

# Kursprogram Introduktion till tjänstebaserade affärssystem (SAFFK-INTRO)

## Labb 4

**Hösten 2018**

**7,5 hp**

### Goal

- To get started with apex,
- To be able to create pages.
- To be able to display contact details using apex
- To be able to edit contact details using apex

### Learning outcomes

After taking this laboratory exercise you will learn how to get started with creating pages, developing your page using apex, and learning some aspects of the apex syntax. You will also learn the Model View Controller in respect to visualforce architecture.

### Overview

This manual contains two sections **Section 1: The basics** – is introduction to concepts that will help you to grasp the concepts related to Force.com and apex. Please read section one before you continue to section 2. **Section 2** is hands on demonstration and exercise.

**Good Luck!**

[workneh@dsv.su.se](mailto:workneh@dsv.su.se)

Workneh Yilma Ayele

# 1.Theoretical discussion

## 1.Introduction to Visualforce

*Some part of this theoretical section is based on the electronic learning documents form Salesforce.com*

Salesforce has a comprehensive platform for building on-demand applications. Like other sophisticated application development platform, the Force.com platform offers separate tools for defining the structure of the data (Data Model), the rules that detail how that data can be manipulated (Business Logic), and the layout that specify how that data should be displayed (User Interface).



Model View Controller (MVC)

- Model – is the data model
- View – is the user interface
- Controller – is the business logic

Splitting up application development tools based on whether they affect the data model, business logic, or user interface is also known as the Model-View-Controller (MVC) application development pattern—the Model is the data model, the View is the user interface, and the Controller is the business logic.

## 2.What is a Visualforce Page?

Developers can use Visualforce to create a Visualforce page definition. A page definition consists of two primary elements:

- Visualforce markup
- A Visualforce controller

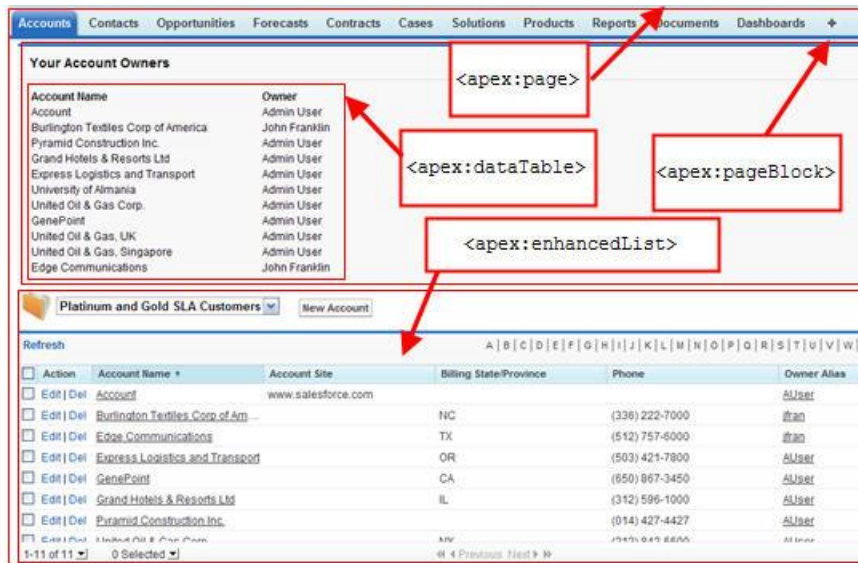
Visualforce is a framework that allows developers to build sophisticated, custom user interfaces that can be hosted natively on the Force.com platform. Visualforce is a tool which allows you to describe the user interface components that live on your Force.com pages.

Visualforce is a framework that allows developers to build sophisticated, custom user interfaces that can be hosted natively on the Force.com platform. The Visualforce framework includes a tag-based markup language, similar to HTML, and a set of server-side “standard controllers” that make basic database operations, such as queries and saves, very simple to perform (quick).

In the Visualforce markup language, each Visualforce tag corresponds to a coarse or fine-grained user interface component, such as a section of a page, a related list, or a field. The behavior of Visualforce components can either be controlled by the same logic that is used in standard Salesforce pages, or developers can associate their own logic with a controller class written in Apex (quick).

Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Force.com platform server in conjunction with calls to the Force.com API. Using syntax that looks like Java and acts like database stored procedures, Apex enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. Apex code can be initiated by Web service requests and from

triggers on objects<sup>1</sup>. Therefore Apex is a development platform to build software-as-a-service (SaaS) software on the top of Salesforce.com.



## a. Visualforce Markup

Visualforce markup consists of Visualforce tags, HTML, JavaScript, or any other Web-enabled code embedded within a single `<apex:page>` tag. The markup defines the user interface components that should be included on the page, and the way they should appear.

## b. Visualforce Controllers

A Visualforce controller is a set of instructions that specify what happens when a user interacts with the components specified in associated

Visualforce markup, such as when a user clicks a button or link. Controllers also provide access to the data that should be displayed in a page, and can modify component behavior.

A developer can either use a standard controller provided by the Force.com platform, or add custom controller logic with a class written in Apex:

- A **standard controller** consists of the same functionality and logic that is used for a standard Salesforce page. For example, if you use the standard Accounts controller, clicking a **Save** button in a Visualforce page results in the same behavior as clicking **Save** on a standard Account edit page. If you use a standard controller on a page and the user doesn't have access to the object, the page will display a insufficient privileges error message. You can avoid this by checking the user's accessibility for an object and displaying components appropriately.
- A standard list controller enables you to create Visualforce pages that can display or act on a set of records. Examples of existing Salesforce pages that work with a set of records include list pages, related lists, and mass action pages.
- A custom controller is a class written in Apex that implements all of a page's logic, without leveraging a standard controller. If you use a custom controller, you can define new

<sup>1</sup> [https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex\\_intro\\_what\\_is\\_apex.htm](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_intro_what_is_apex.htm)

navigation elements or behaviors, but you must also re-implement any functionality that was already provided in a standard controller.

Like other Apex classes, custom controllers execute entirely in system mode, in which the object and field-level permissions of the current user are ignored. You can specify whether a user can execute methods in a custom controller based on the user's profile.

- A controller extension is a class written in Apex that adds to or overrides behavior in a standard or custom controller. Extensions allow you to leverage the functionality of another controller while adding your own custom logic.

Because standard controllers execute in user mode, in which the permissions, field-level security, and sharing rules of the current user are enforced, extending a standard controller allows you to build a Visualforce page that respects user permissions. Although the extension class executes in system mode, the standard controller executes in user mode. As with custom controllers, you can specify whether a user can execute methods in a controller extension based on the user's profile.

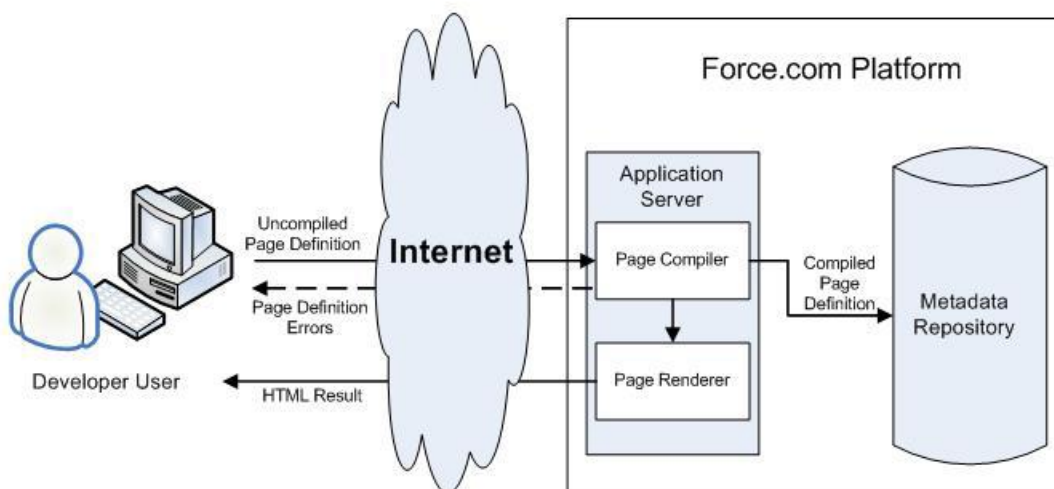
### c. Where Can Visualforce Pages Be Used?

Developers can use Visualforce pages to:

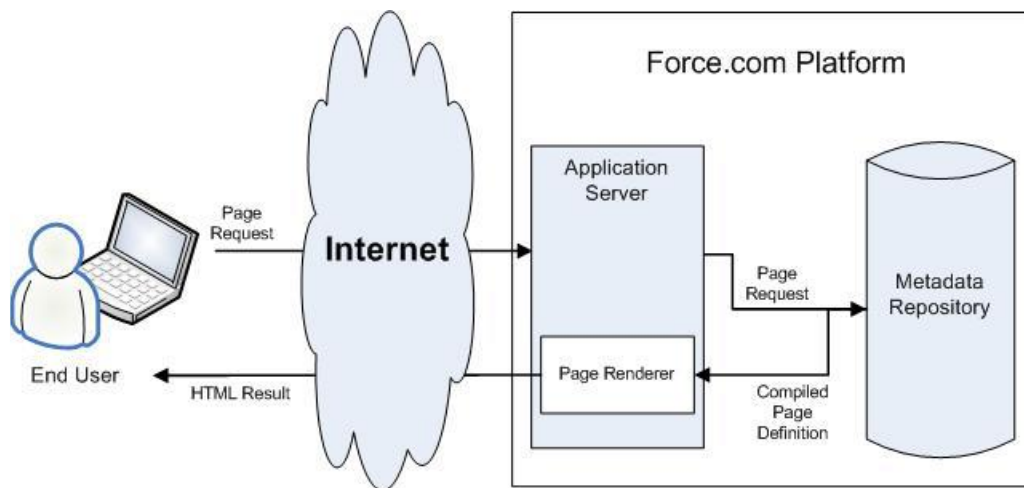
- Override standard buttons, such as the New button for accounts, or the Edit button for contacts
- Override tab overview pages, such as the Accounts tab home page
- Define custom tabs
- Embed components in detail page layouts
- Create dashboard components or custom help pages
- Customize, extend, or integrate the sidebars in the Salesforce console (custom console components)
- Add menu items, actions, and mobile cards in Salesforce1

## 3.How is Visualforce Architected?

### Visualforce System Architecture – Development Mode



## Visualforce System Architecture – Standard User Mode



### a. What are the Benefits of Visualforce?

- User-friendly development
- Integration with other Web-based interface technologies
- Model-View-Controller(MVC) style development
- Concise syntax
- Visualforce components are rendered intelligently by the platform. For example, rather than forcing page designers to use different
- Component tags for different types of editable fields (such as email addresses or calendar dates), designers can simply use a generic `<apex:inputField>` tag for all fields. The Visualforce renderer displays the appropriate edit interface for each field.
- Hosted platform
- Automatically upgradeable

### b. When Should I Use Visualforce?

The Salesforce prebuilt applications provide powerful CRM functionality. In addition, Salesforce provides the ability to customize the prebuilt applications to fit your organization. However, your organization may have complex business processes that are unsupported by the existing functionality. When this is the case, the Force.com platform includes a number of ways for advanced administrators and developers to implement custom functionality. These include Visualforce, Apex, and the SOAP API.

## 4. Visualforce

Visualforce consists of a tag-based markup language that gives developers a more powerful way of building applications and customizing the Salesforce user interface. With Visualforce you can:

- Build wizards and other multistep processes.
- Create your own custom flow control through an application.
- Define navigation patterns and data-specific rules for optimal, efficient application interaction.

## a. Apex

Use Apex if you want to:

Introducing Visualforce When Should I Use Visualforce?

- Create Web services.
- Create email services.
- Perform complex validation over multiple objects.
- Create complex business processes that are not supported by workflow.
- Create custom transactional logic (logic that occurs over the entire transaction, not just with a single record or object).
- Attach custom logic to another operation, such as saving a record, so that it occurs whenever the operation is executed, regardless of whether it originates in the user interface, a Visualforce page, or from SOAP API.

## 5. Tools for Visualforce Development

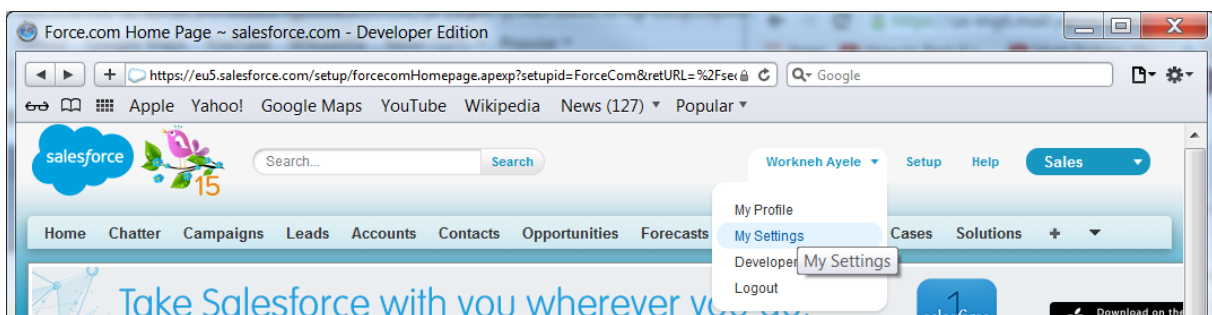
1. You can develop using developer console by choosing: Select Developer console from your name next to Setup
2. Use the page editor as you will see in the next section
3. Using Eclipse IDE (integrated development environment)

## 2. Practical section

### 1. Setting Permissions that are required for Salesforce development

Enable developer mode

- You need to enable development mode on your user record to use the inline page editor
- Select the arrow next to your name
- Select **My Settings**



- Then select Personal information



Edit my personal information

Personal > [Personal Information](#)

- Then select **Advanced User Details** under **My Settings** bar on the left side
- Select **Edit**
- Now you can check **Developer Mode** if it is not already selected then click **Save**

## 2. Creating salesforce page with Visualforce

### Do the following

#### Create an account with the following information

Account Name - GPS Assembler - yourFirstNames

Account Number - 01222767787

Account Site - dsv.su.se

Type - select "Installation Partner"

Industry - Communications

Annual Revenue - 12,5266,566

Phone - 081616161616

Billing Street - Emmylundsvägen 5X LGH 1611

Shipping Street - Emmylundsvägen 5X LGH 1611

Billing City - Solna

Shipping City - Solna

Billing Zip/Postal Code - 17172

Shipping Zip/Postal Code - 17172

Billing Country - Sweden

Shipping Country - Sweden

#### Create the following contacts for this account:

##### **a. Contact 1:**

**First Name** – Ms. Ruth, **Last Name** – David, **Account Name** - GPS Assembler – fristNames, **Phone** – 07005689999, **Email** – [ruth@xyz.com](mailto:ruth@xyz.com), **Mailing Street** - Emmylundsvägen 5X LGH 1611, **Mailing City** – Solna, **Mailing Zip** – 17172, **Mailing Country** – Sweden

##### **b. Contact 2:**

**First Name** – Mr. Joshua, **Last Name** – Goldberg, **Account Name** - GPS Assembler –firstName, **Phone** – 07005689777, **Email** – [joshua@xyz.com](mailto:joshua@xyz.com), **Mailing Street** - Emmylundsvägen 5 LGH 1611, **Mailing City** – Solna, **Mailing Zip** – 17172, **Mailing Country** – Sweden

##### **c. Contacts**

Add contacts with all group members (with your name, and so on)

## a. Creating new page option 1

### Creating and editing pages

Visualforce URL for a page always begins with a base URL for your salesforce instance then

“/apex” followed by and the “/name\_of\_the\_page”

**Example** - Visualforce page URL structure is Base URL + /apex + / {Page Name}

Visualforce pages are actually served up on a different domain to the rest of Salesforce

The URL when you create it: <https://eu5.salesforce.com/apex/MyPage> - here MyPage is your page

The URL after you created it: <https://c.eu5.visual.force.com/apex/MyPage>

### Steps

Inspect the URL: For example

<https://eu5.salesforce.com/005240000016JbZ?noredirect=1&retURL=%2Fui%2Fsetup%2FSetup%3Fsetupid%3DPersonalInfo&setupid=AdvancedUserDetails>

Remove the highlighted part of the **URL** and add - /apex/erpatdsv

<https://eu5.salesforce.com/apex/erpatdsv> then press enter to check if the URL exists

Then your browser will show you that the page doesn't exist if it exists



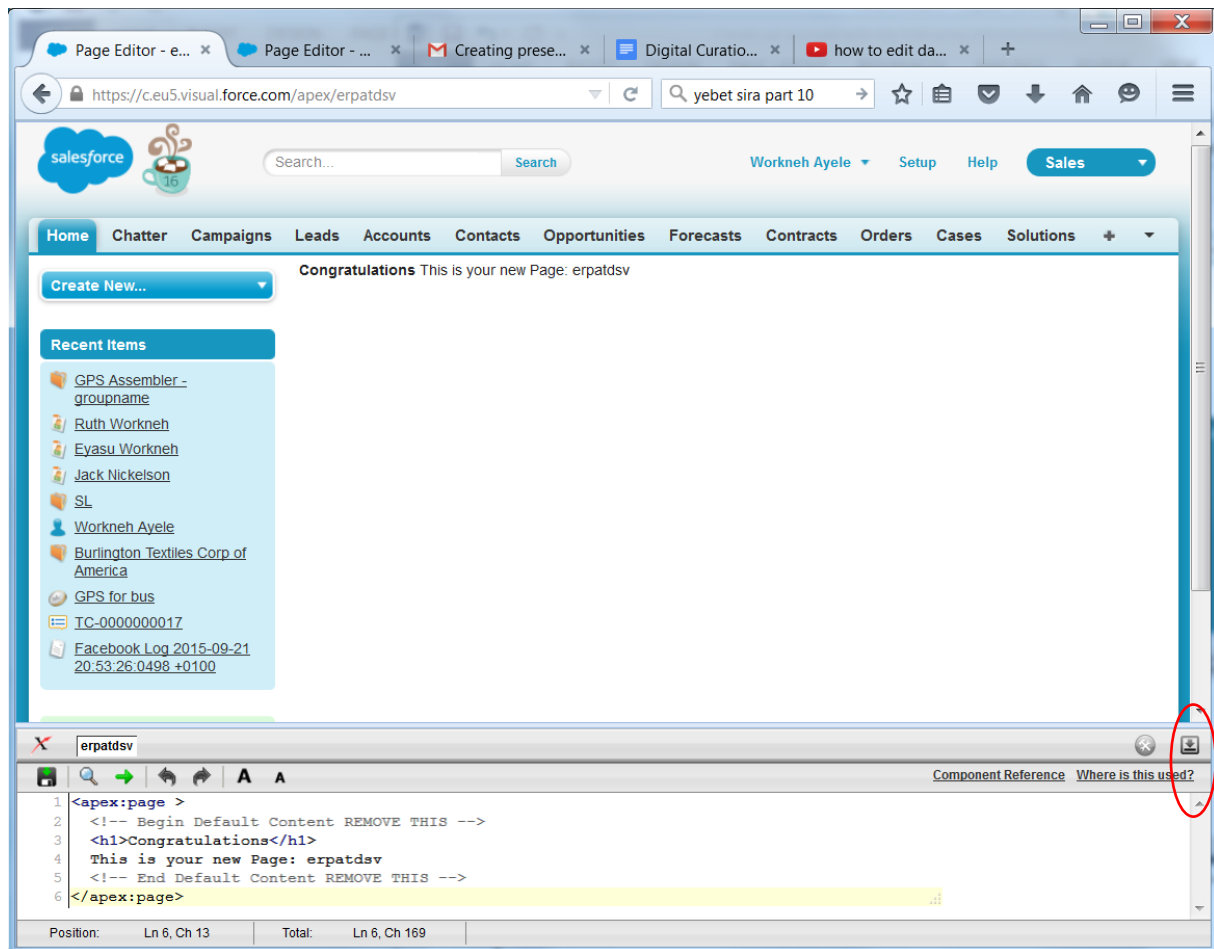
Or



Click the link **Create Page erpatdsv** and the page will be created accordingly

Select the arrow at the bottom right corner of your browser window (circled with red oval as illustrated in the following diagram) – to display the editor window as displayed below





### Code 1:

```
<apex:page >
  <!-- Begin Default Content REMOVE THIS -->
  <h1>Congratulations</h1>
  This is your new Page: erpatdsv
  <!-- End Default Content REMOVE THIS -->
</apex:page>
```

### Accessing data (for example data for Contact Can be Email, Phone and so on)

Similar to mail-merge in Word, placeholders in pages are replaced with real data when a page is viewed

Most basic syntax for data access placeholder is:

- `{!Object.Field}`

Example accessing data using the Account standard controller:

- `{!Account.Name}`
- `{!Contact.Name}`

- {!Contact.Phone}

Replace the code displayed above **Code 1:** with the following code **Code 2:**

**Code 2:**

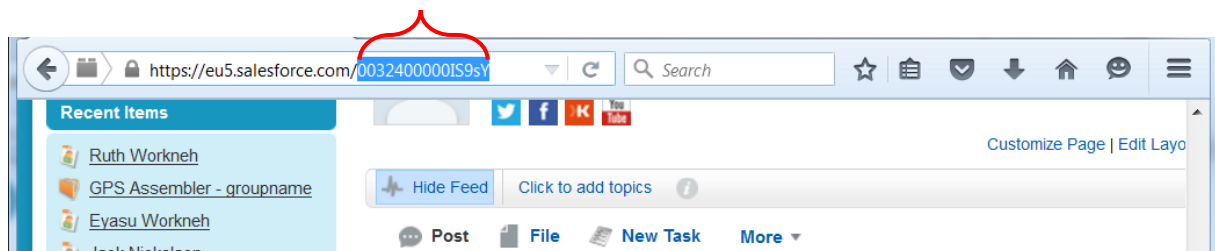
```
<apex:page standardController="Contact">
  <apex:form >
    <apex:pageBlock title="Quick Edit: {!Contact.Name}">
      <apex:pageBlockSection title="Contact Details" columns="1">
        <apex:inputField value="{!Contact.Title}"/>
        <apex:inputField value="{!Contact.Phone}"/>
        <apex:inputField value="{!Contact.MailingStreet}"/>
        <apex:inputField value="{!Contact.MailingCity}"/>
        <apex:outputField value="{!Contact.MobilePhone}" label="Mobile #"/>
        <apex:inputText value="{!Contact.Email}" label="{!Contact.FirstName} + 's Email'"/>
      </apex:pageBlockSection>
      <apex:pageBlockButtons >
        <apex:commandbutton action="{!quicksave}" value="Quick Save"/>
        <apex:commandbutton action="{!save}" value="Save"/>
      </apex:pageBlockButtons>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

**Quick Save** is used to save and to stay with your page, while **Save** will take your new or updated value and displays the actual Contact page of Salesforce

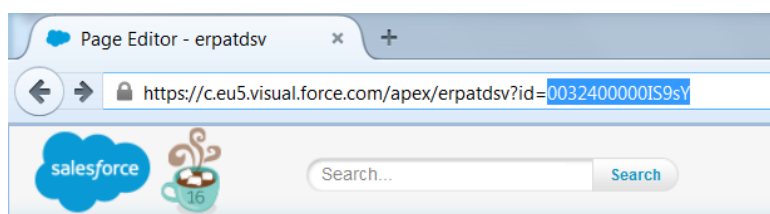
Nothing came up on the screen so we need to use page parameters – parameters are used to provide information to the server

Go to Salesforce and access the contact Ruth in the same account

Copy the contact id as shown below



If you want to display an existing contact, that is Ruth's contact, then you need to specify id of the contact as a parameter, after this you should be able to edit the contact details of Ruth



## Illustration

```
<apex:page standardController="Contact">
  <apex:form >
    <apex:pageBlock title="Quick Edit: {!Contact.Name}">
      <apex:pageBlockSection title="Contact Details" columns="1">
        <apex:inputField value="{!Contact.Title}"/>
        <apex:inputField value="{!Contact.Phone}"/>
        <apex:inputField value="{!Contact.MailingStreet}"/>
        <apex:inputField value="{!Contact.MailingCity}"/>
        <apex:outputField value="{!Contact.MobilePhone}" label="Mobile #"/>
        <apex:inputText value="{!Contact.Email}" label="{!Contact.FirstName + 's Email'}"/>
      </apex:pageBlockSection>
      <apex:pageBlockButtons >
        <apex:commandbutton action="{!quicksave}" value="Quick Save"/>
        <apex:commandbutton action="{!save}" value="Save"/>
      </apex:pageBlockButtons>
    </apex:pageBlock>
  </apex:form>
</apex:page>
```

**Page tag – to create the page**  
 <apex:page>  
 </apex:page>

**Pageblocksection tag – to create the block for the Title, Phone ...**  
 <apex: pageBlockButtons >  
 </apex: pageBlockButtons >

**Commandbutton tag – to create the block for the buttons**  
 <apex: pageBlockButtons >  
 </apex: pageBlockButtons >

Exercise – modify this code illustrated above to display contact name, when you are done take a snapshot or demonstrate to your teacher when you are done.

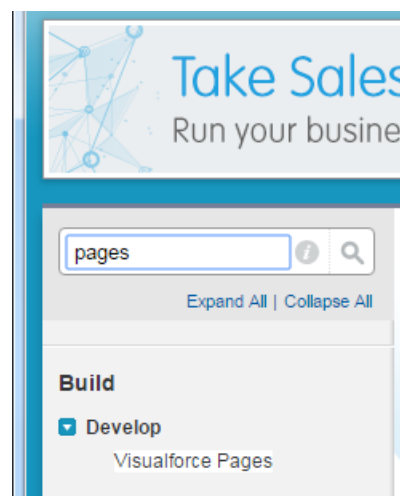
## Creating a page with custom controller option 2

Select **Setup**

Go to **Develop**

Select **Visualforce Pages**

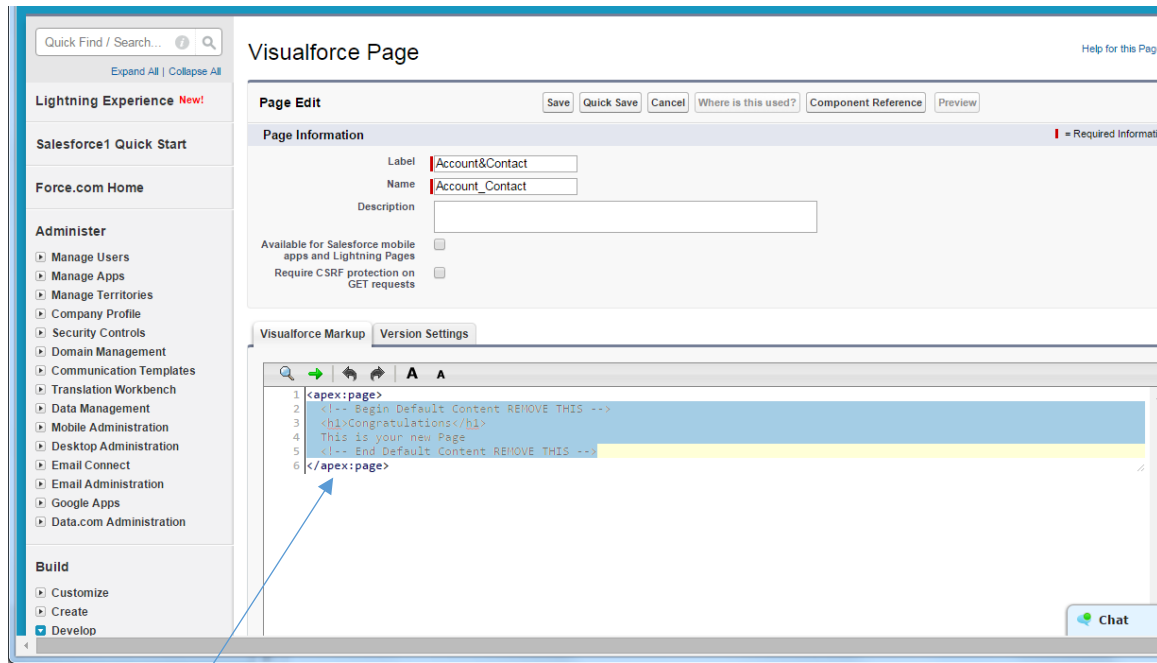
Select **New** button



A   B   C   D   E   F   G   H   I   J   K   L   M					
		Developer Console		New	
Label	Name	Namespace Prefix	Api Version	Description	Created By A

Under Label enter "Account\_Contacts"

Under Name enter "Account\_Contact"



This is where you enter your apex code to change the look and feel of your new page

Modify the apex code as follows

Every code you need to write to create a page has to be written between "<apex:page>" opening tag and "</apex:page>" closing tag

Change the </apex:page> as illustrated below, the script to be added is in red font

<apex:page standardController="Account" extensions="Account&ContactsController" tabStyle="Contact">

Click **Quick Save** (don't click **Save** now)

What do you see? There must be an error message

## Visualforce Page

**Page Edit** [Save](#) [Quick Save](#) [Cancel](#) [Where is this used?](#) [Component Reference](#) [Preview](#)

**Error:** The name can only contain underscores and alphanumeric characters. It must begin with a letter and be unique, and must not include underscore, or contain two consecutive underscores.

**Page Information**

Label   
Name   
Description

Available for Salesforce mobile apps and Lightning Pages ☐  
Require CSRF protection on GET requests ☐

**Visualforce Markup** **Version Settings**

```
1 <apex:page standardController="Account" extensions="Account&ContactsController" tabStyle="Contact">
2 <!-- Begin Default Content REMOVE THIS -->
3 <h1>Congratulations</h1>
4 This is your new Page
5 <!-- End Default Content REMOVE THIS -->
6 </apex:page>
```

You can now edit the name under extension to “Account\_ContactsController”

Click **Quick Save**

Quick Find / Search... [Expand All](#) [Collapse All](#)

**Lightning Experience** **New!**

**Salesforce1 Quick Start**

**Force.com Home**

**Administer**

- Manage Users
- Manage Apps
- Manage Territories
- Company Profile
- Security Controls
- Domain Management
- Communication Templates
- Translation Workbench
- Data Management
- Mobile Administration
- Desktop Administration
- Email Connect
- Email Administration
- Google Apps
- Data.com Administration

**Visualforce Page** [Help for this Page](#)

**Page Edit** [Save](#) [Quick Save](#) [Cancel](#) [Where is this used?](#) [Component Reference](#) [Preview](#)

**Error:** Apex class 'Account\_ContactsController' does not exist [Create Apex class 'public with sharing class Account\\_ContactsController'](#) [Create Apex class 'public class Account\\_ContactsController'](#)

**Page Information** ! Required Information

Label   
Name   
Description

Available for Salesforce mobile apps and Lightning Pages ☐  
Require CSRF protection on GET requests ☐

**Visualforce Markup** **Version Settings**

```
1 <apex:page standardController="Account" extensions="Account_ContactsController" tabStyle="Contact">
2 <!-- Begin Default Content REMOVE THIS -->
3 <h1>Congratulations</h1>
4 This is your new Page
5 <!-- End Default Content REMOVE THIS -->
6 </apex:page>
```

[Create Apex class 'public with sharing class Account\\_ContactsController'](#)

[Create Apex class 'public class Account\\_ContactsController'](#)

Select the first one **Create Apex class 'public with sharing class Account\_ContactsController'**

**You will have another error**

**To fix it Click on**

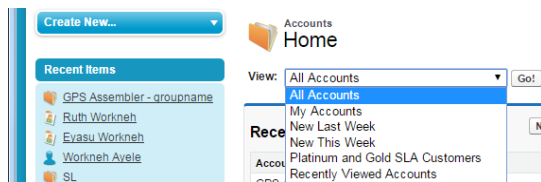
[Create Apex method 'Account\\_ContactsController.Account\\_ContactsController\(ApexPages.StandardController controller\)'](#)

Click on **Quick Save**

If the quick save works you will have no error

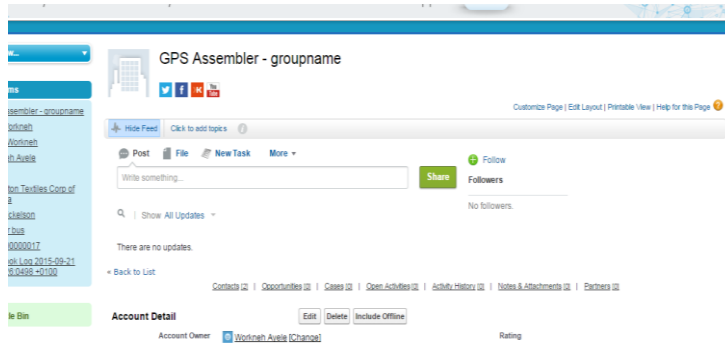
If there is no error like before click on **Accounts** tab,

To view all comments select **All Accounts**



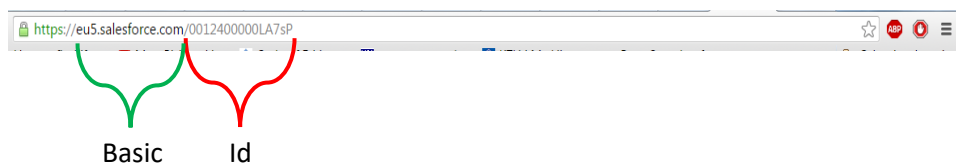
Click **Go**

Select the account you created before “GPS Assembler - groupname”



The URL text indicated below with red brace is an id of the account “GPS Assembler – groupname”

The URL portion indicated with green brace is the basic URL which varies with regions or continents

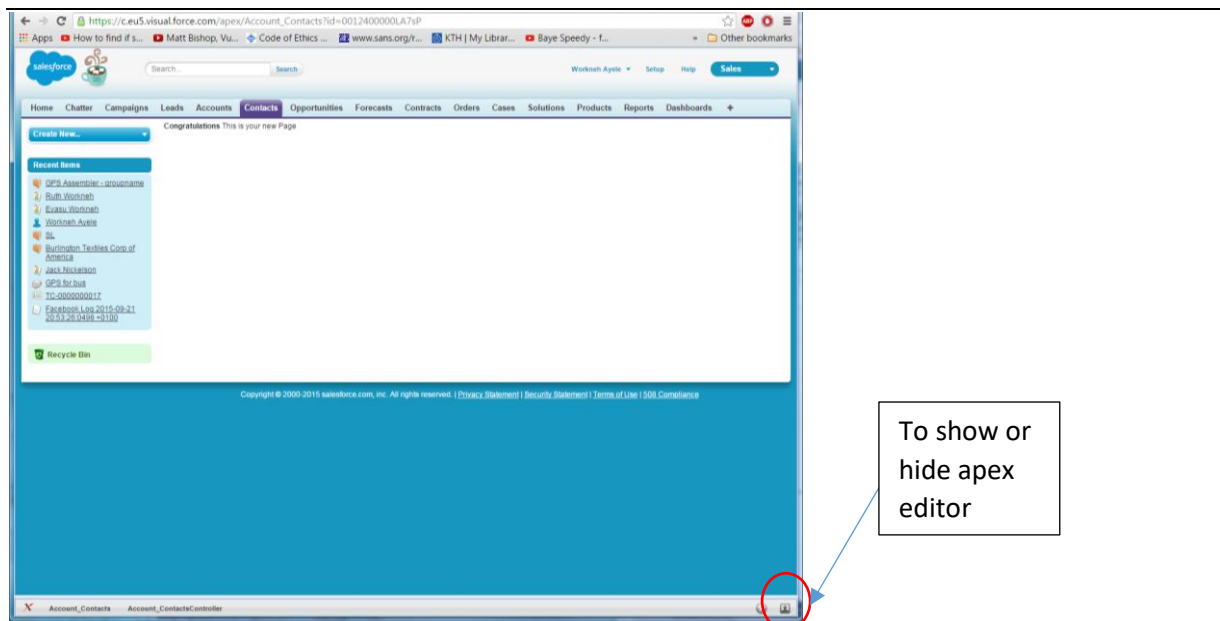


Now to go back to the page you created and to load the new page with your new account “GPS Assembler - groupname”

*Don't delete the ID but modify the URL as shown below by inserting the portion of the URL in red font if the base URL is similar to your URL*

[https://eu9.salesforce.com/apex/Account\\_Contacts?id=YouIDasIndicatedInTheDiagramUP](https://eu9.salesforce.com/apex/Account_Contacts?id=YouIDasIndicatedInTheDiagramUP)

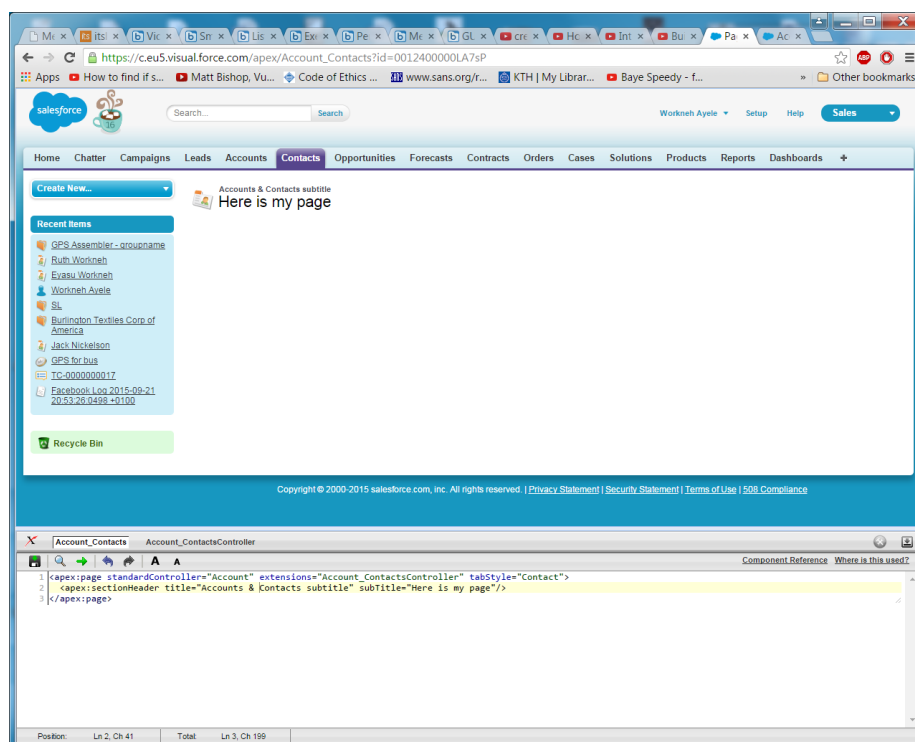
Now you will see the new page without content so click on show editor as shown below



Now enter the following code to add section header to your new page: as shown below then click **Save** button

```
1 <apex:page standardController="Account" extensions="Account_ContactsController" tabStyle="Contact">
2   <apex:sectionHeader title="Accounts & Contacts subtitle" subTitle="Here is my page"/>
3 </apex:page>
```

You will see the new page with title and subtitle



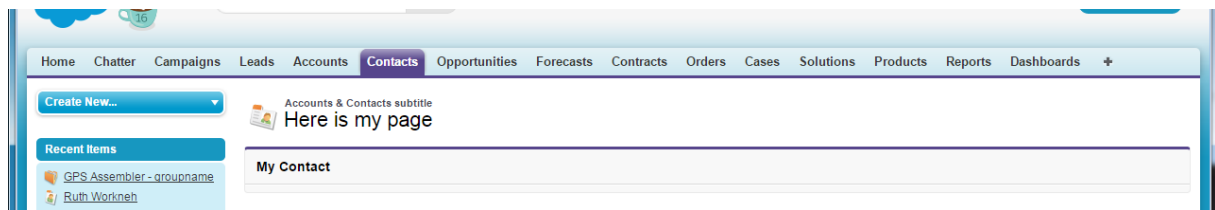
Now add pageBlock with title "My Contacts"

```

<apex:page standardController="Account" extensions="Account_ContactsController" tabStyle="Contact">
  <apex:sectionHeader title="Accounts & Contacts subtitle" subTitle="Here is my page"/>
  <apex:pageBlock title="My Contact">
    </apex:pageBlock>
  </apex:page>

```

Add this



Add "apex:pageBlockTable" to create .... As shown below, this must be between the <apex:pageBlock> and </apex:pageBlock> as shown below and click the **Save** button

You will have an error as shown below

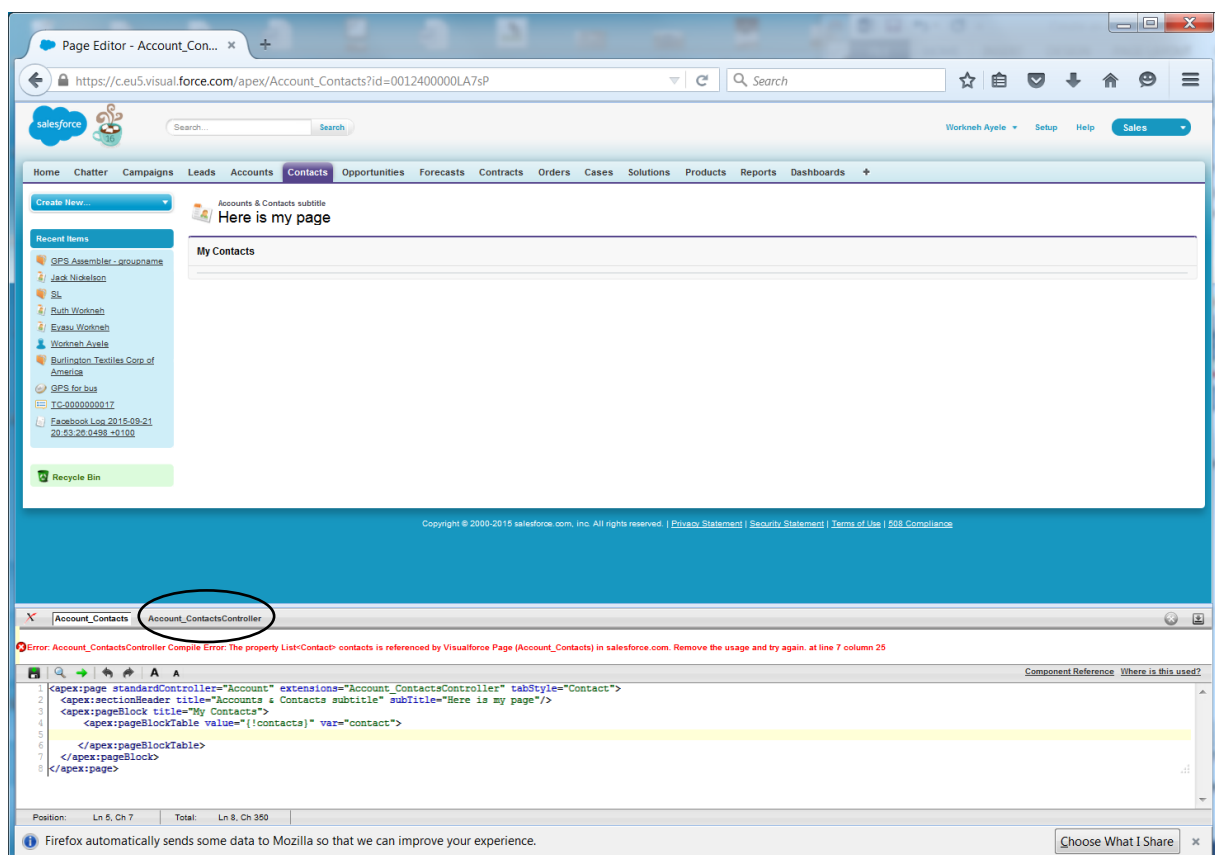
```

1 <apex:page standardController="Account" extensions="Account_ContactsController" tabStyle="Contact">
2 <apex:sectionHeader title="Accounts & Contacts subtitle" subTitle="Here is my page"/>
3 <apex:pageBlock title="My Contact">
4   <apex:pageBlockTable value="{!contacts}">
5   </apex:pageBlockTable>
6 </apex:pageBlock>
7 </apex:page>

```

Error

Add this

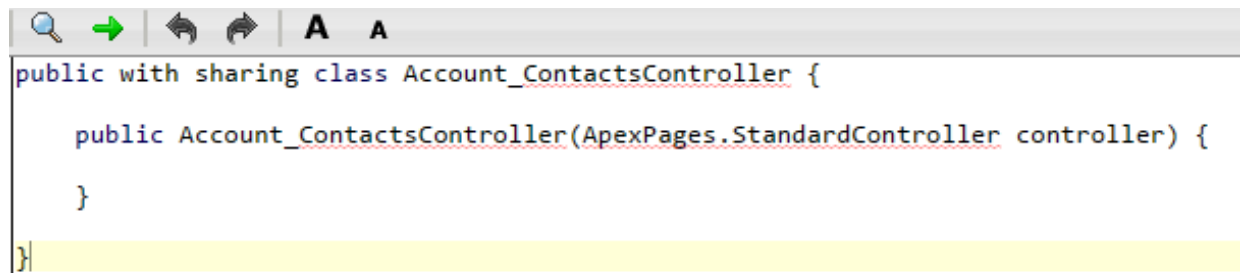


To fix the error

Click **Account\_ContactsController** just above the error message



You will see code similar to java with class declaration



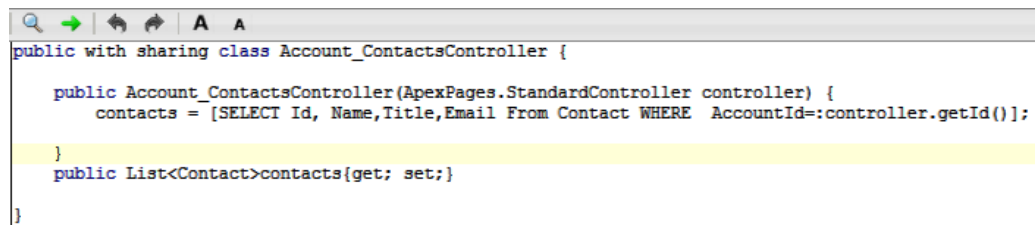
```
public with sharing class Account_ContactsController {  
    public Account_ContactsController(ApexPages.StandardController controller) {  
    }  
}
```

[https://c.eu9.visual.force.com/apex/Account\\_Contacts?id=0012400000LA7sP](https://c.eu9.visual.force.com/apex/Account_Contacts?id=0012400000LA7sP)

Your url link will look similar to the above

**Add** `contacts = [SELECT Id, Name, Title, Email From Contact WHERE AccountId=:controller.getId()];`

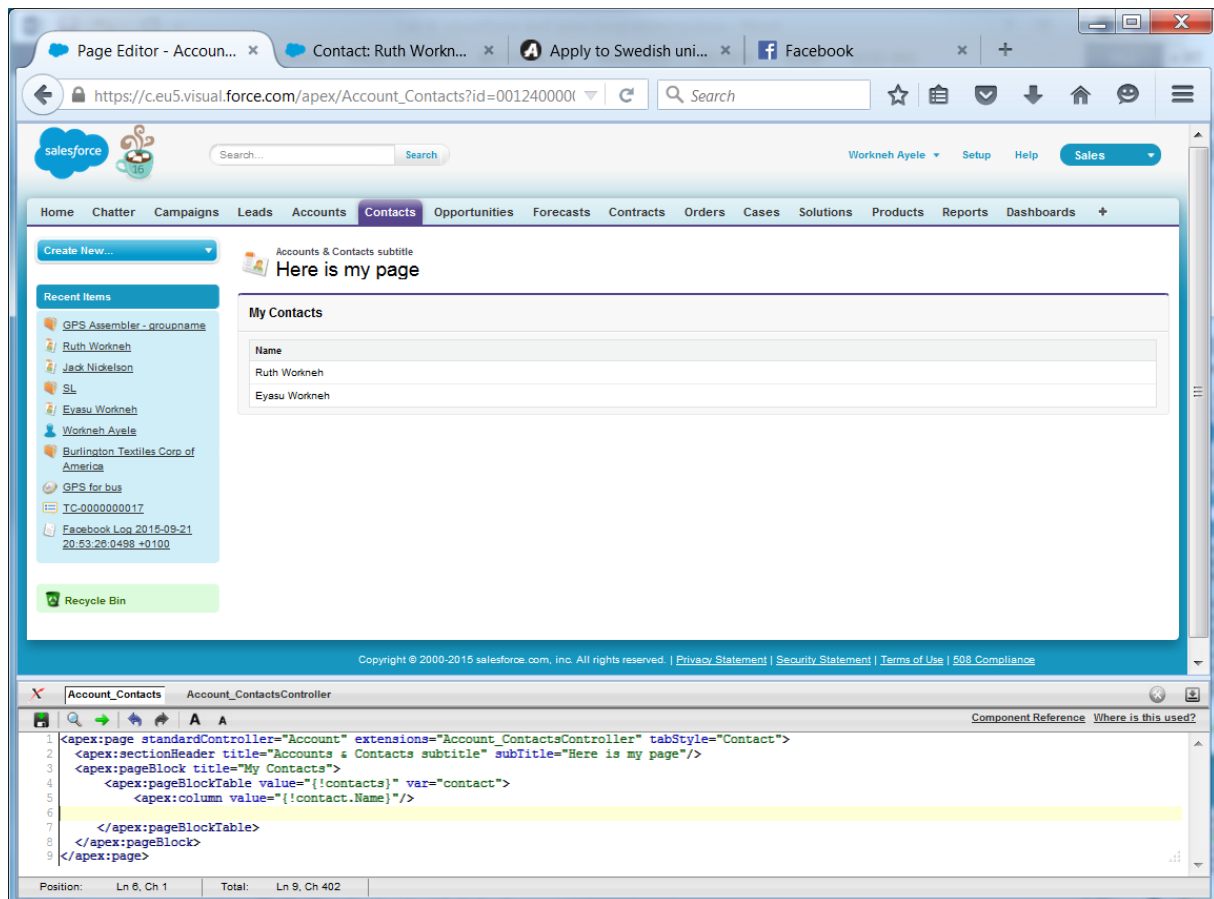
**Add** `List<Contact>contacts{get; set;}` as illustrated below



```
public with sharing class Account_ContactsController {  
    public Account_ContactsController(ApexPages.StandardController controller) {  
        contacts = [SELECT Id, Name, Title, Email From Contact WHERE AccountId=:controller.getId()];  
    }  
    public List<Contact>contacts{get; set;}  
}
```

Go to Accounts\_Contacts on the code view down

Enter the following code is added to create columns with name,....

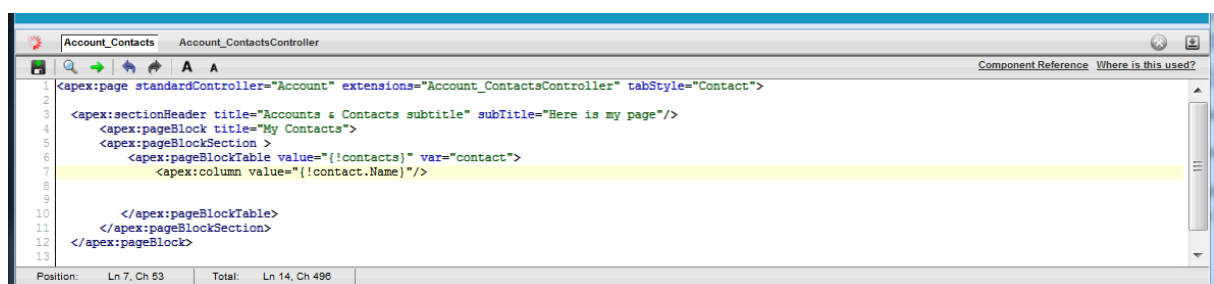


Apex:pageBlockSection

<apex:column value="{!contact.Email}"/>

With the followign code the page looks different with apex:pageBlockSection, note the indentation because tags with in tags should be indented.

**Exercise:** Take snapshot and save it to your desktop



The page will look like the following

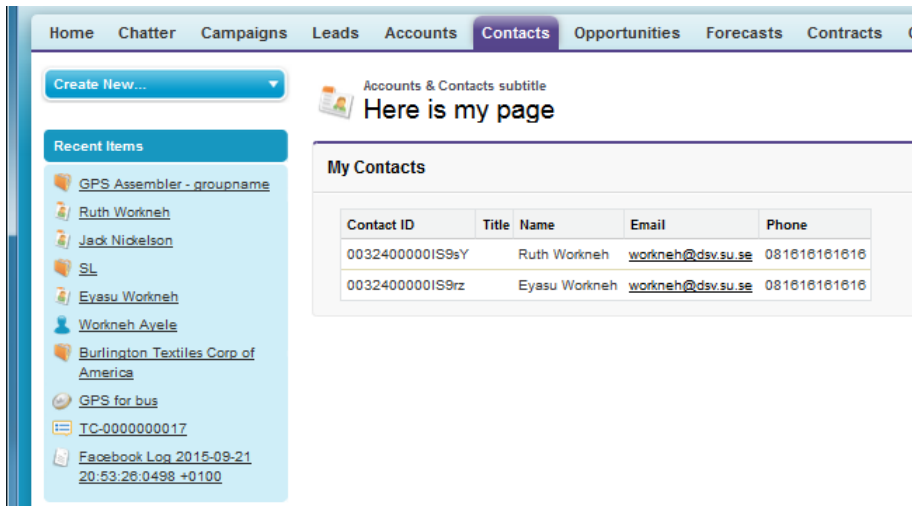


**Exercise:** Take snapshot and save it to your desktop

**Exercise:** do the following by yourself

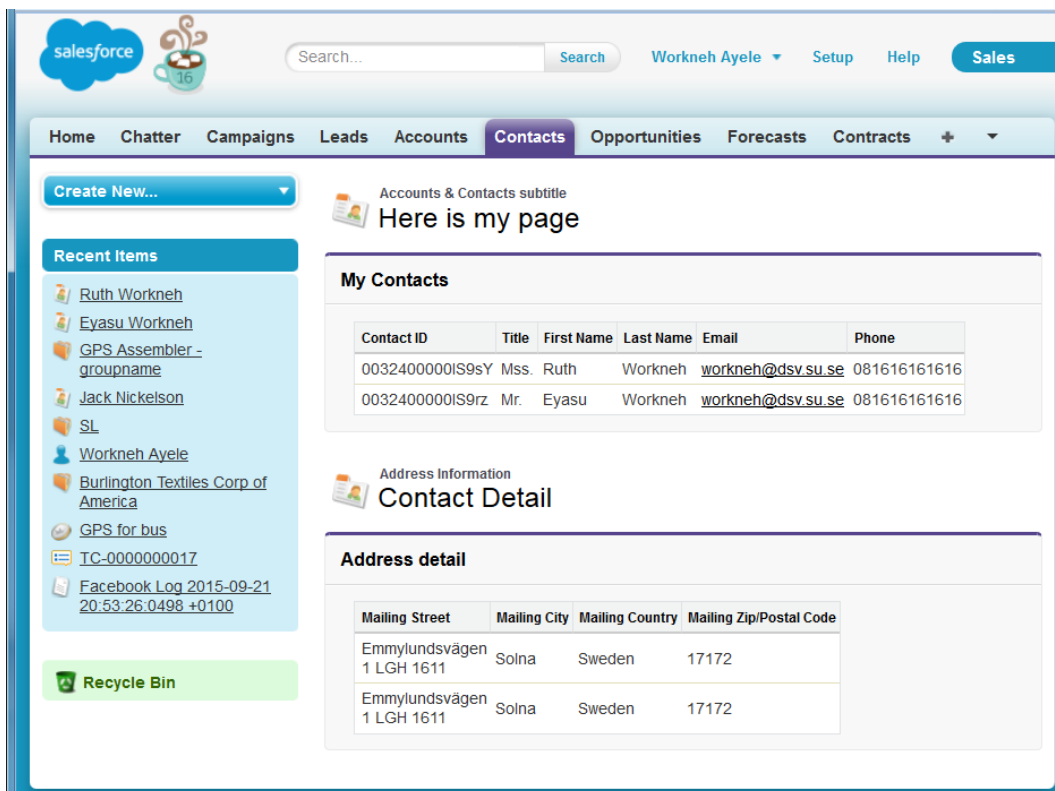
Add (id and title) before name, and (Email and phone) after name

The output should look like the following



**Exercise:** Take snapshot and save it to your desktop

**Exercise :** Modify the code so that your page looks like the following, show this one to your teacher.



**Assessment:**

Explain what you learnt from this beginner level exercise to your teacher

Show the snapshots you have taken

