удалять. И при выполнении команды npm install пакетный менеджер новые пакеты установит, а удаленные из package.json пакеты удалит.

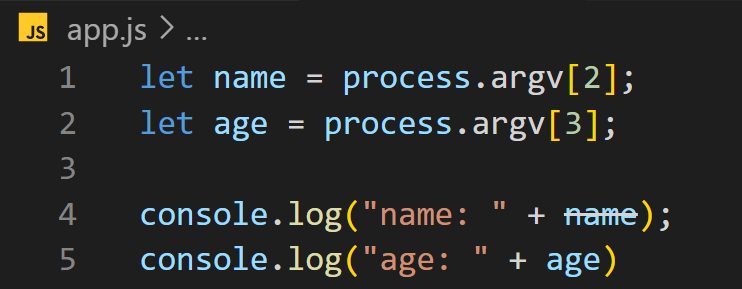
**Семантическое версионирование**

При определении версии пакета применяется семантическое версионирование. Номер версии, как правило, задается в следующем формате "major.minor.patch". Если в приложении или пакете обнаружен какой-то баг и он исправляется, то увеличивается на единицу число "patch". Если в пакет добавляется какая-то новая функциональность, которая совместима с предыдущей версией пакета, то это небольшое изменение, и увеличивается число "minor". Если же в пакет вносятся какие-то большие изменения, которые несовместимы с предыдущей версией, то увеличивается число "major". То есть глядя на разные версии пакетов, мы можем предположить, насколько велики в них различия.

В примере с express версия пакета содержала, кроме того, дополнительный символ карет: "^4.14.0". Этот символ означает, что при установке пакета в проект с помощью команды npm install будет устанавливаться последняя доступная версия от 4.14.0. Фактически это будет последняя доступная версия в промежутке от 4.14.0 до 5.0.0 (>=4.14.0 и <5.0.0).

**Команды npm**

NPM позволяет определять в файле package.json команды, которые выполняют определенные действия. Например, определим следующий файл app.js



В данном случае мы получаем переданные при запуске приложению параметры.

И определим следующий файл package.json



Здесь добавлена секция scripts, которая определяет две команды. Вообще команд может быть много в соответствии с целями и задачами разработчика.

Первая команда называется start. Она по сути выполняет команду node app.js, которая выполняет код в файле app.js

Вторая команда назвывается dev. Она также выполняет тот же файл, но при этом также передает ему два параметра.

Названия команд могут быть произвольными. Но здесь надо учитывать один момент. Есть условно говоря есть зарезервированные названия для команд, например, start, test, run и ряд других. Их не очень много. И как раз первая команда из выше определенного файла package.json называется start. И для выполнения подобных команд в терминале/командной строке надо выполнить команду

npm [название\_команды]

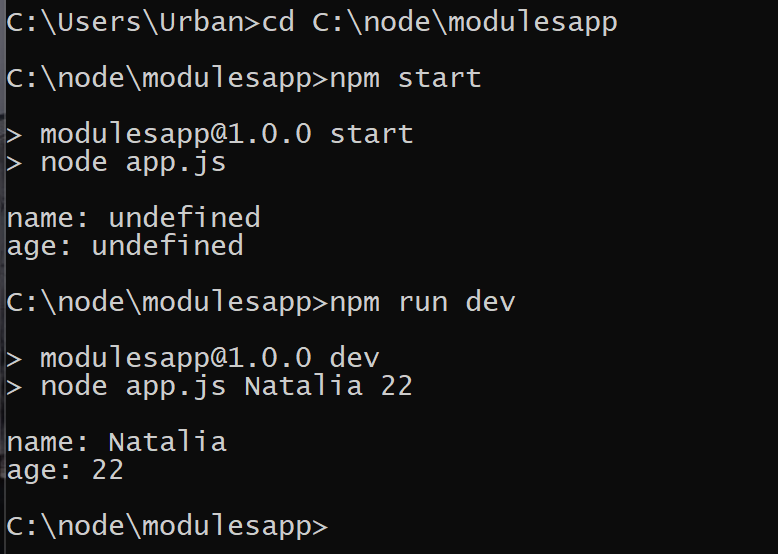
Например, для запуска команды start

npm start

Команды с остальными названия, как например, "dev" в вышеопределенном файле, запускаются так:

npm run [название\_команды]

Например, последовательно выполним обе команды



**Этапы выполнения:**

1. ***Передайте приложению параметры, например имя и возраст (или любые другие на ваше усмотрение);***
2. ***Установите последнюю версию пакетного менеджера и зафиксируйте результат (информацию о текущей версии npm);***
3. ***Создайте проект и добавьте в него файл*** ***package.json;***
4. ***Установите фреймворк express;***
5. ***Зафиксируйте изменения, произошедшие в файле package.json;***
6. ***Добавьте в проект файл app.js для определения простейшего сервера;***
7. ***Воспользуйтесь методом app.listen() для того, чтобы сервер начал прослушивать подключения;***
8. ***В адресной строке браузера введите адрес*** [***http://localhost:3000/***](http://localhost:3000/) ***и зафиксируйте результат скриншотом;***
9. ***Добавьте несколько пакетов, например пакеты, представляющие библиотеку React;***
10. ***Добавьте пакет, используемый для тестирования приложения, в секцию devDependencies;***
11. ***Удалите пакет express;***
12. ***Удалите несколько пакетов;***
13. ***Создайте команды в файле package.json для получения переданных при запуске приложения параметров.***