

**Факултет по математика и информатика**

Проект по дисциплината

**“Фреймуърк системи за уеб  
програмиране”**

Специалност “Софтуерни технологии и дизайн”,

4-ти Курс

# **Aura Framework**

Изготвили: Мартина Димитриева 1501681022

Ралица Паунова 1501681040

Дата: 25.02.2019

<b>Описание</b>	<b>4</b>
Защо да използваме Aura?	4
Къде се използва?	5
<b>Как да го използваме?</b>	<b>6</b>
Документация	6
Настройване на рамката локално за създаване на независими приложения	7
Създаване на приложение/компонент в Salesforce	10
<b>Технологии и Принципи на Фреймуорка</b>	<b>12</b>
Компоненти	12
Основни елементи на Aura компонентите	13
Събития	18
JavaScript контролер	20
Видове събития	20
Java Servlets с Aura	22
<b>Основни компоненти</b>	<b>25</b>
Aura namespace (именовано пространство) :	25
UI именно пространство	26
<b>Примерен проект</b>	<b>30</b>

Разработката има за цел да предостави подробна информация за начина на работа и възможностите, които предлага фреймуърк системата за разработка на уеб базирани системи, **"Aura Framework"**.

Документът дава възможност на читателя да се запознае с основните концепции на фреймуърка, принципите и технологиите които използва, както и начини на употреба и интегриране. Нагледно е изложена употребата на основите компоненти, които предоставя рамката, чрез практически пример в последната секция от документа.

# Описание

Aura е open-source UI фреймуърк, разработен от Salesforce.com за изграждане на динамични уеб приложения за мобилни и настолни устройства. Aura е модерен фреймуърк за изграждане на single-page приложения, насочени към разрастващи се и дългосрочни бизнес решения.

Aura поддържа разработването на приложения с многослойна архитектура, която свързва клиента и сървъра. Използва JavaScript от страна на клиента и Java от страна на сървъра. Aura е компонентно-базиран фреймуърк, който позволява преизползването на компоненти, независими един от друг, съчетавайки основните концепции на събитийно-ориентирана архитектура.

## Защо да използваме Aura?

### Out-of-the-box сет от компоненти

Предлага се с набор от готови компоненти, които могат да се използват за стартиране на приложения, без да е необходима специална конфигурация или модификация. Не е нужно разработчика да прекарва времето си оптимизирайки приложенията за различни устройства, тъй като компонентите се грижат за това.

### Производителност

Използва клиент-сървър архитектура, която разчита на JavaScript от страна на клиента, за да управлява както метаданните на компонентите на потребителския интерфейс, така и самите данни с които работи приложението. Клиентът извиква сървъра само когато е абсолютно необходимо (например, за да получите повече метаданни или данни). Сървърът изпраща единствено данните, необходими на потребителя, за да увеличи максимално ефективността. Фреймуърка използва JSON за обмен на данни между сървър и клиент. Ефикасно се използват наличните сървър, браузър, устройства и мрежа, така че разработчика да може да се съсредоточи върху бизнес логиката на приложението.

### Събитийно-ориентирана архитектура

Използва събитийно-ориентираната архитектура, за по-добро разграничаване и подразделяне на компонентите.

## По-бърз процес на разработване

Дава възможност на екипите да работят по-бързо с готов набор от компоненти, които функционират безпроблемно с десктоп и мобилни устройства. Създаването на едно приложение, използвайки компоненти улеснява неговия дизайн, подобрявайки цялостната ефективност на разработката. Aura осигурява основните концепции на наследяване, полиморфизъм и капсулиране от обектно-ориентираното програмиране и ги прилага към разработването на презентационния слой. Фреймуърка позволява разширяването на компонент или прилагането на интерфейс към компонент. Компонентите са капсулирани и вътрешните им части остават частни, а публичната им форма е видима за потребителите на компонента. Това силно разделение дава на разработчиците на компонент свободата да променят вътрешните детайли по изпълнението и логиката и изолира потребителите компонента от тези промени.

## Съвместимост с устройства и браузъри

Приложенията използват отзивчив дизайн и осигуряват приятно потребителско изживяване. Aura поддържа най-новата технология на браузъра, като например HTML5, CSS3 и събития с докосване.

## Къде се използва?

Aura фреймуърк може да се използва за независими уеб-базирани приложения за мобилни и настолни устройства, локално или на сървър, но основната му употреба е в **Salesforce Lightning Platform** - облачна система, която предоставя платформа като сървис (PaaS - Platform-as-a-service), с цел изграждане и разработването на персонализирани приложения, в зависимост от изискванията на бизнеса.

## Salesforce.com

Salesforce.com е глобален доставчик на съвкупност от облачни услуги, под формата на софтуер като услуга (Software-as-a-service - SaaS) и платформа като сървис (Platform-as-a-service - PaaS), насочени към разработка и развитие на големи бизнес решения.

От своите платформи и приложения в облака, компанията е най-известна със своя продукт за управление на връзките с клиенти Salesforce (CRM - customer relationship management): В най-основни линии, CRM софтуерът обединява информацията за клиентите и документите в една CRM база данни, така че бизнес потребителите да могат по-лесно да получават достъп до нея и да я управляват.

Възможностите на Salesforce не са ограничени само до основните функционалности на CRM системите посочени по-горе. Salesforce предлага различни сървиси на клиентите в зависимост от бизнес нуждите, съобразени с водещите тенденции в разработването на бизнес решения (например: Изкуствен интелект AI, Интернет на нещата IoT).

Salesforce се развива върху Lightning Platform (също известно като Force.com), която представлява платформа като услуга (PaaS). Тя позволява на разработчиците да създават допълнителни приложения, които се интегрират в основното приложение на Salesforce. Приложенията, които се изграждат върху платформата използват редица инструменти, като Lightning Components - инструмент за изграждане на потребителски интерфейс. В основата си този инструмент използва Aura Framework, чиято цел в началото е да бъде използван само в Salesforce, което предполага най-широкото му приложение именно в Salesforce платформата.

## Как да го използваме?

За да използвате тази рамка, може да си клонирате/свалите официалното гит репозитори, което е open source, линк : <https://github.com/forcedotcom/aura>

Рамката Aura може да се използва да за създаване на напълно независими приложения както и на приложения в Salesforce.

Ако използвате рамката в Salesforce, няма нужда от никакви допълнителни настройки, както и от клониране/сваляне на самата рамка. Salesforce платформата се е погрижили за тази част, единственото което трябва да направите е да създадете един нов Lightning Component.

Ако искате да използвате Аура рамката локално, в секция "Настройване на рамката локално за създаване на независими приложения" е описано по-детайлно какви са стъпките които трябва да направите за да можете да използвате тази рамка.

## Документация

Документацията на тази рамка може да я намерите на следните линкове:

- [https://github.com/forcedotcom/aura/blob/master/aura\\_oss.pdf](https://github.com/forcedotcom/aura/blob/master/aura_oss.pdf)
- <https://<myDomain>.lightning.force.com/auradocs/reference.app>, където <myDomain> е името на вашият персонализиран Salesforce domain.
- Ако ползвате рамката локално, документацията може да я намерите на следващия линк: <http://localhost:8080/auradocs/docs.app>

# Настройване на рамката локално за създаване на независими приложения

Най-лесният начин за изграждане на първото приложение на Aura е от конзолата, но можете лесно да използвате и Aura с любимото си IDE.

Aura е open-source фреймуърк, а кода може да откриете в GitHub на следния адрес:

<https://github.com/forcedotcom/aura>

## Предварителни изисквания:

- JDK 1.8
- Apache Maven 3
- Node 4.1.0+
- NPM 2.2.0+
- grunt-cli 1.3.1+
- Google Chrome 67.0+
- Chromedriver 2.41

## Стъпки

- Стъпка 1: Install Node, NPM, grunt, and Chromedriver
  - Инсталирайте Node и NPM (<https://nodejs.org/en/download/>)
  - Инсталирайте grunt-cli (<https://gruntjs.com/getting-started>)
  - Изтеглете Google Chromedriver за вашата оперативна система
    - Mac  
([http://chromedriver.storage.googleapis.com/2.41/chromedriver\\_mac64.zip](http://chromedriver.storage.googleapis.com/2.41/chromedriver_mac64.zip))
    - Linux  
([http://chromedriver.storage.googleapis.com/2.41/chromedriver\\_linux64.zip](http://chromedriver.storage.googleapis.com/2.41/chromedriver_linux64.zip))
  - Разархивирайте chromedriver в ~/bin/
  - Уверете се че node, npm and chromedriver се появяват в променливата PATH (изпълнете `echo $PATH` в конзолата). Ако не можете да видите node, npm or chromedriver в PATH, не забравяйте да го добавите в PATH чрез функцията за експортиране, например: `echo "export PATH=$PATH:$HOME/bin" >> $HOME/.bash_profile` Или поставете директно във вашите \$ HOME / .bash\_profile, \$ HOME / .profile, \$ HOME / .bashrc etc файлове

- Стъпка 2: Клонирайте Aura git repo and компилирайте Archetype
  - Отворете конзолата.
  - Клонирайте или изтеглете Aura maven project
  - Навигирайте до основната директория, където се съхранява aura
  - Компилирайте Aura
    - `mvn clean install`
  - Създадете темплейт от Aura Archetype
    - Отворете конзолата.
    - Навигирайте до директорията където искате да създадете вашият темплейт за проекта и изпълнете:
 

```
mvn archetype:generate -DarchetypeCatalog=local
```
    - Когато трябва да изберете archetype, въведете номера на съответния Aura archetype.
 

```
Choose archetype: 1: local ->
org.auraframework:simple-aura-archetype (archetype for
Aura-based "hello, world")
```
    - Въведете следващите стойности:

```
Define value for property 'groupId': org.myGroup
```

```
Define value for property 'artifactId':
```

```
helloWorld
```

```
Define value for property 'version':
```

```
1.0-SNAPSHOT
```

```
Define value for property 'package': org.myGroup
```

```
**Note**: The artifactId can only contain
alphanumeric characters.
```

- Когато трябва да потвърдите конфигурацията на пропертита, въведете Y. Това е резултата който трябва да видите когато темплейта е създаден успешно.

```
[INFO]
```

```
[INFO] Using following parameters for creating project
from Archetype: aura-archetype:0.0.1-SNAPSHOT
```

```
[INFO]
```

```
[INFO] Parameter: groupId, Value: org.myGroup
```

```
[INFO] Parameter: artifactId, Value: helloWorld
```

```
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
```

```
[INFO] Parameter: package, Value: org.myGroup
```



```

[INFO] Parameter: packageInPathFormat, Value:
org.myGroup
[INFO] Parameter: package, Value: org.myGroup
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: groupId, Value: org.myGroup
[INFO] Parameter: artifactId, Value: foo
[INFO] project created from Archetype in dir: /home/
[INFO]
-----
[INFO] BUILD SUCCESS
[INFO]
-----
[INFO] Total time: 33.656s
[INFO] Finished at: Tue Jul 16 14:39:07 PST 2013
[INFO] Final Memory: 10M/180M
[INFO]
-----

```

- Стъпка 3: Компилирайте и пуснете вашият проект
  - През командния промпт навигирайте до директорията на вашата нова програма.
 

```
cd helloWorld
```
  - Build the app.
 

```
mvn clean install
```
  - Пуснете Jetty server на порт 8080.
 

```
mvn jetty:run
```
  - За да ползвате друг порт, добавете
 

```
-Djetty.port=portNumber. For example, mvn jetty:run
-Djetty.port=9877.
```
  - За да тествате вашето приложение, може да ползвате следващи линк: `http://<myServer>/<namespace>/<component>.cmp`

```
http://localhost:8080/example/helloWorld.cmp
```
  - За да спрете Jetty server и освободите порта когато приключите, само натиснете `CTRL+C` в командния промпт.

### **Забележка:**

Файлът `helloWorld/pom.xml` има секция с зависимости(`<dependencies>`) в която се показани версиите на всички артефакти на вашият проект. Те дефинират версията на Аура, която вашият проект използва и всеки артефакт трябва да използва същата версия.

# Създаване на приложение/компонент в Salesforce

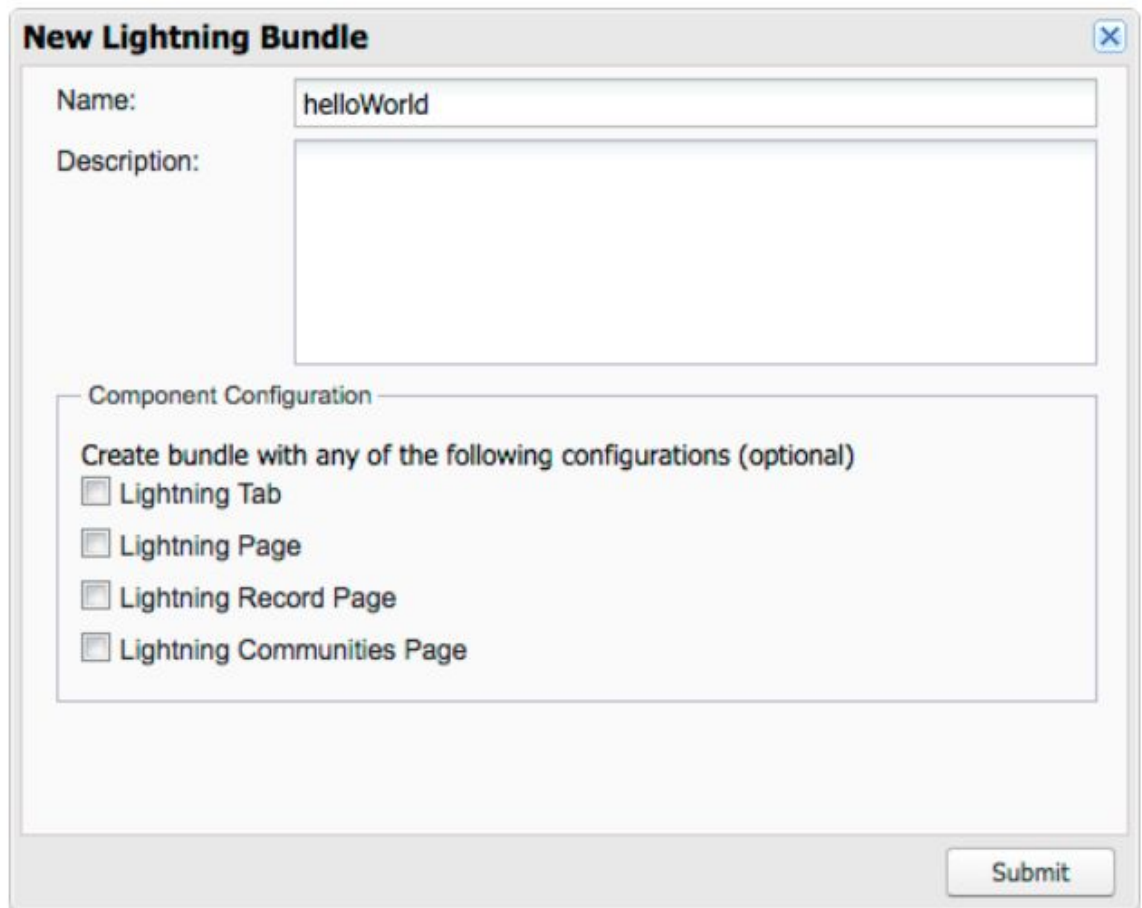
За да се използва рамката в Salesforce, трябва да се използва моделът за програмиране на Aura Components който се базира на самата рамка

*Имайте предвид, че Aura рамката с отворен код има функции и компоненти, които не са налични в модела за програмиране на Aura Components*

За да използвате тази рамка в Salesforce, трябва да си създадете една организация (<https://developer.salesforce.com/signup>), или да ползвате вече съществуваща организация. Важно е да се спомене, че Salesforce препоръчват организацията която ще ползвате за разработване на приложения да не е от тип production. За повече информации, за това какви видове организации в Salesforce съществуват, може да намерите на следния линк: [https://developer.salesforce.com/page/An\\_Introduction\\_to\\_Environments](https://developer.salesforce.com/page/An_Introduction_to_Environments)

Стъпки за създаване на един компонент в Salesforce:

1. Логнете се във вашия акаунт с credenшълите, които сте задали при създаването му в <https://login.salesforce.com/>.
2. Отворете девелоперската конзола или IDE ако ползвате такова. Девелоперската конзола може да я намерите под Вашето име в Организацията ако сте по Classic или от менюто за бърз достъп ако сте в Lightning().
3. От конзолата изберете **File | New | Lightning Component** за да създадете нов компонент.
4. Ще се отвори едно прозорче в което трябва да въведете задължително името на самия компонент. Освен поле за име има и поле да описание което не е задължително. Другите полета от този прозорец няма да ги разглеждаме, тъй като се отнасят за Salesforce. Тези полета се за конфигурация на компонента в Salesforce.



**New Lightning Bundle**

Name: helloWorld

Description:

Component Configuration

Create bundle with any of the following configurations (optional)

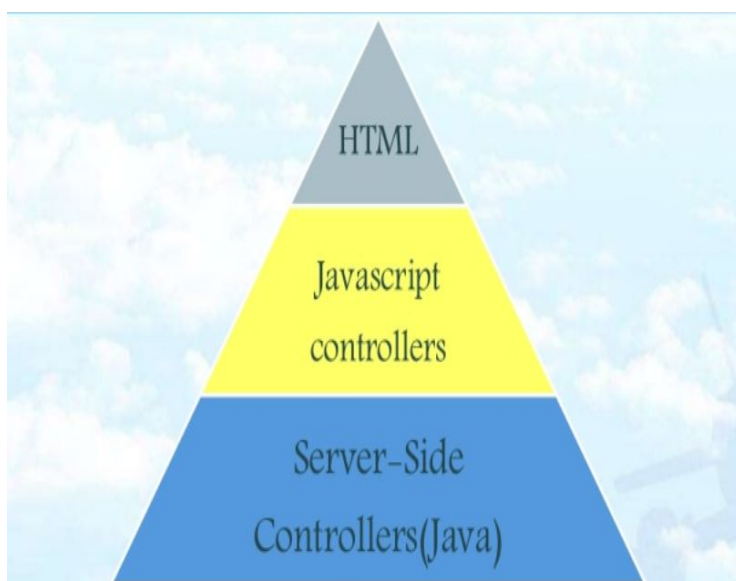
- ☐ Lightning Tab
- ☐ Lightning Page
- ☐ Lightning Record Page
- ☐ Lightning Communities Page

Submit

5. Кликнете на **Submit** за да създадете компонента.

# Технологии и Принципи на Фреймуорка

Aura рамката използва JavaScript от страна на клиента и Java от страна на сървъра за изграждането на своята многослойна архитектура. Следващото изображение предоставя нагледно основните елементи, които се използват в отделните слоеве на едно уеб-базирано приложение:



1. Компоненти и страници (с HTML маркъп/ псевдо HTML маркъп, CSS), които отговарят за потребителския интерфейс.
2. Контролери от страна на клиента (JavaScript) - функции, използвани за логика, изпълняваща се при клиента. Може да осъществи комуникацията клиент-сървър.
3. Сървърен контролер (Java) - Java код, който служи за изпълнение на логика на сървъра и връщане на резултат при комуникацията с клиента.

Понеже архитектурата на Aura рамката е компонентно-базирана и събитийно-ориентирана две ключови неща са заложени в нея:

- Компоненти
- Събития.

## Компоненти

Компонентите са самостоятелни и многократно използвани единици на приложението. Те представляват секция за повторно използване на потребителския интерфейс и могат да варират и да съдържат един ред от текст или дори цялото приложение.

Фреймуъркът включва набор от предварително подготвени компоненти в именуваните пространства (namespaces) **aura** и **ui**. Могат да се сглобят и конфигурират компоненти, за да се формират нови компоненти в

приложението. Всеки компонент е част от определено именовано пространство, което се използва за групиране на свързаните компоненти заедно (такива които отговарят за определена бизнес логика). В брауъра, компонентите се рендерират до HTML DOM елементи.

Един компонент може да съдържа в себе си и други компоненти, както и HTML, CSS, JavaScript или всеки друг код, уеб базиран код. Това дава възможност за изграждане на приложения със сложни потребителски интерфейси.

Подробностите по имплементирането на компонента са капсулирани. Това позволява на потребителя на компонента да се съсредоточи върху изграждането на приложението, докато авторът на компонента може да прави нововъведения и да прави промени, без това да попречи на работата на потребителя. Компонентите си взаимодействат помежду си чрез събития. Сигурният събитийен модел на Aura позволява разработването на слабо обвързани компоненти.

## Основни елементи на Aura компонентите

Компонентите са функционалните единици на рамката. Един компонент модулно капсулира логика с визуални компоненти в потенциално преизползваема част на потребителския интерфейс.

### Маркър на Компонентите

Компонентите съдържат маркър и имат `.cmp` суфикс. Маркърът може да съдържа текст или препратки към други компоненти и също така да декларира метаданни за компонента. Пример за маркър на компонент е представен чрез ***helloWorld.cmp***:

```
<aura:component>
Hello, world!
</aura:component>
```

Компонентите могат да съдържат и HTML и HTML5 тагове, така че може да се използват за маркиране тагове, като `<div>` и `<span>`, без да се наруши работата на приложението.

## Именовано пространство (namespace)

Всеки компонент е част от пространство от имена, което се използва за групиране на свързаните компоненти.

Друг компонент или приложение може да се извика и използва компонент, като добави **<myNamespace: myComponent>** в неговия код за маркиране. Нека компонентът helloWorld се намира в пространството с имена на **"docsample"**. Друг компонент, който не е в същото именовано пространство може да го посочи, като добави **<docsample: helloWorld />** в своята маркировка.

Всички базови aura компоненти директорията **"aura-components/components"**. Всички папки в тази директория са част от съответното именовано пространство. Всяка папка в този namespace реферира към определен компонент и съдържа всички файлове необходими за функционирането на компонента. Тази папка се нарича още **"Component Bundle"** (пакет или съвкупност от файлове).

## Component Bundle (компонентен пакет)

Компонентният пакет съдържа компонент или приложение и всички свързани с него файлове. В следващите редове е предоставен списък със всички типове файлове, които могат да се включат в компонентния пакет. **Важно е да се отбележи, че освен файловете с маркър на компонента или приложението, другите ресурси не са задължителни за изграждането на компонента/приложението.**

- **Компонент или Приложение** - Единствения задължителен ресурс в пакет. Съдържа маркировка за компонент или приложение. Всеки пакет съдържа само един ресурс от този вид.

Приложението (Aura Application) е специален компонент от най-високо ниво, чиито макрърп е във файл .app разширение. Маркърпа е подобен на HTML и може да съдържа компоненти, както и набор от поддържани HTML тагове. Приложението е входна точка за приложението и дава възможност за дефиниране на цялостното оформление на приложението, чрез стилове и глобално използване на JavaScript. Започва с маркер от най-високо ниво **<aura: application>**.

**Пример:** sample.cmp или sample.app

- **CSS Стиловане** - Съдържа стилове, които се прилагат върху компонента. За да се добави CSS към компонента, трябва да се добави нов файл към компонентния пакет, наречен <componentName> .css. Рамката автоматично разбира, към кой компонент принадлежи този нов файл и го включва автоматично, когато компонентът се използва в страницата. Елемента от най-високо ниво в компонента имат добавен специален CSS клас **".THIS"**. Този специален клас спомага за предотвратяването CSS-а на един компонент да променя стилизацията на друг компонент. Рамката хвърля грешка, ако CSS файлът не следва тази конвенция и не се използва **".THIS"** класа.

**Пример:** sample.css

### CSS

.THIS { background-color: grey; } //background цвета на компонента ще е сив

- **Контролер** - Представява контролер от страна на клиента, с методи за обработка на събитията в компонента.

**Пример:** sampleController.js

- **Документация** - Описание, примерен код и една или няколко препратки към примерни компоненти. Файлът е с разширение .auradoc.

**Пример:** sample.auradoc

- **Renderer** - Рендерер при клиента за предефиниране на стандартния процес на рендериране на компонента.  
Помощен файл (Helper) - Помощни методи, които се споделят от контролера и рендера.

**Пример:** sampleRenderer.js

- **Helper** - JavaScript функции, които се споделят и могат да се използват от които се споделят от контролера и рендера

**Пример:** sampleHelper.js

- **Provider** - Клиентски доставчик, който връща информация за конкретния компонент, който се използва по време на изпълнение.

**Пример:** sampleProvider.js

- **Тестови случаи** - Съдържа тестови пакет, който ще се изпълнява в брауъра.

**Пример:** sampleTest.js

## Component IDs

Всеки компонент има два вида идентификатори: локален идентификатор (LocalId) и глобален идентификатор (GlobalId).

- **Локален Идентификатор**

Локалният идентификатор е уникален в рамките на компонента и е в скоупа само на компонента. Използвайки локален идентификатор компонента може да бъде достъпван в JavaScript кода. Начина, по който локален идентификатор се дефинира е чрез използване на "aura:id " атрибута на компонента. Например:

```
<ui:button aura:id="button1" label="button1"/>
```

Компонента **ui:button** може да бъде рефериран в клиентския (JavaScript) контролер чрез cmp.find ("button1"), където cmp е препратка към компонента, в който се съдържа **ui:button** компонента.

### sampleUIButton.cmp

```
<aura:component>
    <ui:button aura:id="button_local_id" label="Get Local ID"
        press="{!c.getComponent}"/>
</aura:component>
```

### sampleUIButtonController.cmp

```
({
    getComponent : function(cmp, eventt)
    {
        var localId = cmp.find("button_local_id");
    }
})
```

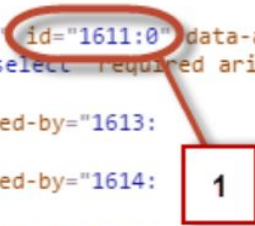


- **Глобален Идентификатор**

Всеки компонент има уникален Глобален идентификатор, генериран за всяка инстанция на компонента. (1) Не е гарантирано, че глобалния идентификатор, ще бъде същият винаги, така че никога не трябва да се разчита на него.

Глобалният идентификатор може да бъде полезен за разграничаване множеството инстанции на компонента по време на живота му в приложението или с цел дебъгване за отстраняване на грешки.

```
▼ <div class="slds-select_container" data-aura-rendered-by="1624:0">
  ::before
  ▼ <select class="slds-select" id="1611:0" data-aura-rendered-by="1625:0" name="select_required" aria-describedby="1611:0-desc">
    <option data-aura-rendered-by="1613:0">Red</option>
    <option data-aura-rendered-by="1614:0">Green</option>
    <option data-aura-rendered-by="1615:0">Blue</option>
  </select>
  ::after
</div>
```



## Атрибути на компонента

Атрибутите на даден компонент са като променливи на класа (полета) в обектно ориентираните езици за програмиране като Java. Те са типизирани полета, които се задават за конкретна инстанция на компонента, и могат да бъдат реферирани в маркиркъпа на компонента, използвайки синтаксиса на aura изразите. Атрибутите позволяват компоненти да бъдат по-динамични.

Използва се <aura: attribute> тага, за да добави атрибут към даден компонент или приложение. Следния пример илюстрира дефинирането на атрибут в приложението **"helloAttributes.app"**:

```
<aura:component>
  <aura:attribute name="whom" type="String" default="world"/>
  Hello {!v.whom}!
</aura:component>
```

Всички атрибути имат тип и име. В примера името на атрибута е "whom" и е от тип String.

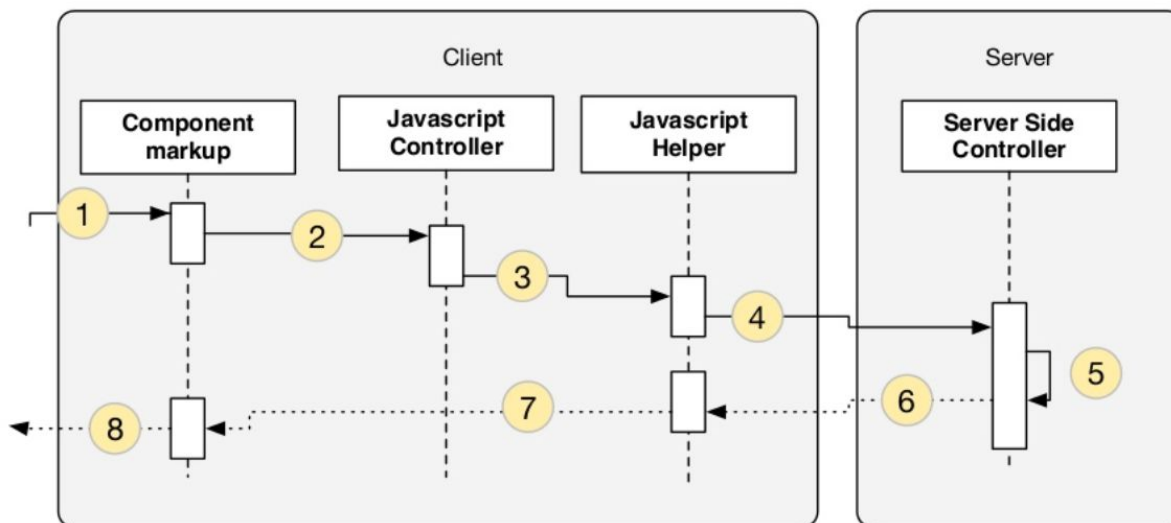
## Aura Expressions (Изрази)

**"helloAttributes.app"** приложението, демонстрирано в секцията "Атрибути на компонента", съдържа израз **{!v.whom}**, който отговаря за визуализирането на динамични данни в приложението. **"{! <Expression>}"** е синтаксиса, който се използва за използване на изрази в маркъпа на компонента. В разглеждания пример изразът **{!v.whom}** реферира към атрибута, дефиниран с име **"whom"**, докато **"v"** представлява View (изглежда) на компонента. Целия израз ни връща стойността, която държи атрибута **"whom"**, която в примера е по подразбиране - **"world"**.

## Събития

Събитията възникват при взаимодействието на клиента с потребителския интерфейс на приложението. Рамката използва събития за препредаване на данни между компонентите. Идеята е при възникване на събитие, да бъде предоставен механизъм за обработване на събитията (handler), който обикновено е под формата на функции в клиентския (JavaScript) контролер.

В схемата по-долу е описан процеса на прихващане на възникналите събития, в следствие на взаимодействието на клиента, тяхното обработване от клиентския (JavaScript) контролер, изпращането на заявки към сървъра под формата на екшъни, и обработването на резултата, върнат от сървъра.



1. Потребителят избира с мишката бутон или взаимодейства с компонент, като по този начин предизвиква събитие в браузъра.
2. При възникване на събитието се извиква функция в JavaScript клиентския контролер, която предоставя персонализирана логика, при обработването на възникналото събитие, преди да извика помощна функция.
3. Клиентския контролер извиква помощна функция от Helper файла. Помощните функции подобряват преизползването на код в контролера, но е опционален и незадължителен.
4. Помощната функция извиква метод на сървърния контролер под формата на екшън и го добавя в опашката от екшъни и чака да бъде изпълнен, когато сървър е свободен.
5. Извиква се методът от страна на сървъра, извършва персонализирана логика и се връщат данни.
6. След изпълнението на метода на сървъра, се извиква JavaScript callback функция, която съдържа резултата, върнат от сървъра и статуса на заявката.
7. Функцията за обратно извикване в JavaScript контролера оценява логиката и актуализира потребителския интерфейс на компонента.
8. Потребителят вижда актуализираното съдържание

## JavaScript контролер

Клиентският контролер обработва събитията в компонента.

Това е JavaScript файл, който дефинира екшън функциите в компонента.

Всяка екшън функция приема три параметъра:

- Компонент (component), към който принадлежи контролера,
- Събитието (event), което се обработва от текущата функция
- Helper, който съдържа помощни функции, които могат да се използват от различните екшън функции в контролера. Той е опционален.

Клиентските контролери са обособени от скоби и къдрави скоби, обозначаващи JSON обект, съдържащ map с двойка **име - стойност**.

### Пример :

```
sampleController.js
{
  handleClick: function (component, event, helper) {

  }
}
```

## Видове събития

Aura фреймуърк разграничава два вида събития:

- **Компонентните събития** - предизвиквани са и се обработват от самия компонент, в който са възникнали или от компонент, който инстанцира или съдържа компонента.
- **Събития, предизвикани в приложението** - могат да се обработят от всички компоненти, които слушат за събитието. Тези събития по същество се основават на традиционния модел публикуване - абонамент (publish-subscribe).

## Създаване на Събития

Събитията се създават в файл с разширение .evt, със същото име както на компонентния пакет (Component Bundle).

Използвайте **type="COMPONENT"** в `<aura:event>` тага при създаването на компонентно събитие или **type="APPLICATION"** ако създавате събития на приложението. Да вземем следния пример със даден евент, съдържащ атрибут **"message"**. Атрибутите се използват, за да мога да съхраняват данни, при комуникацията между отделните компоненти.

```
<aura:event type="COMPONENT">
    <aura:attribute name="message" type="String"/>
</aura:event>
```

## Регистриране на Събития

Един компонент може да "уведоми", че ще предизвика дадено събитие като регистрира събитието в своя маркъп, използвайки `<aura:registerEvent>` тага.

```
<aura:registerEvent name="sampleComponentEvent" type="c:sampleEvent"/>
```

В този пример регистрирахме в `sample.cmp` компонент, събитието, което създадохме в предната секция.

Предизвикването на събитието се извършва в контролера на компонента, където е регистрирано събитието:

```
...
var compEvent = cmp.getEvent("sampleComponentEvent");
// Опционално можем да добавим параметри към
// евента//
compEvent.setParams({"myParam" : myValue });
compEvent.fire();
...
```

## Обработване на Събития

За да се обработи възникналия евент се използва `<aura:handler>` таг в маркъпа на компонента, който ще обработи събитието.

**Например:**

```
<aura:handler name="sampleComponentEvent" event="c:sampleEvent"
action="{!c.handleSampleEvent}"/>
```

Атрибутът “event” определя събитието, което се обработва. Форматът е следния **namespace:eventName**. Атрибутът “action” се за извикване на функционалността на клиентския контролер, която трябва да обработи събитието.

## Java Servlets с Aura

За сървърната логика може да се използва Java като език за програмиране. Както вече стана ясно, фреймуърка поддържа клиентски контролери и сървърни контролери. Събитията винаги са свързани с екшън (action) на клиентския контролер, което може извика сървърния контролер. Например, JavaScript контролер може да обработи дадено събитие и да извика действие на контролера на сървъра, за да се запазят данни в базата данни.

### Създаване на сървърен контролер

За да създадем контролер на сървъра, използвайте анотациите **@ServiceController** и **@AuraEnabled**.

**@ServiceController** - Оказва, че дадения клас се използва като сървърен контролер. Класът също трябва да приложи интерфейса **Controller**.

**@AuraEnabled** - Дава достъп на клиента до метода на контролера. Това означава, че могат да се ползват само методи, анотирани по този начин, могат да се реферират от контролера при клиента.

Методите на контролера, които са анотирани с **@AuraEnabled** трябва да са публични и статични.

```
package org.auraframework.demo.controllers;

@ServiceComponent
public class SimpleServerSideController implements Controller
{
    //Use @AuraEnabled to enable client- and server-side access to the method
    @AuraEnabled
    public static String serverEcho(@Key("firstName")String firstName) {
        return ("From server: " + firstName);
    }
}
```

## Извикване на екшъни на сървърния контролер

За да се върже даден компонент към сървърен контролер трябва да добави controller атрибут в <aura:component> тага.

В клиентския контролер се задава callback функция, която се изпълнява след върнат резултат (response) от сървъра. Сървъра може да върне всеки обект, сериализиран в JSON обект.

### Нека разгледаме следния пример:

Клиентският контролер включва "echo" екшън, който се извиква при натискане на бутон в компонента. От своя страна "echo" екшъна извиква "serverEcho" екшъна на сървъра.

#### SimpleClient.cmp

```
<aura:component controller="org.auraframework.demo.controllers.SimpleController">
  <aura:attribute name="firstName" type="String" default="world"/>
  <ui:button label="Call server" press="{!c.echo}"/>
</aura:component>
```

#### SimpleClientController.js

```
{
  "echo" : function(cmp) {
    ...
  }
}
```

```
{
  "echo" : function(cmp) {
    var action = cmp.get("c.serverEcho");
    action.setParams({ firstName : cmp.get("v.firstName") });
    action.setCallback(this, function(response) {
      var state = response.getState();
      if (state === "SUCCESS") {
        //Update the state and response
      }
      else if (state === "ERROR") {
        ...
      }
    });
    $A.enqueueAction(action);
  }
}
```

Използвайки **cmp.get("c.serverEcho")**, извикваме serverEcho метода на сървъра.

Използваме **action.setParams({})**, за да изпратим данни до сървърния контролер. Стойностите се декларират в мап от име-стойност, като името трябва да отговаря на параметрите в метода на сървърния контролер.

**action.setCallback()** задава callback функция, която да се изпълня след върнат резултат от сървъра.

Резултатите от сървъра са достъпни през **response** променливата, която е аргумент на callback функцията.

**response.getState()** взима статуса на резултата, върнат от сървиса.

**\$A.enqueueAction(action)** добавя сървърния екшън към опашката от екшъни, които предстои да бъдат изпълнение.

### **Забележка:**

В последните си версии Aura фреймуърка дава свобода на разработчиците да пропуснат сървърния контролер при използването и консумирането на външни API-та, като им дава възможност да правят API повиквания директно от клиентския код. Въпреки това, това все още не се счита за добра практика. Препоръчително е вместо това да се използват API извиквания от сървърните контролери, за да се увеличи производителността.

Рамката използва XMLHttpRequest (XHR), за да комуникира от клиента към сървъра, а действията на сървъра са проектирани да минимизират мрежовия трафик и да се осигури по-плавен потребителски опит.



# ОСНОВНИ КОМПОНЕНТИ

## Aura namespace (именовано пространство) :

Компоненти:

- aura:component
  - Коренът на йерархията на компонентите. Provides a default rendering implementation.
- aura:expression
  - Извежда стойността, за която се оценява израз.
- Aura:html
  - Мета компонент, който представя всички html елементи.
- aura:if
  - Оператор за условие. Условно инстанцира и визуализира или тялото, или компонентите в атрибута else.
- aura:iteration
  - Итерира и показва елементите от една колекция. Поддържа итерации, съдържащи компоненти, които могат да бъдат създадени изключително от страната на клиента.
- aura:template
  - Шаблон по подразбиране, използван за зареждане на аурата. За да използвате друг шаблон, наследете aura:template и задайте атрибути, като използвате aura: set.
- aura:text
  - Показва обикновен текст. Когато в семантичния маркър е намерен свободен текст (а не маркер или стойност на атрибут), екземпляр на този компонент се създава със стойност, зададен в текста, намиращ се в семантичния маркър.
- aura:unescapedHtml
  - Стойността, присвоена на този компонент, ще се визуализира като такава, без да се променя съдържанието му. Той е предназначен за извеждане на предварително форматиран HTML,

Събития:

- aura:applicationEvent
  - Коренът на йерархията на всички събития от тип application
- aura:componentEvent
- aura:locationChange
  - Показва, че хеш частта на URL адреса в браузъра е променена.
- aura:methodCall
  - Показва, че публичен метод е бил извикан.
- aura:noAccess

- Показва, че търсеният ресурс не е достъпен поради ограничения на сигурността на този ресурс.
- aura:systemError
  - Показва че грешка се е случила.
- aura:valueChange
  - Показва, че стойността е била променена.
- aura:valueDestroy
  - Показва, че компонент се унищожава.
- aura:valueEvent
  - Коренното събитие на йерархията на събитията за всички събития от type = "VALUE"
- aura:valueInit
  - Показва, че компонент е инициализиран.
- aura:valueRender
  - Показва, че компонент е бил визуализиран.

## UI именно пространство :

Компоненти:

- ui:actionMenuItem
  - Елемент от менюто, който задейства действие. Този компонент е вложен в ui:menu компонента.
- Ui:button
  - Компонентът ui: button представлява бутон, който изпълнява действие, дефинирано от контролер.
- ui:checkboxMenuItem
  - Елемент от менюто с квадратче за отметка, което поддържа множество избори и може да се използва за извикване на действие. Този компонент е вложен в ui:menu компонента.
- ui:inputCheckbox
  - Представя отметка. Поведението му може да се конфигурира чрез събития като клик и промяна.
- ui:inputCurrency
  - Поле за въвеждане на валута.
- ui:inputDate
  - Поле за въвеждане на дата.
- ui:inputDateTime
  - Поле за въвеждане на дата и час
- ui:inputDefaultError
  - Показва грешките на ниво поле.
- ui:inputEmail
  - Представя поле за въвеждане на имейл адрес.
- ui:inputNumber
  - Представя поле за въвеждане на числа/

- `ui:inputPhone`
  - Представя поле за въвеждане на телефон.
- `ui:inputRadio`
  -
- `ui:inputRichText`
  - Поле за въвеждане на текст. Този компонент не се поддържа от LockerService.
- `ui:inputSecret`
  - UI: `inputSecret` компонент представлява поле за парола, което се визуализира като HTML таг за тип парола.
- `ui:inputSelect`
  - Представя падащ списък с опции.
- `ui:inputSelectOption`
  - HTML елемент, който е вложен в компонент `ui: inputSelect`. Означава наличните опции в списъка.
- `ui:inputText`
  - Представя поле за въвеждане, подходящо за въвеждане на един ред от текст в свободна форма.
- `ui:inputTextArea`
  - HTML `textarea`, който може да бъде само за редактиране или само за четене. Плъзгачите може да не се показват в браузърите на Chrome в устройства с Android, но можете да изберете фокус в текстовото поле, за да активирате превъртането.
- `ui:inputURL`
  - Поле за въвеждане на URL
- `Ui:menu`
  - Списък с падащо меню с тригер, който контролира неговата видимост. За да създадете кликуване на връзка и елементи от менюто, използвайте `ui: menuTriggerLink` и `ui: menuList` компонент.
- `ui:menuItem`
  - Елемент от менюто на потребителския интерфейс в компонент `ui: menuList`.
- `ui:menuList`
  - Компонент, който съдържа елементи от менюто.
- `ui:menuTrigger`
  - Връзка, която може да се кликне, която разширява и свива меню. За да създадете връзка за меню `ui:` използвайте `ui: menuTriggerLink`.
- `ui:menuTriggerLink`
  - Връзка, която задейства падащо меню, използвано в менюто `ui:`
- `Ui:message`
  - Представява съобщение с различни нива на тежест
- `ui:outputCheckbox`
  - Показва квадратче за отметка
- `ui:outputCurrency`

- Показва валутата по стандартен или определен формат, например с конкретен код на валутата или с десетични знаци.
- ui:outputDate
  - Показва дата в стандартния или зададения формат на базата на мястото на потребителя.
- ui:outputDateTime
  - Показва дата, час в определен формат или формат по подразбиране въз основа на мястото на потребителя.
- ui:outputEmail
  - Показва имейл адрес в HTML anchor (<a>) елемента.
- ui:outputNumber
  - Показва номера в стандартен или специфичен формат. Поддържа до 18 цифри преди десетичната запетая.
- ui:outputPhone
  - Показва телефонния номер във формат на URL връзка.
- ui:outputRichText
  - Показва форматиран текст, включително етикети като абзац, изображение и хипервръзка, както е посочено в атрибута value.
- ui:outputText
  - Показва текст, определен от атрибута value.
- ui:outputTextArea
  - Показва текстовата област, както е указано от атрибута value.
- ui:outputURL
  - Показва връзка към URL адрес, както е посочено от атрибута value, изобразен върху даден текст (атрибут на етикет) и изображение, ако има такава.
- ui:radioMenuItem
  - Компонентът ui: radioMenuItem представлява елемент от списъка на менюто за единичен избор.
- ui:scrollerWrapper
  - Създава контейнер, който позволява превъртане

#### Събития:

- ui:clearErrors
  - Показва, че грешките при валидирането са изчистени.
- Ui:collapse
  - Показва, че компонентът е сгънат. Това събитие се задейства, когато свиете компонента ui: menuList.
- ui:expand
  - Показва, че компонент е разширен. Това събитие се задейства, когато разширите компонента ui: menuList.
- ui:menuFocusChange
  - Показва, че фокусът на елемента в менюто е променен в менюто. Това събитие се задейства, когато превъртате нагоре и надолу компонента ui: menuList, който задейства промяна на фокуса в елементите на менюто.

- `ui:menuSelect`
  - Показва, че е избран елемент от менюто. Това събитие се задейства, когато изберете елемент от менюто в компонента `ui: menuList`.
- `ui:menuTriggerPress`
  - Това събитие се задейства, когато кликнете върху компонента `ui: menuTriggerLink`.
- `ui:validationError`
  - Показва, че компонентът има грешка (и) за проверка.

# Примерен проект

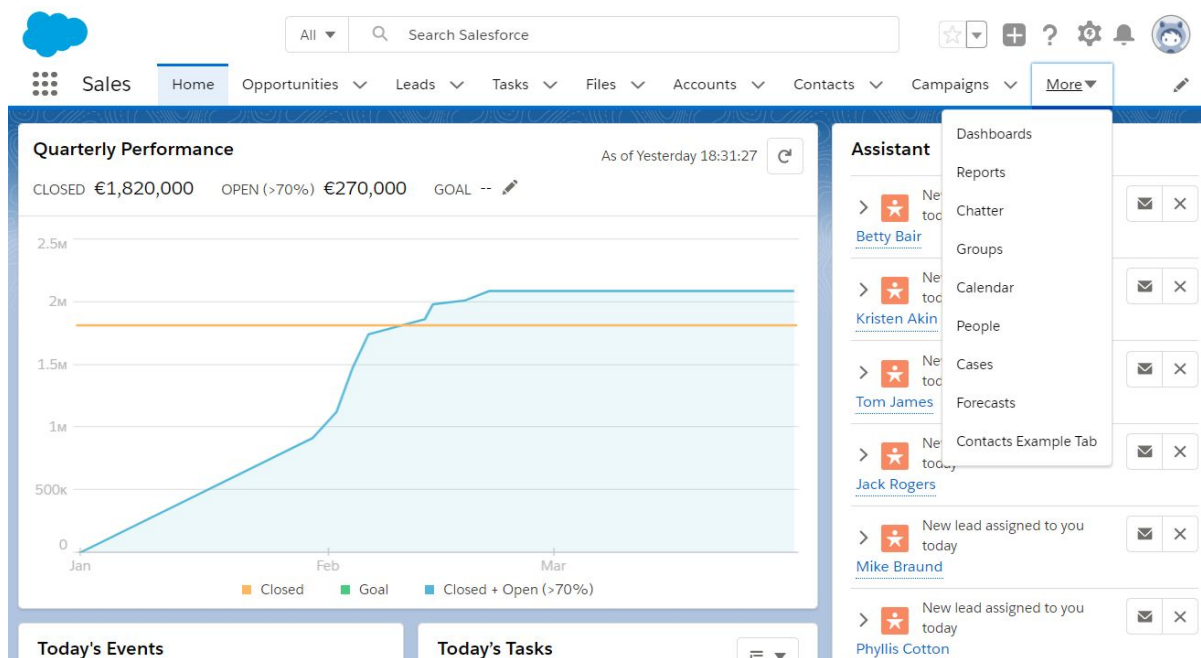
Примерният проект описан в следващата секция е Single page application и е изграден на Salesforce платформата. Този проект се състои от две неща, форма да за добавяне на нов контакт и таблица с всички контакти които съществуват в Salesforce инстанцията.

Ако искате, да разгледате самото приложение, използвайте следните кредитеншали за да се логнете в Salesforce инстанцията където е разработено, или може да си клонирате проекта от гитхъб и да го качите във ваша среда.

- Credentials:  
Username: [auraframework@example.com](mailto:auraframework@example.com)  
Password: example123
- Github линк:  
<https://github.com/limitlessmindgirl/auraFramework-uni-project>

За да достъпите приложението, първо трябва да се логнете в Salesforce инстанцията,

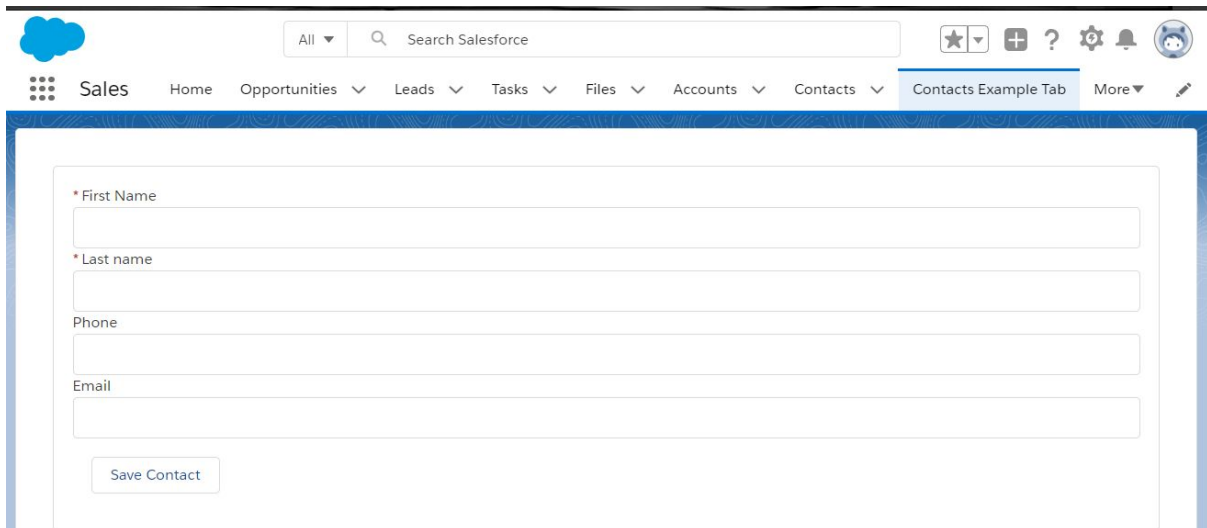
След като сте се аутентикирали в Salesforce, от менюто изберете "Contacts Example Tab"



След като сте избрали "Contacts Example Tab" приложението ще зареди двата компонента, които се използвани в него.

## createContact

createContact компонента в себе си съдържа една форма от четири полета и бутон за добавяне на нов контакт. Освен това във същия компонент е регистрирано и събитието **refreshTableEvent**, което възниква всеки път когато успешно е запазен нов контакт в Salesforce.

A screenshot of the Salesforce user interface showing the 'Create Contact' form. The top navigation bar includes the Salesforce logo, a search bar, and various utility icons. Below the navigation bar, a menu contains links to Sales, Home, Opportunities, Leads, Tasks, Files, Accounts, and Contacts. The 'Contacts' link is selected, and a sub-tab labeled 'Contacts Example Tab' is active. The main content area displays a form with four input fields: 'First Name' (marked with an asterisk), 'Last name' (marked with an asterisk), 'Phone', and 'Email'. A 'Save Contact' button is located at the bottom of the form.

## contactsTable

contactsTable компонента е изграден от html таблица, и показва данните върнати от базата. В този компонент са намират обработчиците(handlers) на събитията които възникват.

Събитията които се обработват в този компонент са две:

- Събитието което възниква при зареждане на страницата
- Събитието което възниква когато един контакт е записан успешно в Salesforce.

**Init** или събитието което възниква при зареждане на страницата има за цел да направи връзка със сървър контролера и да извлече всички контакти които се намират в тази Salesforce инстанция и да ги визуализира. След като ги зареди контактите се сетват в един атрибут който се итерира в таблицата за да ги нарисува на екрана.

Рисуването на контактите на екрана става с помощ на стандартния компонент `aura:iteration`.

**refreshTableEvent** е събитието което възниква когато един контакт е записан успешно в Salesforce. Това събитие е от тип `application`.

Обработчика на това събитие има за цел да презареди таблицата с новите записи които са били добавени в базата.

NAME	TITLE	PHONE	EMAIL
Tom Ripley	Regional General Manager	(650) 450-8810	triplep@uog.com
Liz D'Cruz	VP, Production	(650) 450-8810	ldcruz@uog.com
Edna Frank	VP, Technology	(650) 867-3450	efrank@genepoint.com
Avi Green	CFO	(212) 842-5500	agreen@uog.com
Siddhartha Nedaerk			
Jake Llorac			
Rose Gonzalez	SVP, Procurement	(512) 757-6000	rose@edge.com
Sean Forbes	CFO	(512) 757-6000	sean@edge.com
Jack Rogers	VP, Facilities	(336) 222-7000	jrogers@burlington.com
Pat Stumuller	SVP, Administration and Finance	(014) 427-4427	pat@pyramid.net
Andy Young	SVP, Operations	(785) 241-6200	a_young@dickenson.com
Tim Barr	SVP, Administration and Finance	(312) 596-1000	barr_tim@grandhotels.com
John Bond	VP, Facilities	(312) 596-1000	bond_john@grandhotels.com
Stella Pavlova	SVP, Production	(212) 842-5500	spavlova@uog.com
Lauren Boyle	SVP, Technology	(212) 842-5500	lboyle@uog.com
Babara Levy	SVP, Operations	(503) 421-7800	b.levy@expressi&t.net
Josh Davis	Director, Warehouse Mgmt	(503) 421-7800	j.davis@expressi&t.net
Jane Grey	Dean of Administration	(520) 773-9050	jane_gray@uoa.edu
Arthur Song	CEO	(212) 842-5500	asong@uog.com
Ashley James	VP, Finance	+44 191 4956203	ajames@uog.com