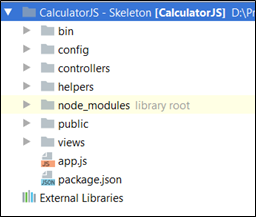
**Lab: Calculator using Javascript and ExpressJS**

Този документ дефинира пълен пътеводител за създаването на приложение за калкулатор с Express.js Framework, от създаването на рамката до внедряването на моделите, изгледите и контролерите, необходими за функционирането на приложението ни.  
Уверете се, че вече сте преминали през ръководството за начинаещи: JavaScript. В това ръководство ще използваме: WebStorm. Можете да изтеглите скелета на калкулатора от тук. Останалата част от необходимия софтуер, който не е задължителен, е описан в ръководството по-горе.

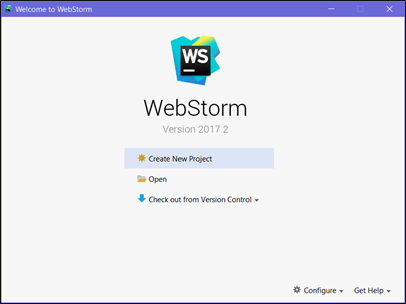
# I. Base Project Overview

Node.js е платформа, написана в JavaScript и осигурява функционалност отзад. Експрес е модул (за момента можем да свързваме модула като клас, който осигурява известна функционалност), който обвива Node.js по начин, който прави кодирането по-бързо и по-лесно и е подходящ за архитектурата на MVC.  
Първоначално проектът е със следната структура:

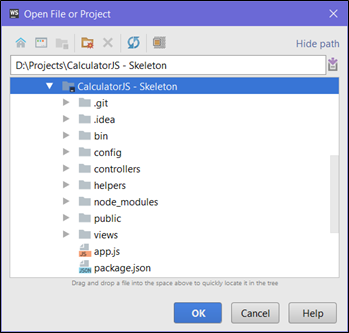


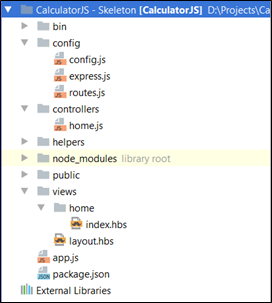
Тук можем да видим няколко папки. Нека да ги разгледаме един по един и да видим за какво са:  
1. bin - Съдържа един файл, наречен www, който е отправна точка на нашата програма. Самият файл съдържа известна логика за конфигуриране, необходима за локално изпълнение на сървъра.  
2. node\_modules (корен на библиотеката) - Що се отнася до името ни, в тази папка поставяме всяка библиотека (модул), от която зависи проектът ни.  
3. public(публично) - Всичко в нашата публична папка (файлове, изображения и т.н.) ще бъде достъпно за всеки потребител. Ще поговорим по-късно.  
4. routes (маршрути) - папка, в която ще поставим нашата конфигурация на маршрута. Ще разберем какъв е маршрутът.  
5. views (изгледи)- Там ще запазим вижданията за нашия модел. Отново ще използваме шаблони с помощта на двигателя за преглед на дръжките.  
6. app.js - Скриптът съдържа логиката, необходима за стартиране на сървъра.  
7. package.json - файл, съдържащ информация за проекта (като името на проекта, лиценза и т.н.). Най-важното е, че има част "зависимости", в която ще се намират всички имена и версии на всеки модул, използван от нашите проекти.  
Да видим какво имаме в скелета, преди да започнем да работим по него:

## 1. Open the Project



Нека да продължим напред и да заредим скелета. Кликнете върху "Отваряне" и намерете изтегления и разкопчания проект на скелет:

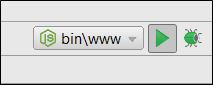




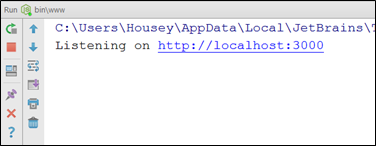
Това е нашият проект Node.js. В предишните стъпки описахме как се стигнахме дотук. Сега нека да поговорим за Node.JS:  
Както знаем, това е платформа, написана на Javascript, осигуряваща функционалност отзад. Това ни дава много гъвкавост, тъй като в нашия front-end обикновено се използва JavaScript. Това прави комуникацията по-лесна.  
Освен това, Node.JS е бърз, защото използва C ++ зад кулисите, както и защото е способен да извършва асинхронни повиквания. Тя използва системата за събития.  
Резюме: Изтеглехме проекта и сме готови да започнем да пишем код!

## 2. Run the Project

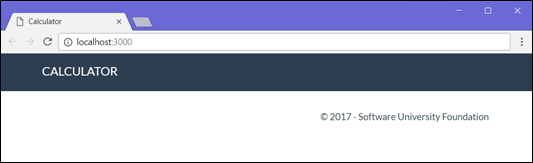
След като отворихме проекта, нека се опитаме да го изпълним, за да можем да видим с какво работим. Отидете в горната дясна част на WebStorm, където ще намерите бутон Run, който изглежда като зелен бутон за възпроизвеждане:



Ето как ще стартираме Express приложение. Отидете напред и кликнете върху бутона. Ако всичко върви според плана, трябва да видим това съобщение в конзолата:



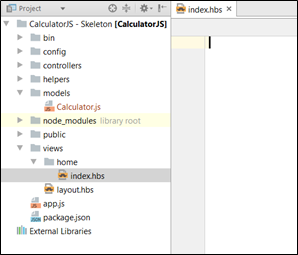
Сега можем да отворим страницата и да видим какво имаме:



# II. Implementing Functionality

## 1. Create Calculator View

Преди да можем да имаме някаква функционалност, би било хубаво да имаме представа за това, срещу което работим, така че нека продължим и да създадем формуляр, който потребителят ще използва за изчисления:  
Отидете в изгледите / началната папка и отворете файла index.hbs:



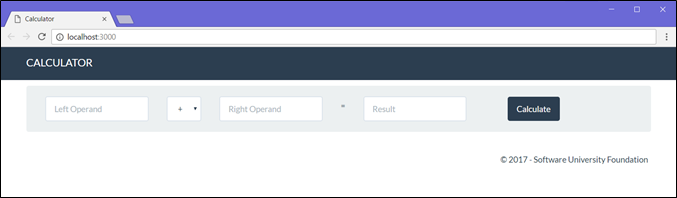
Празно е?! Как се виждат горните и долните колони, показани по-горе? Отговорът е, че използваме глобален файл с оформление, така че не е нужно да копирате и поставяме нашето оформление на страницата във всеки един изглед в нашия проект (който може да има десетки или стотици прегледи). Целият HTML дизайн е вътре в layout.hbs. Няма да се докосваме до това, така че нека да отидем до файла index.hbs и да добавим формуляра ни:

<**div class="container body-content span=8 offset=2"**>  
 <**div class="well"**>  
 <**form class="form-inline" action="/" method="POST"**>  
 <**fieldset**>  
 <**div class="form-group"**>  
 <**div class="col-sm-1 "**>  
 <**input type="text" class="form-control" id="leftOperand" placeholder="Left Operand"**  
 **name="calculator[leftOperand]" value="**{{**calculator**.**leftOperand**}}**"**>  
 </**div**>  
 </**div**>  
   
 <**div class="form-group"**>  
 <**div class="col-sm-4 "**>  
 <**select class="form-control" name="calculator[operator]"**>  
 <**option value="+"**{{**selectif calculator**.**operator '+'**}}>+</**option**>  
 <**option value="-"**{{**selectif calculator**.**operator '-'**}}>-</**option**>  
 <**option value="\*"**{{**selectif calculator**.**operator '\*'**}}>\*</**option**>

<**option value="/"**{{**selectif calculator**.**operator '/'**}}>/</**option**>  
 </**select**>  
 </**div**>  
 </**div**>  
   
 <**div class="form-group"**>  
 <**div class="col-sm-4 "**>  
 <**input type="text" class="form-control" id="rightOperand" placeholder="Right Operand"**  
 **name="calculator[rightOperand]" value="**{{**calculator**.**rightOperand**}}**"**>  
 </**div**>  
 </**div**>  
   
 <**div class="form-group"**>  
 <**div class="col-sm-2 "**>  
 <**p**>=</**p**>  
 </**div**>  
 </**div**>

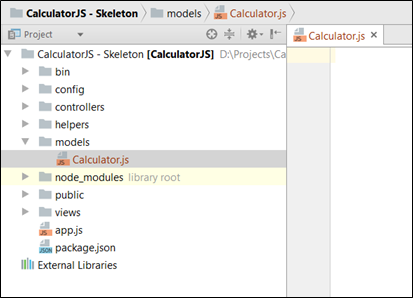
<**div class="form-group"**>  
 <**div class="col-sm-4 "**>  
 <**input type="text" class="form-control" id="result" placeholder="Result"**  
 **name="result" value="**{{**result**}}**"**>  
 </**div**>  
 </**div**>  
   
 <**div class="form-group"**>  
 <**div class="col-sm-4 col-sm-offset-4"**>  
 <**button type="submit" class="btn btn-primary"**>Calculate</**button**>  
 </**div**>  
 </**div**>  
 </**fieldset**>  
 </**form**>  
 </**div**>  
 </**div**>

Ще запазим състоянието на операндите и оператора за лесна употреба, така че синтаксисът на кормилото, който виждате тук, прави точно това. Помощникът {{selectif}} е малко по-специален: избира оператора от падащия списък въз основа на последния използван оператор. Ще видим как това се изпълнява малко по-късно. Засега нека да разгледаме нашето уеб приложение на http: // localhost: 3000 и да видим как се справяме:

Изглежда добре! Освен че не прави нищо. Първо, да стигнем до създаването на нещо, което ще държи данните ни: модела.

## 2. Create Calculator Model

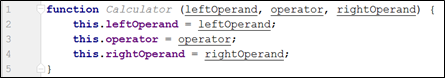
Време е да проектираме нашия основен модел - Калкулаторът. Той ще съдържа следните свойства:  
• leftOperand  
• оператор  
• rightOperand  
Нека да създадем нашия модел по начина JavaScript. Тъй като не използваме база данни в това упражнение, ние просто ще определим калкулатора като прост клас на JavaScript.  
Създайте папка, наречена "модели" и вътре в нея, създайте файл с JavaScript, наречен "Calculator.js":



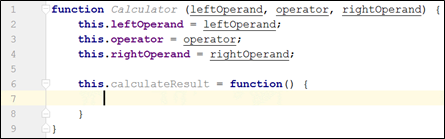
Сега, нека да напишем кода, който ще определи нашия модел Калкулатор:  
1. Създайте функция, която ще бъде извикана при създаването на нов калкулатор:



2. Вътре, дефинирайте свойствата на калкулатора:

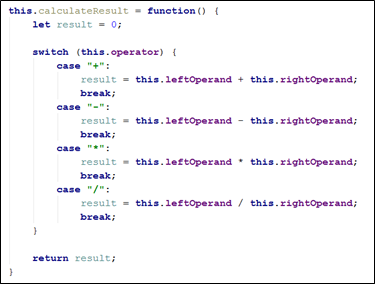


3. Създайте функция за изчисляване на резултата от свойствата:

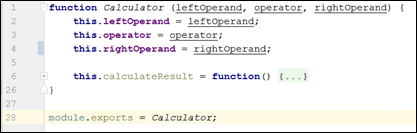


Вътре в тази функция ще напишем логиката, която е необходима за изчисляване на резултата от операндите и оператора. Нека да създадем логиката, необходима за това:

4. Напишете изчислителната логика:



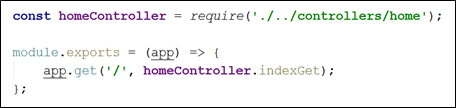
5. Почти готово: експортирайте нашия модел, за да може той да бъде видим за външния свят (извън файла Calculator.js):



6. Резюме: Сега знаем как да създадем прост модел и да го направим видим за външния свят.

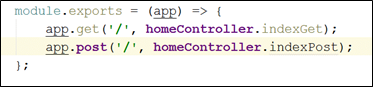
## 3. Add a Route, which Calls the Controller Action

Що се отнася до маршрута - с ExpressJS, цялата ни логика на маршрутизиране обикновено се намира в файл, наречен routes.js. Ето какво изглежда понастоящем:

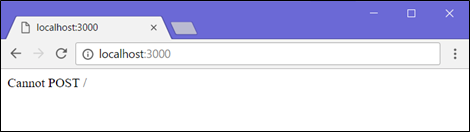


Нека разбием този код на разбираеми парчета:  
• const homeController = изисква ('./../ контролери / начало')  
Този кодиран код внася логиката на нашия контролер. Преди да можем да се обадим на някакви методи, трябва да знаем, че тези методи съществуват, нали?  
• module.exports  
Това е част от кода, който взима кода в него и го излага на външния свят. Тук въвеждаме кода си, защото Node.JS трябва да има достъп до него, така че може да изпълни действието, когато потребителят се обади на посочения маршрут (примери: / изчисляване, / редактиране / 2, / влизане).  
• app.get ('/', homeController.indexGet)

Тази част от кода казва на Node.JS да изслушва всички заявки GET на маршрута "/" и когато намери това - за да изпълни метода homeController.indexGet (затова е необходимо да импортираме домейн контролера там, ).  
Сега, когато разбрахме как работи маршрутизиращата система в Express.JS, е време да добавим нашия собствен маршрут, който наричаме свое собствено действие от контролера, което получава данните от потребителя и го използва за изчисляване на резултата.  
Тъй като ще изпратим формулярни данни на сървъра, ще използваме заявка за HTTP POST, за да го направим. Следователно, трябва да кажем routes.js да очакваме POST искания и да изпълним съответното действие на контролера:



Току-що направихме маршрут, който слуша "/" и при съвпадение изпълнява действието indexPost вътре в домашния контролер. Това е добре, но проблемът е, че все още нямаме действие с това име. Така че, познайте какво ще стане, ако се опитаме да изпратим формуляр до сървъра:



Не е толкова страхотно. За последния ни трик ще създадем свой собствен контролер, който ще обработва това, което потребителят ни е изпратил и ще върне изгледа с резултата от изчислението.

## 4. Implement the Controller Action

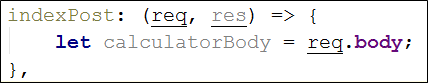
Сега, след като създадохме изгледа, който ще държи данните ни и ще позволи на потребителя да взаимодейства с нашето уеб приложение, е време да създадем движещата сила зад цялото приложение - контролера.  
Както се оказа, вече имаме инсталиран домашен контролер и действие, настроено на маршрута "/", в противен случай дори няма да можем да видим нашия калкулатор. Главният контролер можете да намерите в папката "контролери". Да видим как изглежда:



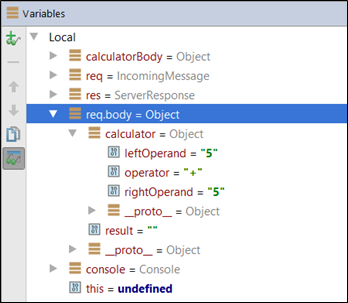
Не се случва много тук ... Нека да го счупим:  
• module.exports  част от кода, който отнема кода вътре и го излага на външния свят. Тук поставяме кода си, защото маршрутизаторът трябва да има достъп до него, така че той може да изпълни действието, когато потребителят се обади на посочения маршрут (точно сега, това е "/").  
• indexGet: (req, res)  Това е действителното действие на контролера. Това е функция, която държи логиката, която ще бъде изпълнена, когато се нарича. Това не се различава от обикновения метод.  
Той има 2 параметъра: req и res. Те съхраняват данни за HTTP заявката и HTTP отговор съответно. Те ще бъдат използвани за получаване на данни от потребителя и също така правят такива неща, като показване на изгледи в отговора. Запомнете - всичко, което правим тук, е връщането на различен HTML код въз основа на логиката, която сме въвели в нашето приложение.  
• res.render ("home / index")  Тази функция прави изглед в отговора (по същество отнема всичко, което е вътре в "home / index.hbs", преминава през шаблона на двигателя и я връща обратно на потребител.  
Така че, използвайки това новосъздадено знание, нека се опитаме да създадем свои собствени действия.  
Първо, ще започнем, като обявим действието:



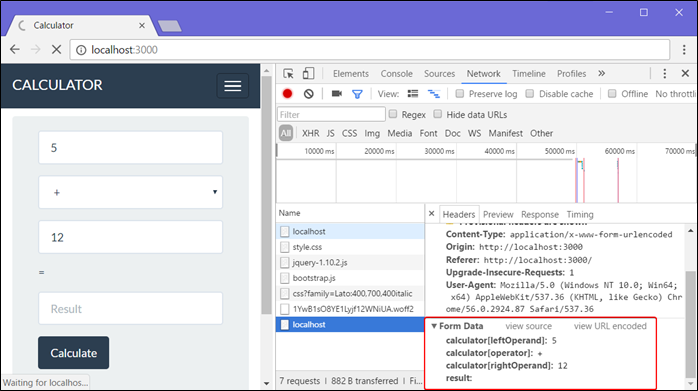
Както знаем от по-горе, имаме параметри req и res. Ще използваме параметъра req, за да получите данните от заявката. Това са данните, които потребителят ни е изпратил чрез формуляра. Можем да се справим с тези данни чрез достъп до req.body:

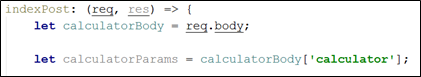


Преди да продължим, нека видим какво ще притежава този имот, ако го разгледаме по време на сесия за отстраняване на грешки:



Данните идват от искането като променлива на калкулатора. Защо се случва това? Е, нека да разгледаме какви данни изпращаме на сървъра с инструмент като Инструменти за разработчици на Chrome (F12):

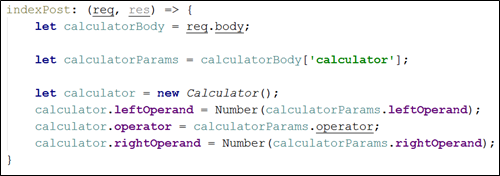
Формулярът има калкулатор с 3 стойности, изпратени като струни. И като такъв, това включва тялото на искане - нещо, до което можем да получим достъп, като използваме свойството req.body. Нека да се върнем в нашата контролна дейност и да обработваме данните.  
Можем да получим достъп до свойствата на калкулатора чрез достъп до тялото на заявката и получаване на калкулатора като ключ:



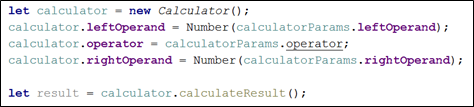
Достъпът до стойностите на calculatoryBody чрез клавиша "калкулатор" е същият като просто написване на "req.body ['calculator']", но ние извличаме всяка стъпка в променливи за яснота.  
Преди да можем да използваме нашия модел калкулатор, трябва да кажем на администратора, че то съществува. Ще го направим, като го импортираме, като използваме изисква () в горната част на файла:



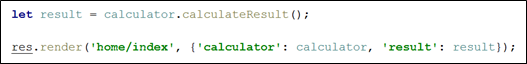
След това трябва да създадем пример за нашия модел на калкулатора, който ще използваме за съхранение на данните в него:



Използваме функцията Number (), за да превърнем операндите от низове в числа. Сега, след като получихме данните, е време да изчислим резултата от това, което имаме понастоящем. Не забравяйте, че функцията calculateResult (), която написахме преди малко? Сега е време да го използвате:



След това всичко, което сме оставили, е да направим изгледа. Можем да направим това, като използваме функцията за рендиране в параметъра на метода res:

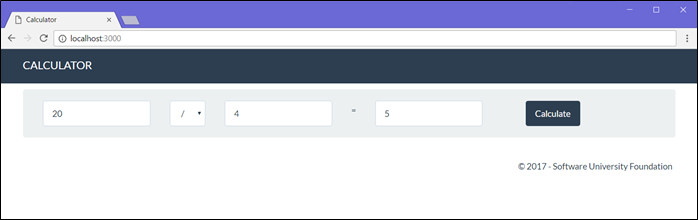


Нека разбием какво прави res.render ():  
• "у дома / индекс"  
Този параметър определя кой изглед да се върне.  
• {'калкулатор': калкулатор, 'резултат': резултат}  
Този параметър е обект на JavaScript, който уточнява какво ще изпратим до изгледа (в нашия добре познат ключ -> двойки стойност).  
Така че, когато изпратим обекта на калкулатора и резултантната стойност на изгледа, можем да попълним полетата на формуляра с нашите данни. Това се случва тук:

Използваме данните от контролера в изгледа home / index.hbs, за да зададем стойностите на входовете на формуляри за това, което искаме. В този случай зададете операндите и изберете последния използван оператор.

# III. Test the Application

Цялата ни упорита работа трябва най-накрая да се изплати, нали? Ако сте следвали всички стъпки правилно и прочетете целия обяснителен текст, надявайте се, че трябва да имаме функциониращ калкулатор!



# IV. \* Implement Extra Functionality

Също като последния път имате свободата да прилагате допълнителна функционалност, като допълнителни оператори, валидиране на вход и каквото и да е друго, за което можете да си представите. Честит кодиране.🙂  
Следващия път ще използваме същата логика като тази в тази лаборатория, за да внедрим напълно функционираща система от блогове, с база данни зад нея за съхранение на всичко и дори удостоверяване и сигурност на потребителите. Забавлявайте се с JS!🙂