Sencha

First Steps with Sencha Ext JS 7.8+

Installation, Templates, and Components

```
const bookInfo = {
     title:    "First Steps with Sencha Ext JS 7.8+",
     subtitle: "Installation, Templates, and Components",
     authors: [
      {
          name: "Andres Villalba",
          role: "Sencha Sales Engineer"
     },
     {
          name: "Kevin Avila",
          role: "Web Developer"
     }
     ],

     edition:  “First Edition, 2025”
};
console.log(bookInfo);
```

Table of Contents

01 Introduction

Sencha Ext JS is a JavaScript framework designed for front-end web development. It stands out for its versatility and power, offering over 140 ready-to-use components for web projects. This framework is especially efficient for building robust, high-performance enterprise applications, providing all the tools needed to develop the front end of modern web applications.

Ext JS employs JavaScript as its programming language, with a declarative approach to view creation. Its object-based architecture facilitates the use of patterns such as MVC and MVVM.

Depending on the license acquired, Ext JS includes a range of additional tools and utilities that enhance its capabilities:

- **Sencha Command**
  Command-line tool for creating, minifying, building, and testing applications.
- **Sencha Architect**
  Low-code editor that enables Ext JS application development in an intuitive visual environment.
- **Sencha Themer**
  Graphical tool for designing and customizing visual themes.
- **Rapid Ext JS**
  Microsoft Visual Studio Code Add-on that simplifies the creation of Ext JS views, and adds a property inspector.
- **ReExt**
  A component that integrates Ext JS components into React-based projects.

Toolkits

Ext JS includes two **toolkits** that share the same core but differ in focus and optimization:

- **Modern Toolkit**
  Uses CSS to render application views and is optimized for mobile devices, such as tablets and smartphones.
- **Classic Toolkit**
  Primarily uses JavaScript to render views and is optimized for web applications running on desktop devices.

It is important to note that while each toolkit may be better suited to certain devices, both can be used on any platform. For instance, the Classic Toolkit is not limited to desktop devices, nor is the Modern Toolkit exclusively for mobile devices. The choice of toolkit depends on the specific needs of the application.

The Sencha Difference

Why choose Sencha Ext JS over other front-end frameworks?

1. **Comprehensive Framework**
   Ext JS is a complete, ready-to-use solution that includes integrated tools and features. It provides everything you need, such as UI components (grids, forms, charts), data handling, and themes. In contrast, other frameworks often serve as a core foundation requiring additional libraries to achieve similar functionality.
2. **Robust UI Components**
   Ext JS offers a set of over 140 ready-to-use UI components designed to work seamlessly together. These include complex grids, charts, and data visualization tools, which are often challenging to implement in other frameworks without external libraries.
3. **Enterprise Application Focus**
   Ext JS is designed for large, complex, enterprise-level applications. It includes features like integrated data handling, state management, and layouts that simplify managing complex applications. While other frameworks can also handle large applications, they often require more manual configuration and third-party tools.
4. **Consistency**
   Since Ext JS provides everything in one package, it ensures a more consistent

development experience. Other frameworks require more decisions about which libraries and tools to use, which can lead to inconsistencies.

5. **Mature framework**
   Ext JS has been in the market for a long time and is well-suited for building traditional, complex, data-driven applications. If you work in industries where such applications are essential (like finance, healthcare, etc.), Ext JS could be the best option due to its focus on performance and scalability.

Choose Ext JS if you need a comprehensive, enterprise-ready solution with many integrated features, particularly for complex, data-driven applications.

Important

This manual will focus primarily on the **Classic Toolkit**. Although some class names may vary between toolkits, the underlying logic remains the same.

This document will guide you through the first steps with Sencha Ext JS, from installing the software to creating key components like grids, charts, and forms.

02

Project Creation

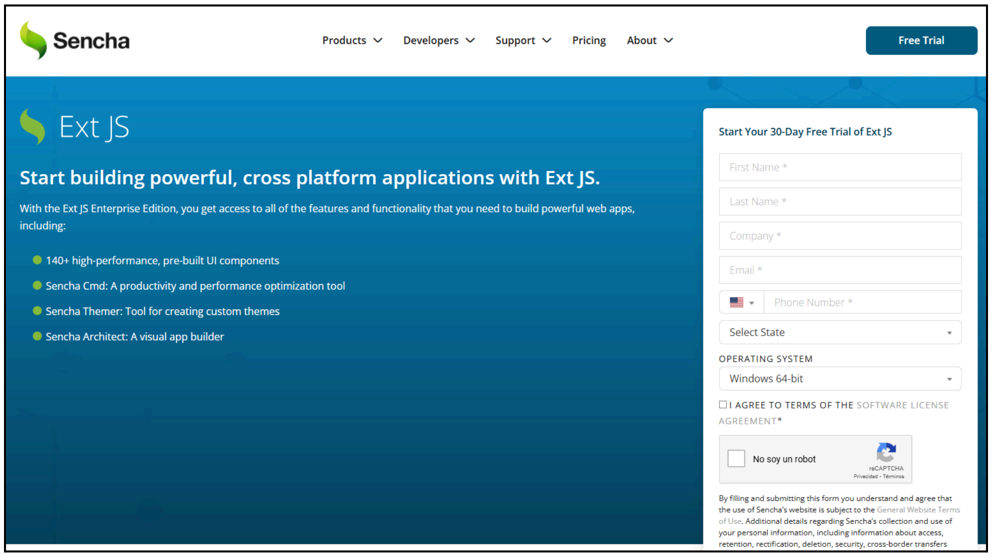
Before we start, we need to obtain a valid Sencha Ext JS license. We will go through the steps to download both, the trial and the commercial license.

Trial License Project

Downloading the Trial Version

This module explains the steps to install the trial version of Ext JS (using version 7.8.0), create your first project, and perform its initial configuration.

To download the trial version of Ext JS, visit www.sencha.com and select **Download Trial** at the top of the screen. Alternatively, you can access the download panel directly via the following link: https://www.sencha.com/products/extjs/evaluate/.



Start building powerful, cross platform applications with Ext JS.

With the Ext JS Enterprise Edition, you get access to all of the features and functionality that you need to build powerful web apps, including:

- 140+ high-performance, pre-built UI components
- Sencha Cmd: A productivity and performance optimization tool
- Sencha Themer: Tool for creating custom themes
- Sencha Architect: A visual app builder

Start Your 30-Day Free Trial of Ext JS

OPERATING SYSTEM: Windows 64-bit

☐ I AGREE TO TERMS OF THE SOFTWARE LICENSE AGREEMENT*

By filling and submitting this form you understand and agree that the use of Sencha's website is subject to the General Website Terms of Use. Additional details regarding Sencha's collection and use of your personal information, including information about access, retention, rectification, deletion, security, cross-border transfers

Once on the page, enter your information to start downloading Sencha Command. In the downloads section, you will find access to various Sencha tools, such as: Sencha Command, Rapid Ext JS, Sencha Architect, and Sencha Themer.

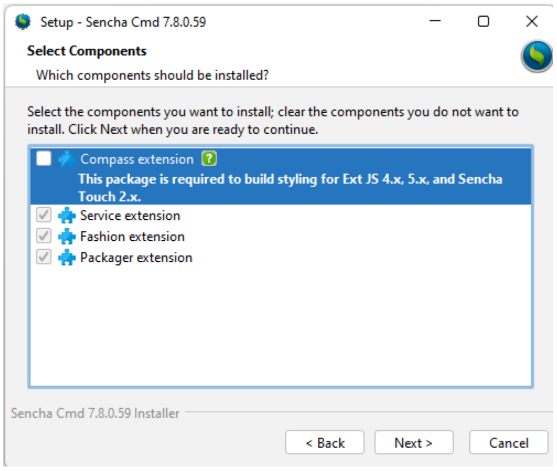
Installing Sencha Command

1. Download Sencha Command.
2. Extract the files from the package and run the installer.
3. Follow the instructions provided during the installation process.

Note

On Windows, you may receive a warning message from Microsoft Defender. Click Run Anyway to proceed.

4. In the installer components screen, you do not need to select the Compass extension, as it was used in earlier versions of Ext JS and is not required for version 7.8.0.


Setup - Sencha Cmd 7.8.0.59

Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [ ] Compass extension
  This package is required to build styling for Ext JS 4.x, 5.x, and Sencha Touch 2.x.
- [✓] Service extension
- [✓] Fashion extension
- [✓] Packager extension

Sencha Cmd 7.8.0.59 Installer

Using npm for Environment Management

If you prefer to use npm to manage your Ext JS environment or need more information about this method, refer to the official Sencha documentation:

- Getting Started with the ZIP File
  https://docs.sencha.com/extjs/latest/guides/getting_started/getting_started_with_zip.html
- Getting Started with NPM
  https://docs.sencha.com/extjs/latest/guides/getting_started/getting_started_with_npm.html

Trial Project Creation

Ext JS offers various templates for project creation. In the later chapter, **"Commercial License Project Creation"**, we will delve deeper into those templates. For now, we will learn how to create a basic project using the default template.

General Steps

1. Create a project folder.
2. Generate the project.
3. Build the project.

Project Folder

The first step is to select a folder where your project will reside. If you plan to use Sencha's test server, you can place the folder anywhere on your system. However, if you are working with a commercial web server, make sure to adhere to its specific rules and requirements.

Generate the Application

Once Sencha Command is installed, follow these steps to generate the application:

Step 1: Open a PowerShell Window

On Windows, you can do this in several ways:

- **From the Project Folder:**
  In Windows Explorer, select the desired folder, press **Shift** and right-click, then select **Open PowerShell Window Here**.
- From the Run Window Panel:
  Press **Windows + R**, type powershell, and press **Enter**.

Step 2: Navigate to the Project Folder

Ensure that you are located within the project folder. For example, if your project will be in `D:/SenchaProject/MyApp`, navigate to that folder before executing any commands.

Step 3: Generate the Application

Run the following command to generate the application:

```
sencha generate app [toolkit] -ext [ApplicationName] [Folder/Directory]
```

Where:

- **Toolkit**
  Specifies the type of application:
  - **classic**: For web applications targeting desktop devices.
  - **modern**: For web applications optimized for mobile devices.
- **ApplicationName**
  A simple name that complies with variable naming rules.
- **Folder/Directory**
  The relative path where the project will be generated.

Example:

```
sencha generate app classic -ext MyApp ./
```

This command will create a project named **MyApp** using the Classic Toolkit in the current folder.

Building the Project with Sencha Command

Once the project is generated, it is essential to build it to ensure proper functionality. Use **Sencha Command** with the following main commands:

- **sencha app build [environment]**
  Builds the application for one of the three available environments:
  - **development**: For ongoing development work.
  - **testing**: For testing purposes.
  - **production**: For deployment-ready build.
- **sencha app refresh**
  Reads the application's manifest and integrates necessary plugins and components.

- **sencha app watch**
  Activates Sencha's test server and enters watch mode to automatically regenerate the application when code changes are made.

Recommended Order of Commands

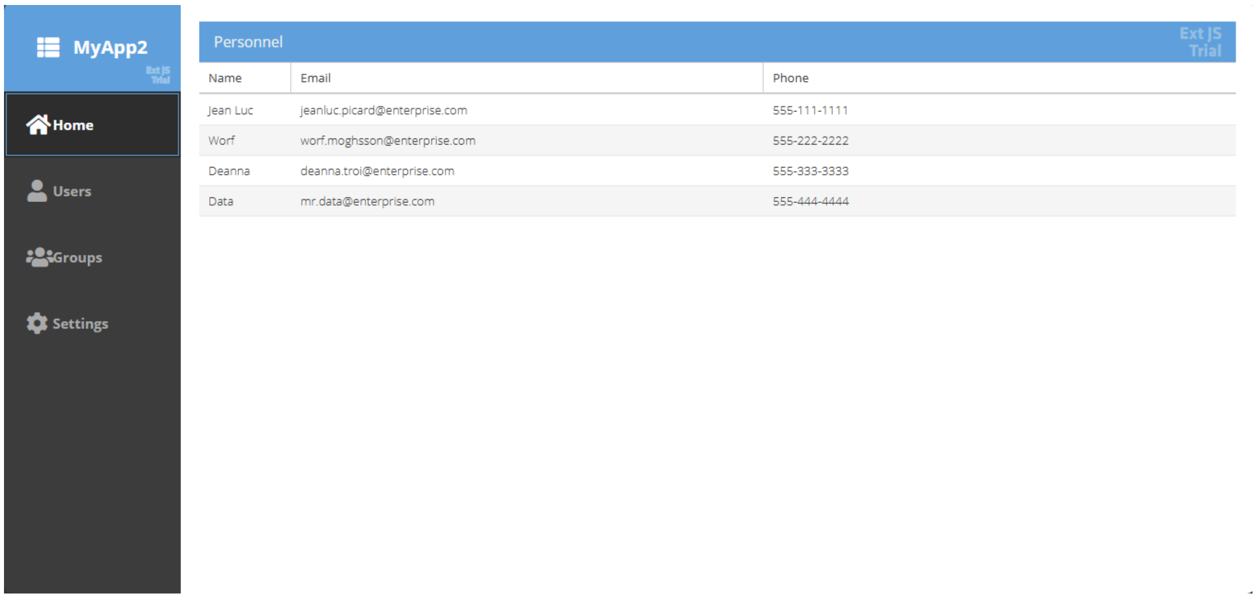
Run the following commands in sequence, from within the project folder:

```
sencha app build development
sencha app refresh
sencha app watch
```

Con estos pasos, habrás construido tu proyecto, integrado la configuración del manifest y activado el servidor de prueba.

Test Server

The test server will run at **`localhost:1841`**, displaying the base project.



Name	Email	Phone
Jean Luc	jeanluc.picard@enterprise.com	555-111-1111
Worf	worf.moghsson@enterprise.com	555-222-2222
Deanna	deanna.troi@enterprise.com	555-333-3333
Data	mr.data@enterprise.com	555-444-4444

Commercial License Project

This section outlines the essential steps for creating a project in Ext JS 7.8.0, including the necessary initial commands and configurations for its creation and customization.

Software Activation

If you have a commercial license and have not yet activated the software, you need to do so through Sencha's support panel.

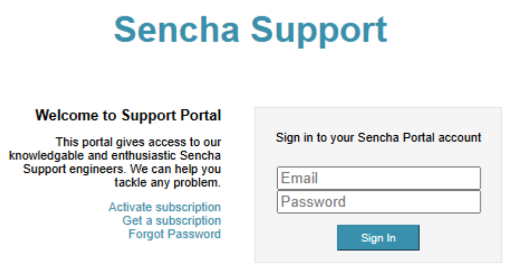
Upon purchasing a license, you will receive two activation codes (keys):

- **Support Panel Activation Key**
- **Additional Software Activation Key** (e.g., Sencha Architect and Themer)

To activate the software:

1. Visit the support panel at http://support.sencha.com.

2. Click on **Activate Subscription.**

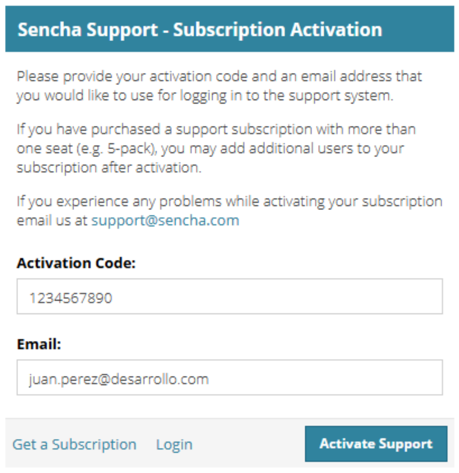


Sencha Support

Welcome to Support Portal

This portal gives access to our knowledgable and enthusiastic Sencha Support engineers. We can help you tackle any problem.

3. On the following screen, enter your activation code and the email address used when purchasing the license.



Sencha Support - Subscription Activation

Please provide your activation code and an email address that you would like to use for logging in to the support system.

If you have purchased a support subscription with more than one seat (e.g. 5-pack), you may add additional users to your subscription after activation.

If you experience any problems while activating your subscription email us at support@sencha.com

Activation Code: 1234567890

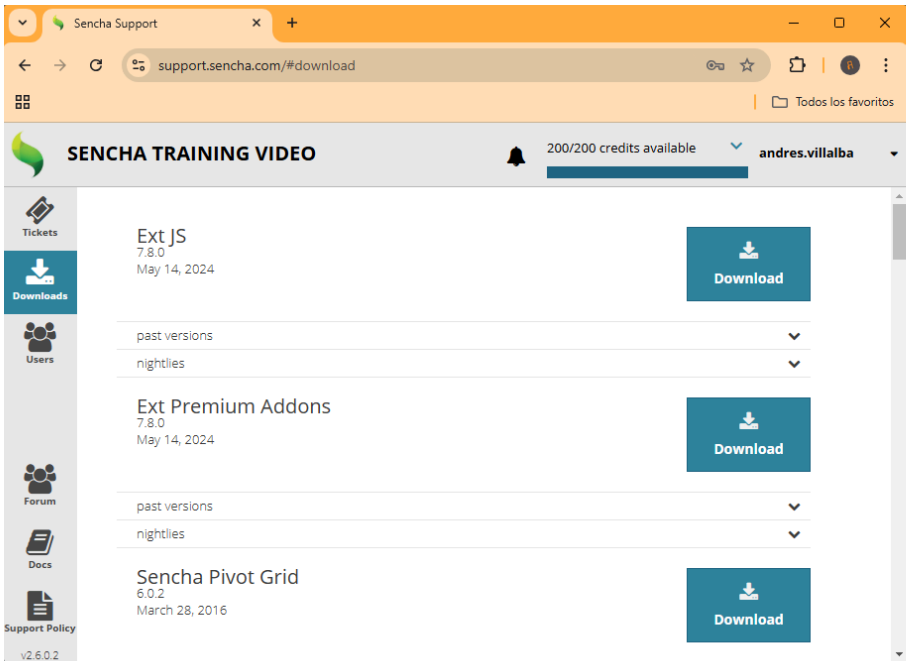
Email: juan.perez@desarrollo.com

Note

If you have more than one license linked to your account, additional licenses do not require direct activation. They will be managed from the same support panel.

Download and Installation

Once your license is activated, go to the **Downloads** section within the support panel (http://support.sencha.com) and download the following components:



Ext JS
7.8.0
May 14, 2024
Download
past versions
nightlies

Ext Premium Addons
7.8.0
May 14, 2024
Download
past versions
nightlies

Sencha Pivot Grid
6.0.2
March 28, 2016
Download

- **Sencha Cmd**
  A command-line tool essential for creating, building, testing, and deploying your projects.

- **Ext JS**
  While downloading the Ext JS SDK is not strictly necessary, having a local copy is highly recommended because:

  - It speeds up project creation by avoiding dependency on online downloads.

  - It allows you to work offline without an internet connection.

Save the Ext JS SDK in an easily accessible folder. For example:
`C:\sencha\ext-7.8.0`.

Installing Sencha Command

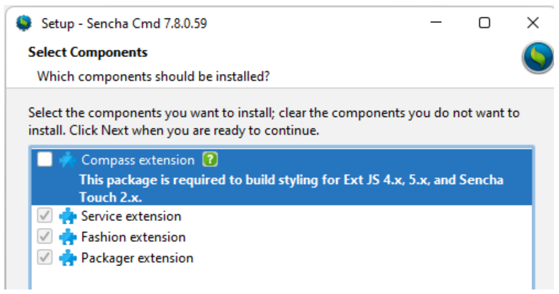
After downloading sencha Command, follow these steps to install it. The steps are identical to those for the trial version:

1. Extract the **downloaded file**.
2. **Run the installer** and follow the on-screen instructions.

Note

*On Windows, you might see a warning message from Microsoft Defender. Select **Run Anyway** to proceed with the installation.*

Important:
During the component selection screen in the installation process, it is not necessary to include the Compass extension. This tool was used in earlier versions of Ext JS and is not required for version 7.8.0.



Setup - Sencha Cmd 7.8.0.59

Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [ ] Compass extension
  This package is required to build styling for Ext JS 4.x, 5.x, and Sencha Touch 2.x.
- [x] Service extension
- [x] Fashion extension
- [x] Packager extension

If you prefer to use **npm** for the installation or need more information about this method, refer to the official Sencha documentation:

- **Getting Started with the ZIP File**
  https://docs.sencha.com/extjs/latest/guides/getting_started/getting_started_with_zip.html
- **Getting Started with NPM**
  https://docs.sencha.com/extjs/latest/guides/getting_started/getting_started_with_npm.html

Project Creation and Available Templates

As with any development process, the first step is to create or select a folder where your project will reside.

- **Using the Sencha Test Server**
  If you plan to use the Sencha test server (activated with sencha app watch), you can place your project folder anywhere on your system.

- **Using Your Own Web Server**
  If you'll work with your own web server, make sure to follow its specific rules and requirements to ensure the project runs correctly.

During the installation of Sencha Command, a folder containing the Sencha SDK was created. It is important to locate and remember the path to this folder, as it will be referenced during project configuration and generation.

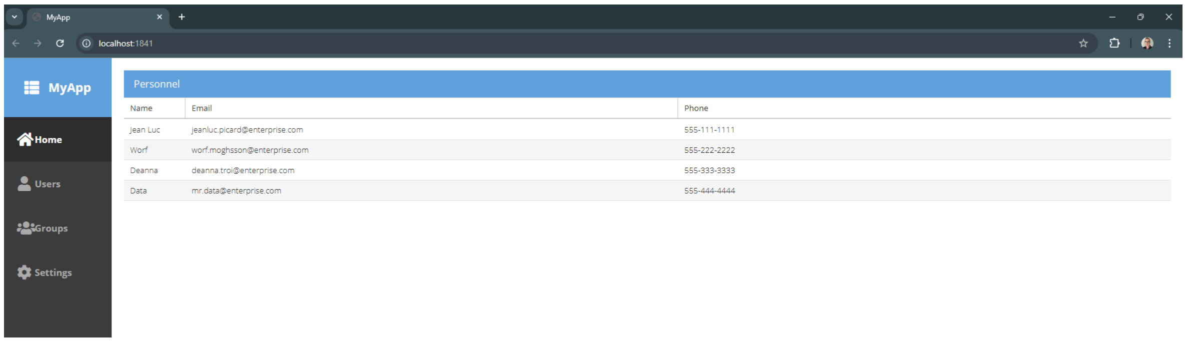
Sencha Ext JS provides several predefined templates to kickstart your project. These templates include:

- Basic Template
- Administrative Template
- Executive Template
- Blank Project

In the following sections, you'll learn how to generate each of these project types.

If this is your first project with Ext JS, it is recommended to start with the **Basic Template** to familiarize yourself with the framework's tools and workflows.

New Project Based on the Basic Template



Personnel

Name	Email	Phone
Jean Luc	jeanluc.picard@enterprise.com	555-111-1111
Worf	worf.moghsson@enterprise.com	555-222-2222
Deanna	deanna.troi@enterprise.com	555-333-3333
Data	mr.data@enterprise.com	555-444-4444

The **Basic Template** provides a starter project that includes a tab panel (tabpanel) with four default options. It is ideal for learning the fundamentals of Ext JS and starting structured development.

Use the following instructions to generate the project:

```
sencha -sdk [PATH_TO_SDK] generate app --[toolkit] [AppName] [ProjectPath]
```

Command Parameters

- **Path to SDK**
  Specify the location of the previously downloaded Ext JS SDK.
  Example: `d:/ext-7.8.0/`
- **Toolkit**
  Indicates the toolkit to use: Classic or Modern.
- **AppName**
  The name of the application, must follow these rules:
  - No spaces or special characters.
  - Must adhere to valid JavaScript naming conventions.
- **ProjectPath**
  Specifies the location where the project will be generated.

Example

To create a project named "MyApp" using the **Classic** toolkit, run:

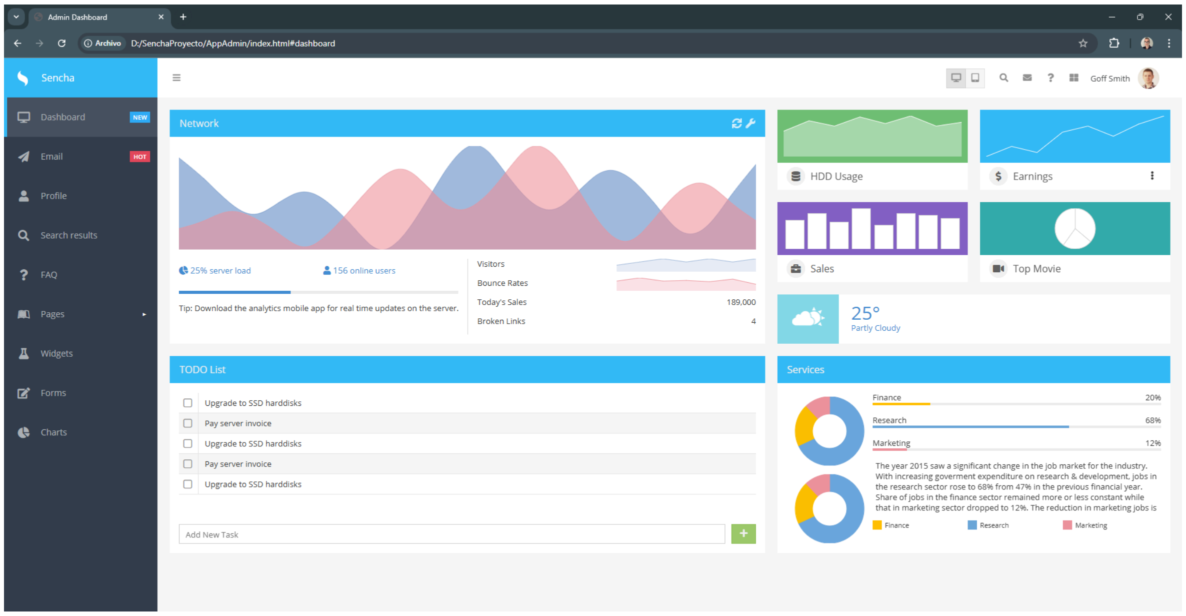
```
sencha -sdk d:/ext-7.8.0/ generate app --classic MyApp .\
```

This command will generate a new project called **MyApp** in the current folder, using the SDK located at d:/ext-7.8.0/.

After creating the project, it's essential to build it to ensure everything works correctly. Run the following command inside the project folder: **`sencha app build`**.

This command will generate the necessary files for the selected environment and prepare the project for execution or testing.

New Project Based on the Administrative Template



The **Administrative Template** is designed for complex projects and provides an initial structure with advanced panels and tree navigation. This template is ideal for robust enterprise applications.

To create an administrative project, you must run the command **from within the Ext JS SDK folder**.

Use the following command:

```
sencha generate app -s .\templates\admin-dashboard\ [AppName] [ProjectPath]
```

- **AppName**
  The name of the application, following these rules:
  - No spaces or special characters.
  - Must be a valid JavaScript name.
- **ProjectPath**
  Specifies the location where the project will be generated.

The toolkit is not specified for this template because the administrative panel is a **universal application**. It uses both the Classic and Modern toolkits based on the device requirements.

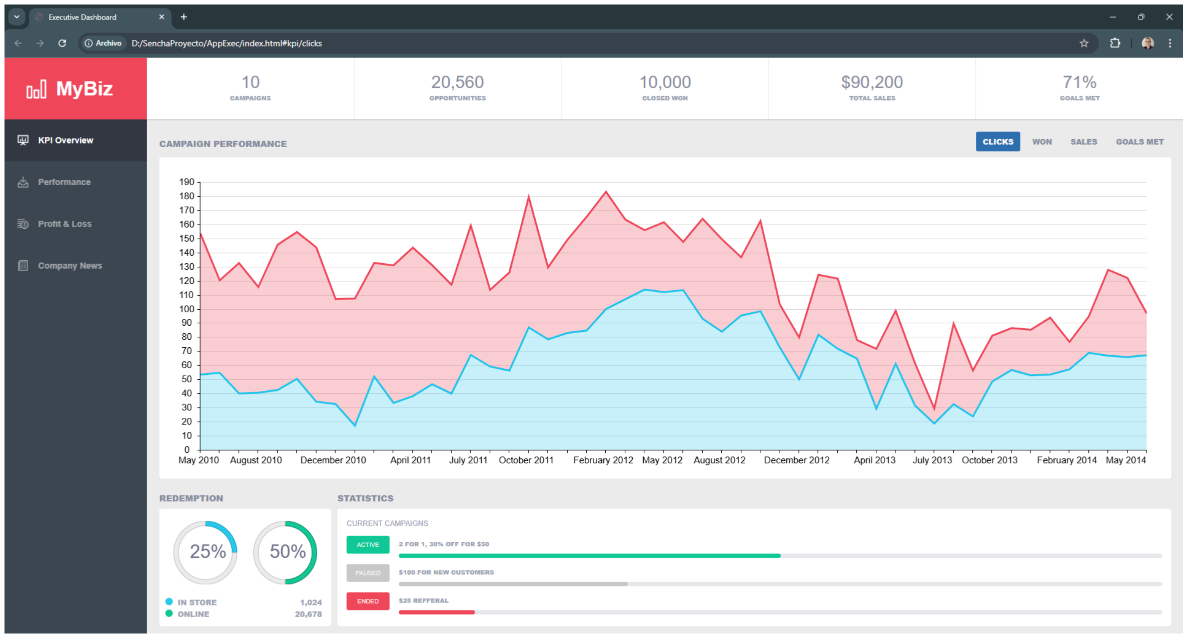
Example:

```
sencha generate app -s .\templates\admin-dashboard\ AppAdmin d:\AppAdmin
```

After creating the project, generate the necessary files by running: **`sencha app build`**.

Given the complexity of administrative projects, it is recommended to test the application on a local server, such as **Apache** or **IIS**, to ensure that resources and configurations load correctly.

New Project Based on the Executive Template



The **Executive Template** provides an initial structure with a tab panel (tabpanel). It is a more specific variation of the basic template, designed for applications focused on data visualization and analysis, such as dashboards or executive reports.

The procedure is similar to that of the administrative template, with the only difference being the name of the template in the command. Execute the following command **from within the Ext JS SDK folder**:

```
sencha generate app -s .\templates\executive-dashboard\ [AppName] [ProjectPath]
```

- **AppName**
  Defines the name of the application. It must be a valid JavaScript name: No spaces or special characters.
- **ProjectPath**
  Specifies the location where the project will be generated.

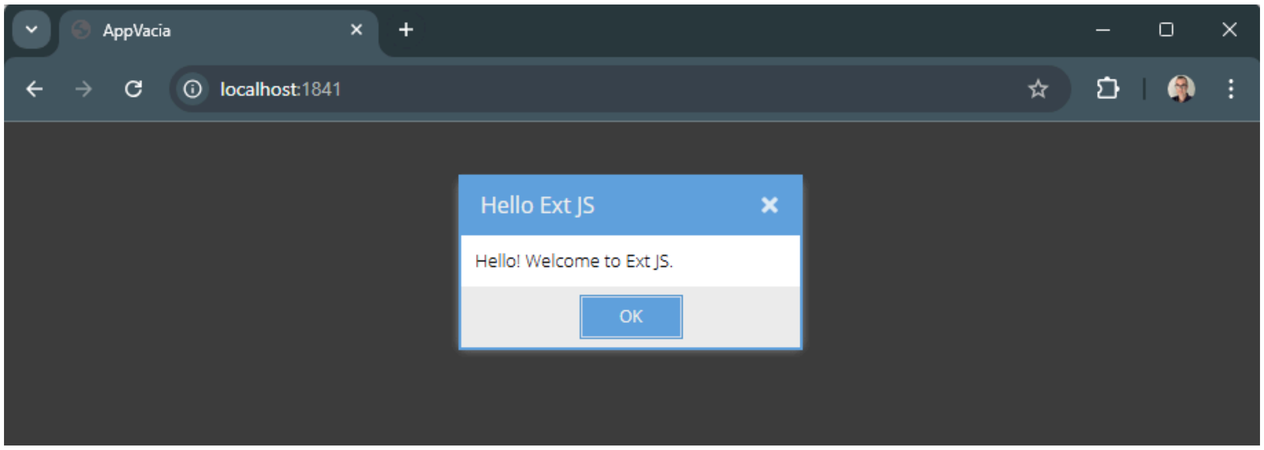
To create a project named "AppExe" in the d:\AppExe folder, run:

```
sencha generate app -s .\templates\executive-dashboard\ AppExe d:\AppExe
```

Once the project is generated, build the required files by executing the following command inside the project folder: **`sencha app build`**.

Due to the nature of executive projects, it is recommended to test the application on a local server, such as **Apache** or **IIS**, to ensure that all resources and configurations function correctly.

New Blank Project



Hello Ext JS

Hello! Welcome to Ext JS.

The **Blank Project** template is ideal for experienced users who want to start with a lightweight and fully customizable project. This option does not include navigation, giving you complete freedom to define the application's structure according to your needs.

Use the following command to generate a blank project:

```
sencha app init -e [PathToSDK] --[toolkit] [AppName]
```

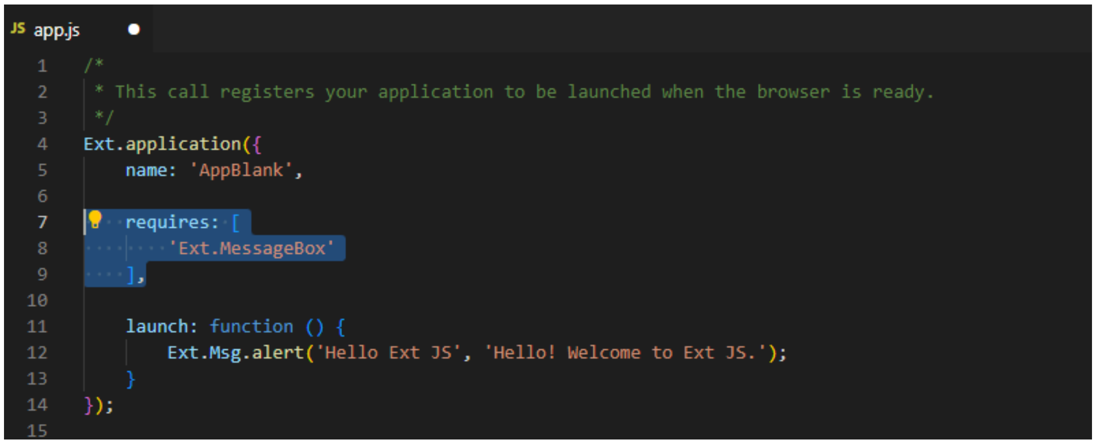
- **Path to SDK**
  Specifies the location of the previously downloaded SDK.
  Example: `d:/ext-7.8.0`
- **Toolkit**
  Choose the toolkit you want to use: Classic or Modern.
- **AppName**
  The name of the application. It must follow these rules:
  - No spaces or special characters.
  - Must be a valid JavaScript name.

To create a project named "AppBlank" using the Classic toolkit, run:

```
sencha app init -e d:\ext-7.8.0 -classic AppBlank
```

After generating the project, open the app.js file and remove the line referencing **`Ext.MessageBox`**. This step is necessary because this component is included by default in the Ext JS SDK.

Once you've made the adjustment, build the project by running the following command inside the project folder: **`sencha app build`**.



```
/*
 * This call registers your application to be launched when the browser is ready.
 */
Ext.application({
    name: 'AppBlank',

    requires: [
        'Ext.MessageBox'
    ],

    launch: function () {
        Ext.Msg.alert('Hello Ext JS', 'Hello! Welcome to Ext JS.');
    }
});
```

Building the Project

After generating your project, it's recommended to build it to ensure proper configuration. Sencha Command provides three main commands for this purpose:

- **sencha app build [environment]**
  Builds the application for one of three available environments: **development**, **testing**, or **production**.

- **sencha app refresh**
  Reads the app.json (manifest file) and integrates the necessary plugins and components for the project.

- **sencha app watch**
  Starts the Sencha test server and activates a watch mode, which automatically updates the application when changes are detected in the code.

For Simple Projects

For simpler projects, such as those created with the **Basic** or **Blank** templates, it is recommended to execute the following commands in this order, from the project folder:

```
sencha app build development
sencha app refresh
sencha app watch
```

What These Commands Achieve:

- Build the application in the development environment.
- Configure the required plugins and components.
- Start the Sencha test server at http://localhost:1841

For Administrative and Executive Projects

For more complex projects, such as those based on the **Administrative** or **Executive** templates, it is recommended to skip Sencha app watch and use only:

```
sencha app build development
sencha app refresh
```

Important Notes

1. **About Sencha app build:**
   This command is not required after every code change. Use it only in the following cases:
   - Preparing the application for production.
   - Resolving build-related issues.
   - Cleaning up outdated files from a previous build.
2. **About Sencha app refresh:**
   Execute this command only when changes are made to the app.json file.

By following these practices, you can optimize your development process and avoid unnecessary rebuilds.

03

The Basic Project

In this section, we will work on a project using the **Classic Toolkit** and its **Basic Template**. We'll learn how to modify navigation, create a grid, a form, and a bar chart. Except for the navigation menu, the other components can be applied to any template.

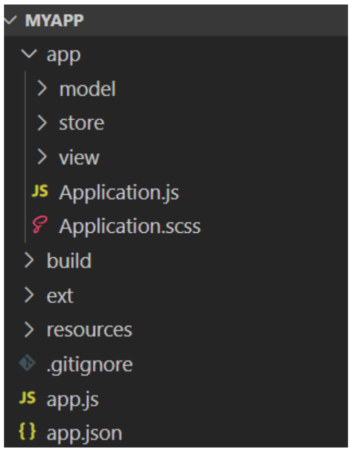
Selecting an IDE

The Integrated Development Environment or IDE is the development environment used for creating Ext JS web applications. Since Ext JS is based on JavaScript, you can select the IDE of your preference.

For this manual, we recommend **Microsoft Visual Studio Code (VS Code)** due to its compatibility with Ext JS plugins, which will facilitate development.

Project File Structure

The **Basic Template** in Ext JS generates a series of folders and files that form the foundation of the project and allow for its customization.



```
MYAPP
  app
    model
    store
    view
    Application.js
    Application.scss
  build
  ext
  resources
  .gitignore
  app.js
  app.json
```

Below are the main elements:

- **app**
  This is the main folder where custom Ext JS project files are created and stored. It may include: Views, Models, Stores, Controllers, and ViewModels.

  When referencing files in the app folder, Ext JS uses the application name to construct the class path according to naming conventions. This ensures clear and structured references.

For example, if your application is named **MyApp** and you want to load the file **app/view/main/List.js**, the corresponding class would be: `MyApp.view.main.List`

  - **MyApp**: The application name, defined in the app.js file.
  - **view**: folder where the file is located, in this case, part of the views folder.
  - **main**: Represents a specific subfolder or module within the view folder.
  - **list**: The name of the JavaScript file containing the class or component.

  In the app folder, there are the following subfolders:

  - **model**
    Stores data models used to describe the structure of backend information.
  - **store**
    Contains **data stores**, configured to connect to backend endpoints.
  - **view**
    Contains the application's **views** and **controllers**, such as navigation menus, grids, and forms.
  - **Application.js**
    The main application file. Although we won't modify it in this manual, it defines the application's base structure.

- **build**
  This folder stores files for each application environment (development, testing, production). For instance, a production build will be saved here.

- **ext**
  Contains the **Ext JS SDK**. This folder **should not be modified** directly. If you need to change a base class in Ext JS, it is recommended to use an override.

- **resources**
  Designed to store static assets such as images, videos, fonts, and other resources.

- **app.js**
  This is the primary class of the system alongside **app/Application.js**. It defines the application's name and allows you to configure global events and variables. We won't modify this file in this manual.

- **app.json**
  This configuration file lets you add components, change the application's language, and make other adjustments. Any modifications to this file will require running the following commands: **`sencha app refresh`** and **`sencha app build`**.

Editing the Navigation in an Ext JS Basic Template Project

In a project based on the **Basic Template** of Ext JS, navigation is implemented as a **tab panel**, defined in the app/Main.js file. Each tab corresponds to a section of the application, and its configuration is set within the items array.

Each element in the items array represents an individual tab in the navigation. Here's an example of the structure:

```
items: [
     { title: 'Section 1', items: [{ xtype: 'SectionView1' }],
     { title: 'Section 2', items: [{ xtype: 'SectionView2' }],
     { title: 'Section 3', items: [{ xtype: 'SectionView3' }],
     // ...
}]
```

Key Properties for Each Tab:

- **title**
  The name displayed on the tab.

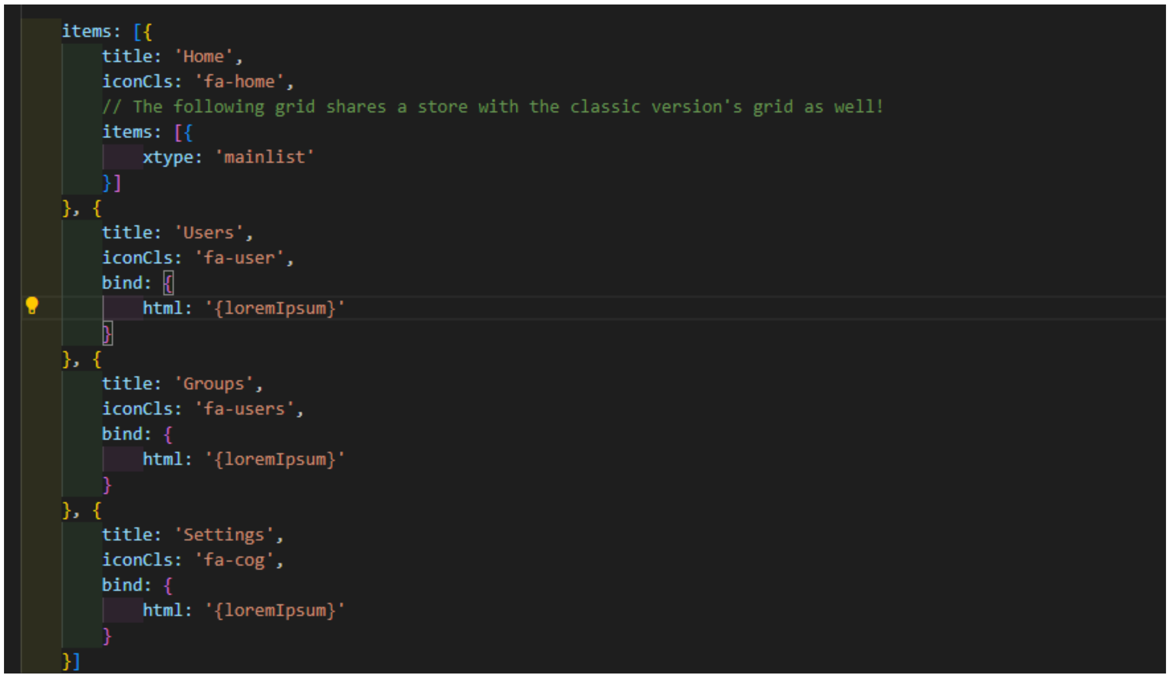
- **iconCls (opcional)**
  The icon for the section. Use a **Font Awesome** icon class in the format fa fa-[ICON].

For example: `iconCls: 'fa fa-home'`.
For a full list of icons, visit: https://fontawesome.com/v5/search?o=r&m=free

- **items**
  An object describing the content of the section. Typically includes:
  - An xtype, specifying the type of component (e.g., grid, form, chart).
  - A layout, often set to fit to ensure the content uses 100% of the available space.

When we create a new project, by default, the Basic Template includes four sections:

- The first section links to a simple grid view (mainlist).
- The other three sections contain placeholder content, such as lorem ipsum text.



```
items: [{
    title: 'Home',
    iconCls: 'fa-home',
    // The following grid shares a store with the classic version's grid as well!
    items: [{
        xtype: 'mainlist'
    }]
}, {
    title: 'Users',
    iconCls: 'fa-user',
    bind: {
        html: '{loremIpsum}'
    }
}, {
    title: 'Groups',
    iconCls: 'fa-users',
    bind: {
        html: '{loremIpsum}'
    }
}, {
    title: 'Settings',
    iconCls: 'fa-cog',
    bind: {
        html: '{loremIpsum}'
    }
}]
```

To Add a Section

1. **Update the items array**
   Add a new object to the array as shown below:

```
{
      Title: 'My Section',
      iconCls: 'fa fa-heart',
      layout: { type: 'fit' },
      items: [{
            xtype: 'my-section'
      }]
}
```

2. **Define the Class for the Section Content**
   Each section needs a corresponding class to define its content. For instance:
   - `Class name: MyApp.view.Examples.MyGrid`
   - `xtype: GridExample`
3. **Add the Class to `requires`**
   Add the class to the `requires` array in app/Main.js:

```
requires: [ 'MyApp.view.Examples.MySection' ]
```

To Remove a Section

Simply remove the corresponding object from the items array.

04

Ext JS View and Components

The MVC Model in Ext JS

The **MVC (Model-View-Controller)** architecture in Ext JS allows for modular code structuring, making applications easier to maintain and scale. Below are the main components and their roles:

- **View**
  Defines the visual structure of a component or page, including properties like titles, columns, events, and layouts.

- **Controller**
  Handles the application's logic, responding to events triggered in the view.

- **Data Store**
  Manages data by connecting to the backend. It loads, saves, and syncs data with the server, typically in combination with a data model.

- **Data Model**
  Describes the structure of the data being handled. It defines fields, validations, and relationships with other models. While optional, using data models is recommended for better control over retrieved data.

Ext.Define vs Ext.Create

Since Ext JS is object-oriented, you can define classes and create objects from them.

Class Naming Convention

Classes should follow this format: `ProjectName.folder(s).ClassName`

Example:
`MyApp.view.example.MyClass`

Ext.define

Used to define reusable classes for later instantiation. This is common for defining views, controllers, stores, or models.

Example:

```
Ext.define('MyApp.view.example.MyClass, { /* configuración */ })
```

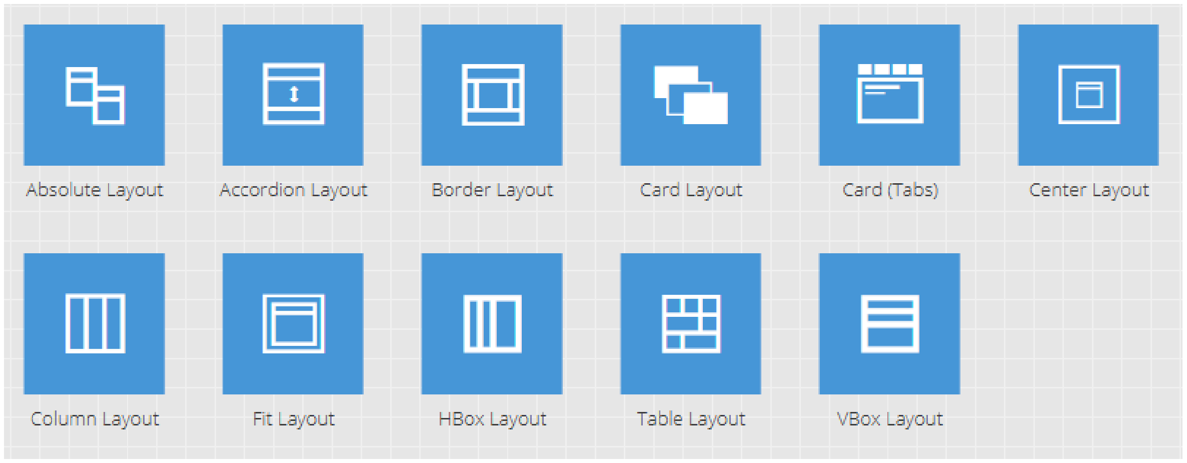
Ext.create

Used to instantiate objects from previously defined classes. This is often called within methods or events.

```
Ext.create('MyApp.view.example.MyClass', { /* configuración */ })
```

View 1: Panels and Layouts

Panels in Ext JS are containers that encapsulate components and manage their arrangement using layouts. Below are some common layouts:



- **Absolute Layout**
  Positions components directly using coordinates.

- **Accordion Layout**
  Organizes children in an accordion-style layout.
- **Border Layout**
  A central panel surrounded by panels in the north, south, east, and west positions.
- **Card Layout**
  Similar to a slideshow, displaying one panel at a time.
- **Card (Tabs)**
  Tabbed view, ideal for organizing information across screens.
- **Center Layout**
  Centers the component within the container panel.
- **Column Layout**
  Arranges components in columns, in a similar way to an hbox.
- **Fit Layout**
  Stretches the inner component to occupy 100% of the available space in the container.
- **HBox Layout**
  Lays out components horizontally in rows.
- **Table Layout**
  Arranges components in a table format.
- **VBox Layout**
  Stacks components vertically, useful for forms.

You can combine panels and layouts to create more complex views. For detailed examples, explore the **Sencha Kitchen Sink**:
https://examples.sencha.com/extjs/latest/examples/kitchensink/#layouts

Practical Example 1: Data Grid View

In this example, we'll create a new section in the navigation, add a data grid, and connect it to a store. We'll start with a project created using the **classic toolkit** and **basic template**.

Add a Menu Item

Add a New Menu Option

- Open the file **`MyApp.view.main.Main`** (main.js).
- Locate the items array and clear its existing sections, leaving it empty.
- Add a new tab for the grid:

```
items: [{
    title: 'Home',
    iconCls: 'fa fa-border-all',
    layout: 'fit',
    items: [{
        xtype: 'example-view'
    }]
}]
```

- **Title**. Tab title.
- **iconCls**. Font Awesome icon class for the tab.
- **Layout**. Specifies how the content fits within the tab.
- **Items**. Links to a sub-panel using the xtype example-view.

Update the requires Array

Add the class for the grid view (ExampleView) to the **`requires`** array in the same file:

```
requires: [
      'Ext.plugin.Viewport',
      'Ext.window.MessageBox',

      'MyApp.view.main.MainController',
      'MyApp.view.main.MainModel',
      'MyApp.view.main.List',
      'MyApp.view.example.ExampleView'
],
```

Optional Layout Adjustments

If desired, remove internal padding from the sections by changing the **`bodyPadding`** property:

```
bodyPadding: 0,
```

The Grid View

Create the Grid View

- Create a new file: **app/view/example/ExampleView.js**.
- Define the grid **`MyApp.view.example.ExampleView`** in the same file.
- To create columns, let's add the `columns` array, and describe the basic parameters for each column.

```
Ext.define('MyApp.view.example.ExampleView', {
    extend: 'Ext.grid.Panel',
    xtype: 'example-view',

    columns: [
        { text: 'Id',  dataIndex: 'id', width: 120 },
        { text: 'Name', dataIndex: 'name', flex: 1 },
        { text: 'E-mail', dataIndex: 'email', flex: 1 },
        { text: 'Active', dataIndex: 'active', width: 50 },
    ],

});
```

In Ext JS grids, columns offer a wide range of options to customize their appearance and behavior. To begin, you need to configure at least the following properties for each column:

- **text**
  The title of the column displayed in the grid header.
- **dataIndex**
  The name of the field from the store or data model that this column will

display. The value should match the field name defined in the store or model.

- **width**
  A fixed width for the column in pixels. If a unit is included, it must be written as a string (e.g., '200px'). Alternatively, use minWidth and maxWidth for a variable width.
- **flex**
  Defines a proportional width for the column relative to other columns. The larger the flex value, the more space the column will occupy.

Grid:https://docs.sencha.com/extjs/latest/classic/Ext.grid.Panel.html
Column Options: https://docs.sencha.com/extjs/latest/classic/Ext.grid.column.Column.html

- Add a title by using the title property and enable the use of a scrollbar with scrollable.

```
xtype: 'example-view',

title: 'Directory',
scrollable: true,

columns: [
```

- Finally, add pagination, which we will do using a bottom toolbar. Ext JS allows us to create toolbars in various ways. The main ones are **tbar** and **bbar**:

  - **tbar**: Top toolbar.
  - **bbar**: Bottom toolbar.

  Both toolbars can contain an array or a single element. However, for pagination, the toolbar **MUST NOT** be an array.

```
title: 'Directory',
bbar: { xtype: 'pagingtoolbar' }
columns: [
```

Pagination: https://examples.sencha.com/extjs/latest/examples/kitchensink/#paging-grid
Toolbars: https://examples.sencha.com/extjs/latest/examples/kitchensink/#toolbars

- Before proceeding with the store file, add a couple of lines to link the store to the view. Specify the store file to include and define the type of store:

```
    requires: [
        'MyApp.store.ExampleStore'
    ],
    title: 'Directory',

    store: {
        type: 'example-store'
    },

    bbar: { xtype: 'pagingtoolbar' }
```

The type and class name will be defined in the next section.

The ExampleStore Class

Although it is possible to include the store directly in the view, we will separate it into its own file. This way, the store can be reused in other parts of the application.

Data stores are usually located in the /app/store folder.

- Create the first store, naming it ExampleStore.js, and place it inside /app/store.

```
Ext.define('MyApp.store.ExampleStore', {
    extend: 'Ext.data.Store',
    alias: 'store.example-store',
});
```

It is important that the class name **`MyApp.store.ExampleStore`** matches the filename and its corresponding path, as well as the name added in the requires section of the ExampleView.js file.

Additionally, the alias must start with the prefix store. and should use the same name (without the prefix) in **ExampleView.js**.

Basic stores should extend from **`Ext.data.Store`**.

- Now, add the store proxy to connect it to the backend. To do this, specify the following properties within a proxy property:

```
Ext.define('MyApp.store.ExampleStore', {
    extend: 'Ext.data.Store',
    alias: 'store.example-store',
    proxy: {
        type: 'ajax',
        url: '/data/directory.json',
        reader: {
            type: 'json',
            rootProperty: 'data',
            totalProperty: 'count',
        }
    },
});
```

    - **type**
      The type of connection to make. For now, we will use ajax.
    - **url**
      The relative or absolute path to the endpoint that will provide the data.
    - **Reader**
      The reader configuration. Here, specify the file type, root property, and count property.

- The endpoint loaded in the previous example contains 50 records and follows this format:

```
{
    "count": 25,
    "success": true,
    "data": [
        {
            "id": 1,
            "name": "Miguel Pérez",
            "email": "miguel.perez@example.com",
            "country": "Guatemala",
            "phone": "+58-60439108",
            "active": 1
        },
        {
            "id": 2,
            "name": "Carlos López",
            "email": "carlos.lopez@example.com",
            "country": "Argentina",
            "phone": "+58-65684255",
            "active": 1
        },

        // ...
        {
            "id": 50,
            "name": "Carmen Pérez",
            "email": "carmen.perez@example.com",
            "country": "Panama",
            "phone": "+53-14075955",
            "active": 1
        }
    ]
}
```

- **Important**: Specify that the store will be auto loaded to avoid explicitly calling the load method.

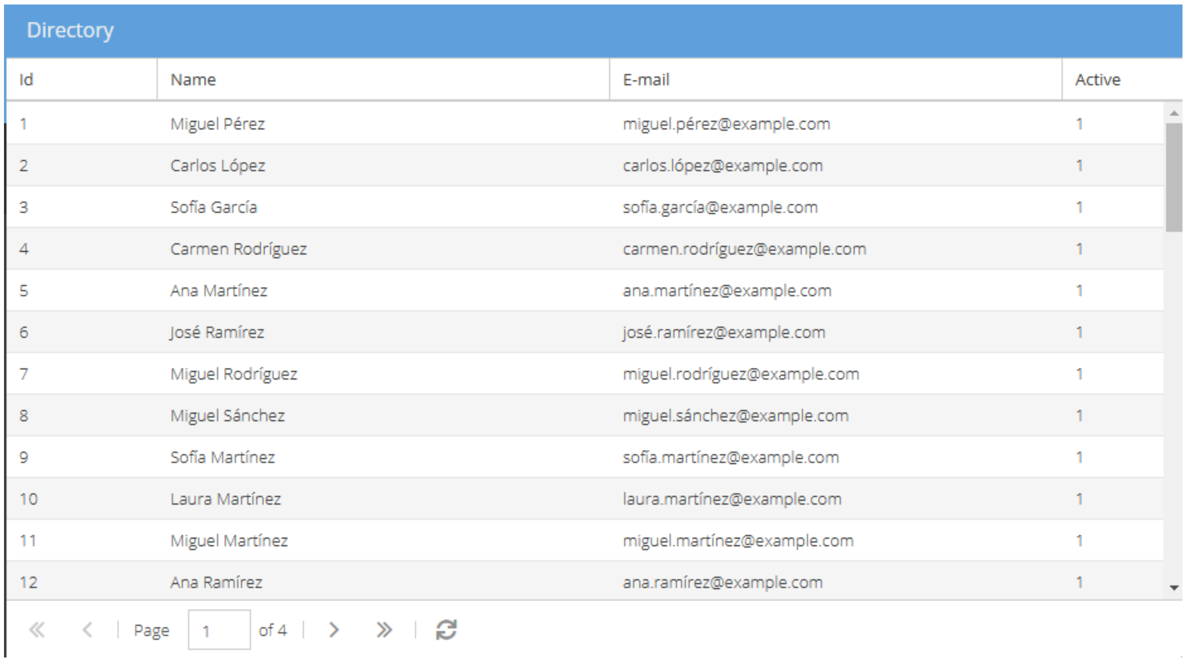
```
    alias: 'store.ejemplo-almacen',

    autoload: true,

    proxy: {
        type: 'ajax',
        url: '/data/directorio.json',
```

Test the project

- Once you have completed the example up to this point, you can review your grid. Before testing, make sure you have executed the following commands:
    - sencha app refresh
    - sencha app build development
- To test, you can use any of the following options:
    - Open the index.html file.
    - Run sencha app watch and open the URL http://localhost:1841
    - Set up a web server with Apache or IIS and open the project's path



Directory

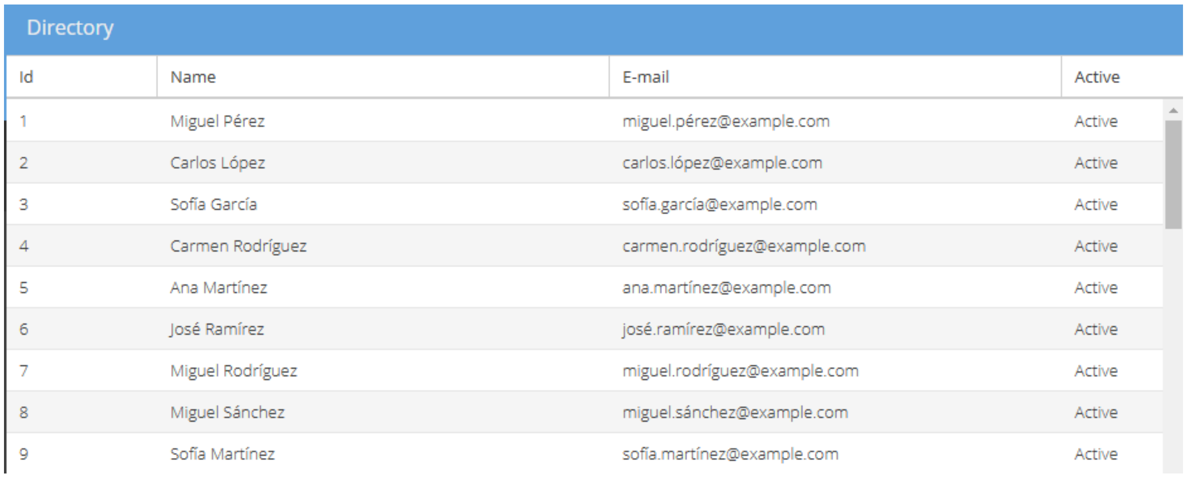
Id	Name	E-mail	Active
1	Miguel Pérez	miguel.pérez@example.com	1
2	Carlos López	carlos.lópez@example.com	1
3	Sofía García	sofía.garcía@example.com	1
4	Carmen Rodríguez	carmen.rodríguez@example.com	1
5	Ana Martínez	ana.martínez@example.com	1
6	José Ramírez	josé.ramírez@example.com	1
7	Miguel Rodríguez	miguel.rodríguez@example.com	1
8	Miguel Sánchez	miguel.sánchez@example.com	1
9	Sofía Martínez	sofía.martínez@example.com	1
10	Laura Martínez	laura.martínez@example.com	1
11	Miguel Martínez	miguel.martínez@example.com	1
12	Ana Ramírez	ana.ramírez@example.com	1

Other Options: Column Visualization and Behavior

- Let's modify the display for the "active" column so that it doesn't simply show a value of 1 or 0.

```
{
    text: 'Active',
    dataIndex: 'active',
    width: 50,
    renderer: function (value) {
        return (value == '1') ? 'Active' : 'Inactive';
    }
},
```

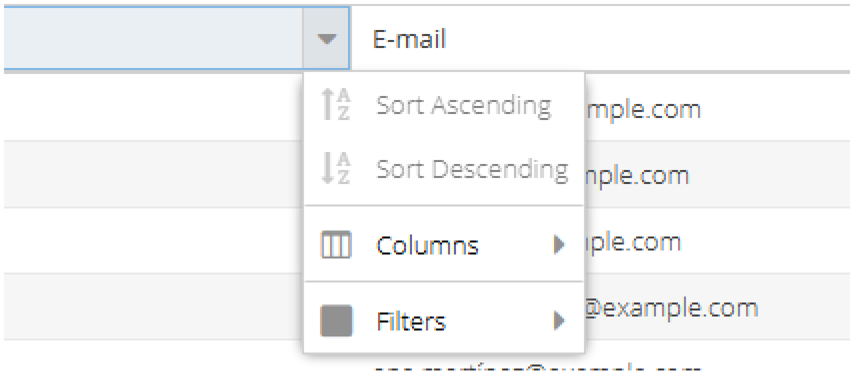
Resulting in the following screen:



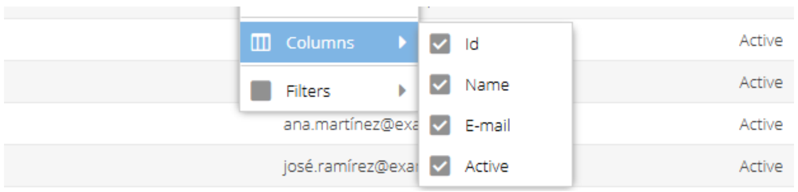
Directory

Id	Name	E-mail	Active
1	Miguel Pérez	miguel.pérez@example.com	Active
2	Carlos López	carlos.lópez@example.com	Active
3	Sofia García	sofia.garcía@example.com	Active
4	Carmen Rodríguez	carmen.rodríguez@example.com	Active
5	Ana Martínez	ana.martínez@example.com	Active
6	José Ramírez	josé.ramírez@example.com	Active
7	Miguel Rodríguez	miguel.rodríguez@example.com	Active
8	Miguel Sánchez	miguel.sánchez@example.com	Active
9	Sofia Martínez	sofia.martínez@example.com	Active

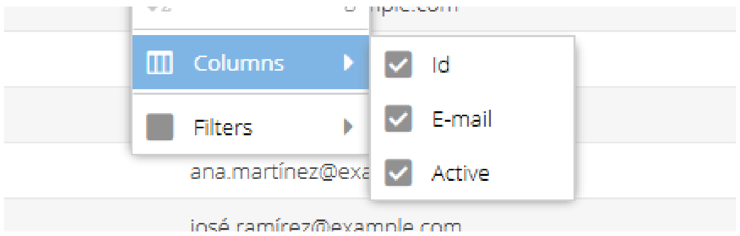
- **Sorting**: By default, the grid allows sorting for all its columns. To disable sorting for a column, add the **sortable** property with a value of false.



- **Show/Hide Columns**: By default, Ext JS provides the option to show or hide all columns.



If you want to disable this feature for a specific column, you can use the **hideable: false** property.



Filters

- Finally, to add filters, include the corresponding plugin in the grid's file.

```
Plugins: {
  gridfilters: true
},
```

- In the same file, specify the filter type for each column:

```
    { text: 'Id', dataIndex: 'id', width: 120, filter: 'number' },
    { text: 'Name', dataIndex: 'name', flex: 1, filter: 'string' },
    { text: 'Email', dataIndex: 'email', flex: 1, filter: 'string' },
    {
        text: 'Active',
        dataIndex: 'active',
        width: 80,
        filter: 'boolean',
        renderer: function (value) {
            return (value == '1') ? 'Active' : 'Inactive';
        }
    },
],
```

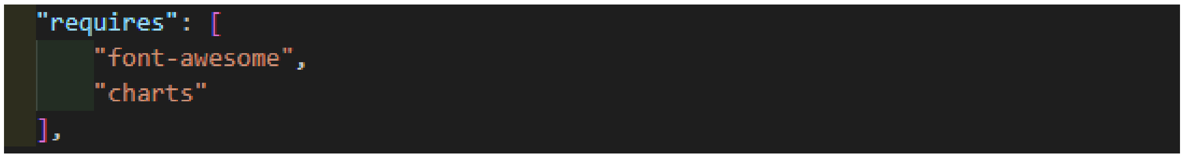
Filter Examples: https://examples.sencha.com/extjs/latest/examples/kitchensink/#grid-filtering
Filters: https://docs.sencha.com/extjs/latest/classic/Ext.grid.filter.Filters.html

View 2: Bar Chart

In this section, we will learn how to create a chart in Ext JS using data from a JSON file. Although it's not mandatory to have followed the previous steps, we assume you have a properly configured Sencha project ready for use.

Activating Chart Functionality

The first step is to open the **app.json** file, locate the requires section, and add "**charts**". This will load the charting package in Ext JS.



```
"requires": [
        "font-awesome",
        "charts"
],
```

Whenever the **app.json** file is modified, it's necessary to refresh the project with the command: **sencha app refresh;** in some cases, you may also need to rebuild the project using: **sencha app build [development | testing | production]**.

Adding the Option to the Menu

1. Open the file MyApp.view.main.Main (main.js). Make a few changes within this file.

2. Add a tab for our second section, which will contain the bar chart:

```
items: [{
    title: 'Chart',
    iconCls: 'fa fa-chart-bar',
    layout: 'fit',
    items: [{
        xtype: 'example-chart'
    }]
}]
```

Here, we define the title, icon, layout, and a reference to its sub-panel using the xtype **example-chart**.

3. In the same file, locate the requires section and add the class for the grid file (which we will call **ExampleChart**).

```
requires: [
      'Ext.plugin.Viewport',
      'Ext.window.MessageBox',

      'MyApp.view.main.MainController',
      'MyApp.view.main.MainModel',
      'MyApp.view.main.List',
      'MyApp.view.example.ExampleView',
      'MyApp.view.example.ExampleChart'
],
```

The Data

The file we will use is **earnings.json**. It contains 12 records with the following format:

```
{
    "count": 12,
    "success": true,
    "data": [
        {
            "month": "January",
            "sales1": 345,
            "sales2": 782,
            "sales3": 532,
            "total": 1659
        },
        // ...
    ]
}
```

The ChartStore Class

We will start by creating the data store. The steps are very similar to those performed for the grid.

4. Create a file in the store folder called **ChartStore.js**, which will connect to the backend to retrieve data from the JSON file.
5. Name the class **`MyApp.store.ChartStore`** and set its alias as **`store.chartstore`**. Similar to the grid store, extend it from **`Ext.data.Store`**.
6. We will not create a data model, so this time we will use the fields property to describe the fields to fetch.
7. Connect the store to the endpoint using the appropriate url.
8. Ensure the **`autoLoad`** property is included to automatically fetch data from the endpoint.
9. Define the reader with the properties **`rootProperty`** and **`totalProperty`**, corresponding to the backend's response fields.

```
Ext.define('MyApp.store.ChartStore', {
    extend: 'Ext.data.Store',
    alias: 'store.chartstore',

    autoLoad: true,
    proxy: {
        type: 'ajax',
        url: '/data/earnings.json',
        reader: {
            type: 'json',
            rootProperty: 'data',
            totalProperty: 'count'
        }
    },
});
```

The ExampleChart Class

10. n the folder view/example, create a chart folder and the file **ExampleChart.js**.

11. Extend from **`Ext.panel.Panel`**, as this will be a container for the chart.

12. Add a title for the panel using the **`title`** property. In this case, we will name it **"Monthly Sales"**:

```
Ext.define('MyApp.view.ejemplo.ExampleChart', {
    extend: 'Ext.panel.Panel',
    xtype: 'example-chart',
    title: 'Monthly Sales'
});
```

13. Add a **`cartesian`** chart as the panel's content, using the **`items`** array. Define the **`layout`** as **`"fit"`** to occupy 100% of the panel.

```
Ext.define('MyApp.view.example.ExampleChart', {
    extend: 'Ext.panel.Panel',
    xtype: 'example-chart',
    title: 'Sales',
    layout: 'fit',
    items: [{
        xtype: 'cartesian'
    }]
});
```

14. Add the previously created **store** using the **`store`** property and specify the type as **`chartstore`**.

15. Include the class name in the requires array

```
requires: [
    'MyApp.store.ChartStore'
],
items: [{
    xtype: 'cartesian',
    store: {
        type: 'chartstore'
    },
}],
```

To configure the chart, define the axes and series.

16. Define the Y-axis using type: 'numeric' and position: 'left' for the vertical axis.
17. Define the X-axis using type: 'category' and position: 'bottom' to position it at the bottom.

```
axes: [{
    type: 'numeric',
    position: 'left',
    title: 'Sales'
}, {
    type: 'category',
    position: 'bottom',
    title: 'Months'
}],
```

18. Use the **`series`** property to define the rest of the chart. Specify the type, the X field, and the Y field, matching the backend's data:

```
series: [{
    type: 'bar',
    xField: month,
    yField: 'total'
}]
```

There are several series types available for Cartesian charts, such as **`bar`**, **`line`**, and **`area`**.

Additionally, since the **`series`** property is an array, you can add multiple charts to a single container.

Charts: https://examples.sencha.com/extjs/latest/examples/kitchensink/#charts
Cartesian: https://docs.sencha.com/extjs/latest/classic/Ext.chart.CartesianChart.html
Series: https://docs.sencha.com/extjs/latest/classic/Ext.chart.series.Series.html

Other Visualization Options

Highlight and label

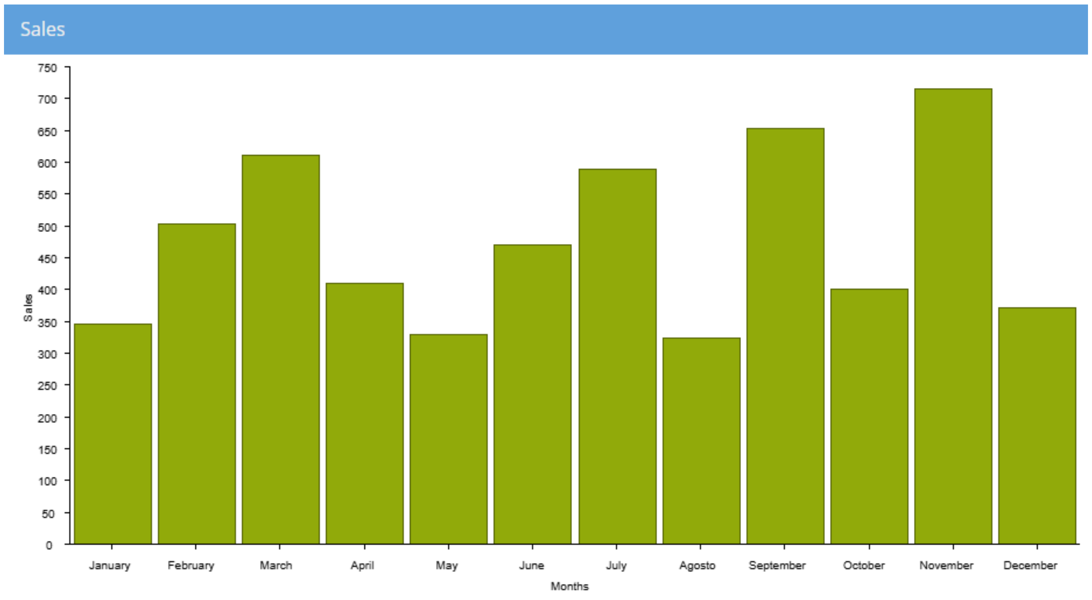
19. To add a highlight effect, use the **`highlight`** property in the series configuration. This will make the bar stand out when the cursor hovers over it.

20. To add a label, use the **`label`** property to display the value above each bar.

Here's how to implement both in the **series** configuration:

```
series: [{
  type: 'bar',
  xField: 'month',
  yField: 'total',
  highlight: true,
  label: {
    field: 'total',
    display: 'insideEnd'
  }
}]
```

21. The previous configuration will yield a bar chart:



- To customize the highlight color, instead of just setting the highlight property to true, you can specify more detailed properties such as fillStyle, strokeStyle, and lineWidth within the highlight configuration.

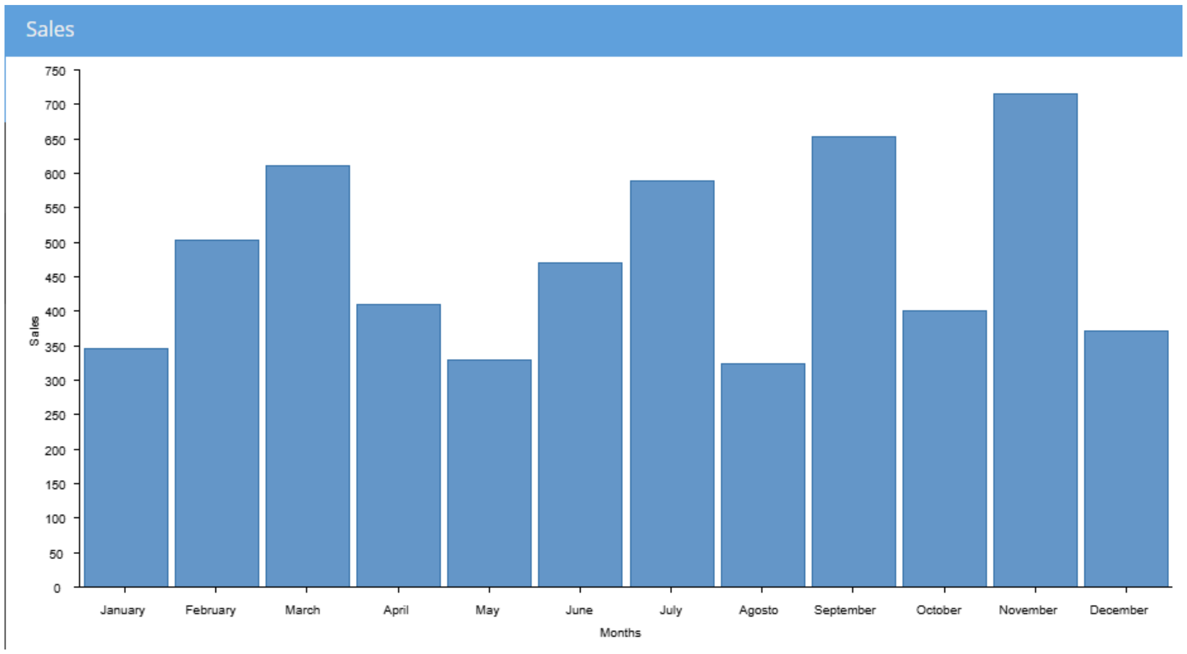
```
highlight: {
  fillStyle: '#369',
  strokeStyle: '#036'
},
```

Colors Property

- The colors property allows you to specify a color for each chart in the same series. It accepts an array of color codes that will be applied sequentially to the series.

```
colors: ['#69C'],
```

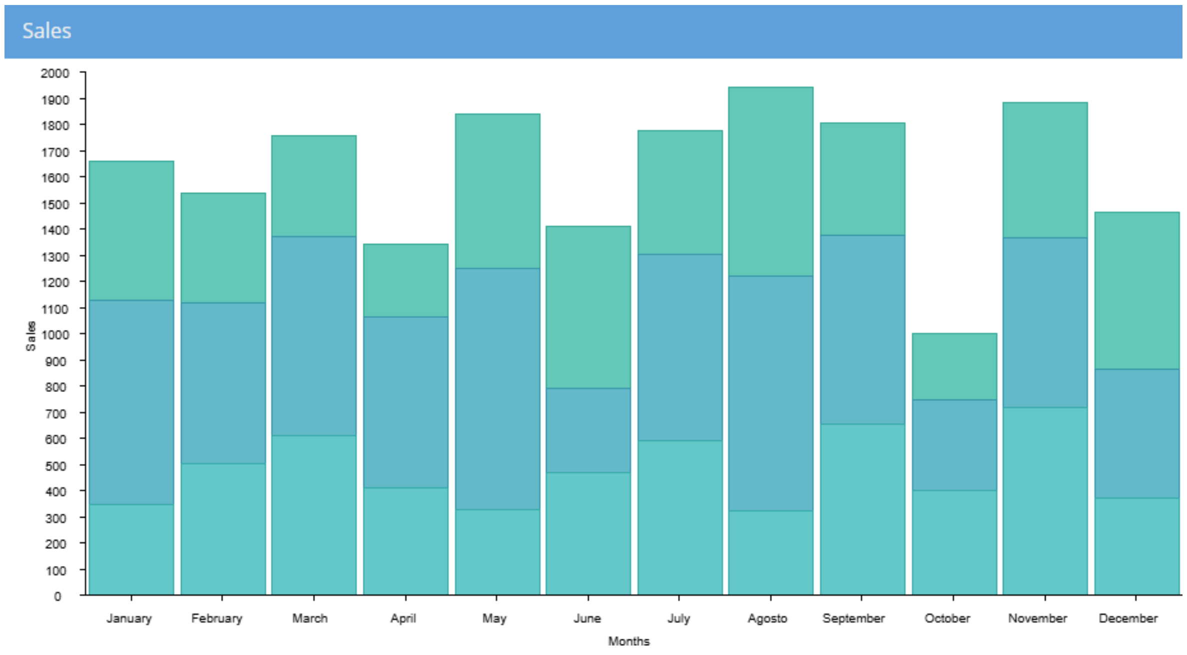
This example assigns a single color (#69C) to the bars in the chart. You can add more colors if you have multiple series.



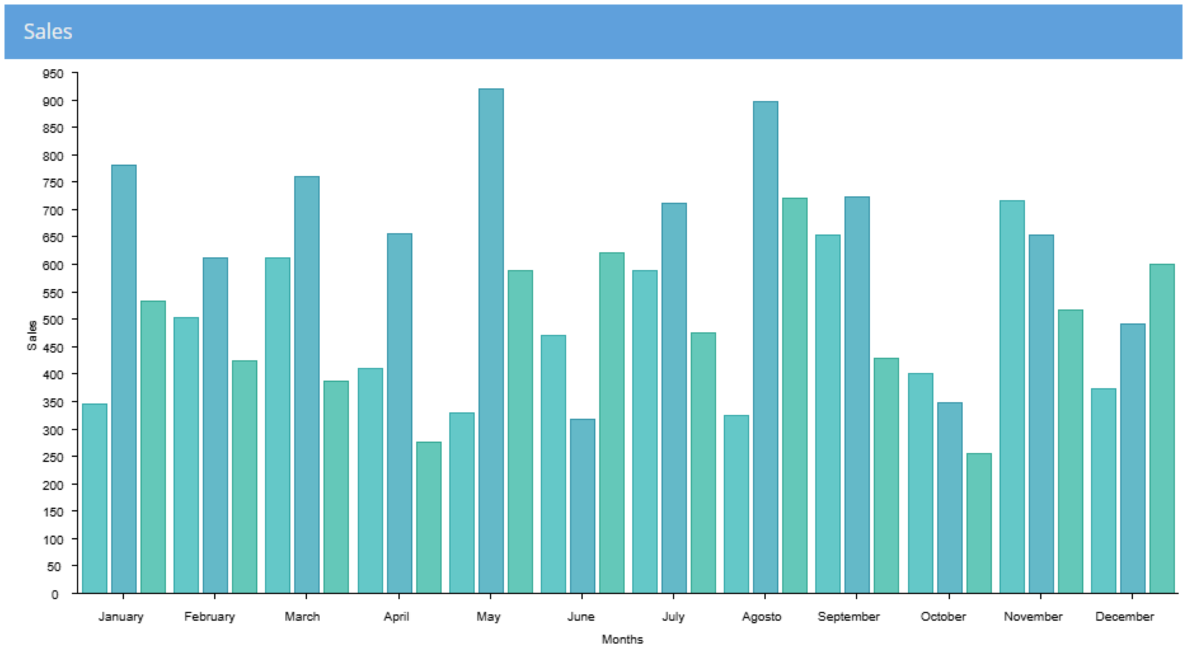
- You can include multiple fields within the **`yField`** property to display several data points in the same series.

```
yField: [ 'sales1', 'sales2', 'sales3' ],
colors: ['#6CC', '#6BC', '#6CB' ],
```

- **yField**. Specifies the fields to be used in the series. In this case, we're using sales1, sales2, and sales3 as separate fields for the Y-axis.
- **Colors**. Defines the colors for each field, where each one will be represented by a different color.

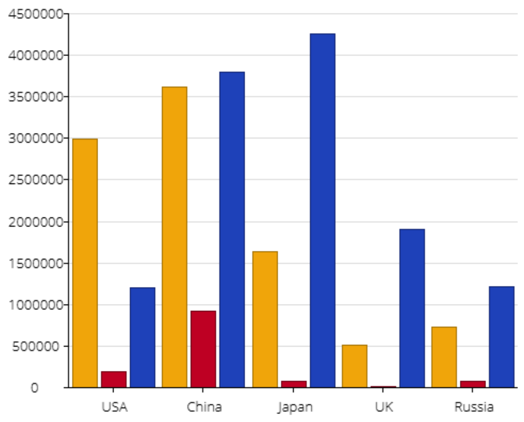


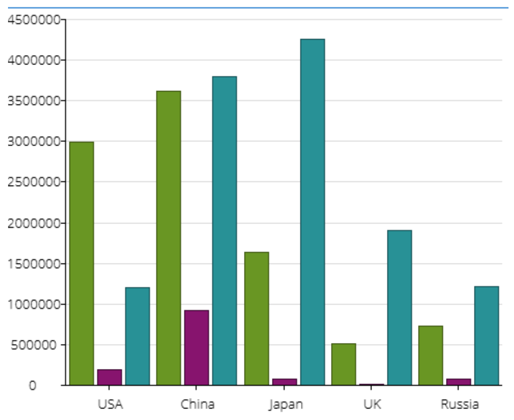
- By default, the bars will be stacked on top of each other. To disable stacking and show the bars side by side, you can set the stacked property to false.



Using Themes

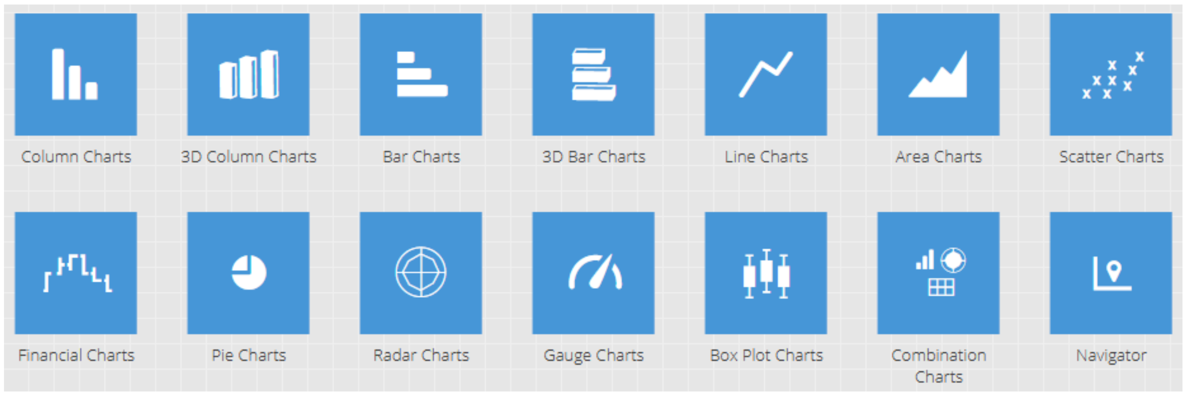
- To apply a theme to the chart, add the desired theme to the **`requires`** array. You can use themes like **`Ext.chart.theme.Category1`**, **`Category2`**, etc.





Additional Resources:

There are a wide variety of chart configurations and themes available in Ext JS. For more information, you can explore the Sencha's KitchenSink:
https://examples.sencha.com/extjs/latest/examples/kitchensink/#charts



View 3: Capture Form

In this section, we will explore how to create a capture form in Sencha Ext JS. We will learn step by step how to configure it and use key components. As in the previous sections, we assume that the project has already been created.

Adding an Option to the Menu Bar

Add the Option to the Menu As with the previous sections, let's add a new tab to the menu.

1. Open the file **`MyApp.view.main.Main`** (main.js).

2. Add the tab to our third section, which will contain a capture form.

```
items: [{
      title: 'Capture',
      iconCls: 'fa fa-pen-square',
      layout: 'fit',
      items: [{
            xtype: 'example-form'
      }]
}]
```

   We will specify the title, icon, layout, and a link to its sub-panel with the xtype example-form.

3. In the same file, search for the **`requires`** section, and add the class of our grid file (which will be called **ExampleForm**).

```
requires: [
      'Ext.plugin.Viewport',
      'Ext.window.MessageBox',

      'MyApp.view.main.MainController',
      'MyApp.view.main.MainModel',
      'MyApp.view.main.List',
      'MyApp.view.ejemplo.ExampleView',
      'MyApp.view.ejemplo.ExampleChart',
      'MyApp.view.ejemplo.ExampleForm'
],
```

The View and the Controller

We will create the view class and the controller class separately. This allows us to separate the logic (controller) from the view (form design).

The View Class

Although it is possible to create the form panel directly, we will create a container panel with a form panel inside it. This will allow us to center the form panel in the middle of the screen.

1. Navigate to the **app/view/example** folder, and inside it, create the **ExampleForm.js** file.
2. This file will contain the view definition for the **`MyApp.view.example.ExampleForm`**.

```
Ext.define('MyApp.view.ejemplo.ExampleForm', {
    extend: 'Ext.panel.Panel',
    xtype: 'example-form',

    items: [{
      xtype: 'form',
      title: 'Form Example',
    }]
});
```

3. To easily visualize the form, we will add the following properties:

```
Ext.define('MyApp.view.ejemplo.ExampleForm', {
    extend: 'Ext.panel.Panel',
    xtype: 'example-form',
    layout: 'center',
    bodyStyle: {
        backgroundColor: '#ccc'
    },

    items: [{
      xtype: 'form',
      title: 'Form Example',
      width: 500,
      height: 600,
      border: true,
      bodyPadding: 16,
    }]
});
```

- **layout:center**
  The main panel has its layout set to center, which means all its elements will be positioned in the middle of the container panel.

- **bodyStyle**
  We will modify the main panel's body style to have a light gray background, making it visually distinguishable from the form.

- **items**
  The only element inside the container panel is a form, which will contain all the form’s input fields. Additionally, we must define the following properties:

  - **xtype**
    Set to form, allowing easy validation and submission to the backend later.

  - **title**
    The title of the form/window.

  - **width**
    The width of the window.

  - **height**
    The height of the window.

- **border**
  Adds a border to the panel to visually separate it from the background.

- **bodyPadding**
  Adds inner padding within the form.

4. Before adding the form fields, we will configure the toolbar, which will be placed inside the form panel in our code. While we could use **`bbar`**, we will be using the **`buttons`** property to create a simple toolbar with predefined button styles.

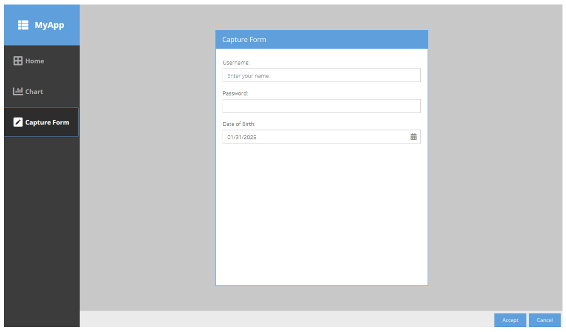
5. The buttons included will be **Save** and **Cancel**:

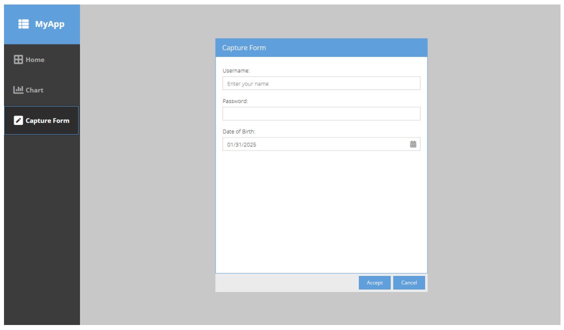
```
buttons: [{
    text: 'Accept'
}, {
    text: 'Cancel'
}],
```

   Where we place the buttons code, will determine where the buttons will be displayed:

   - **If placed inside the panel**, the buttons will be positioned at the bottom of the entire container.

   - **If placed inside the form**, the buttons will appear at the bottom of the form itself.





6. To add input fields, we will place them inside the form by using the items property. Ext JS provides a variety of components for creating form fields, including:

a. **textfield**. A standard text input field.

b. **numberfield**. A numeric input field that allows decimals.

c. **checkboxfield**. A checkbox selection field.

d. **radiofield**. A radio button selection field.

e. **datefield**. A date selection field.

f. **timefield**. A time selection field.

g. **combobox**. A dropdown list, configurable for searching.

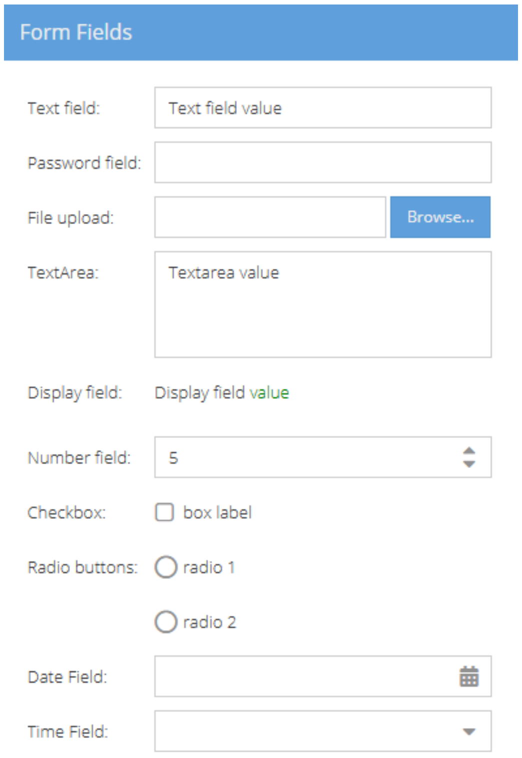
h. **tagfield**. Similar to a combobox but allows multiple item selection.

i. **hiddenfield**. A hidden input field.

j. **filefield**. A file upload field.

k. **textareafield**. A multi-line text input field.

l. **displayfield**. A read-only field for displaying information.


Form Fields

Text field:	Text field value
Password field:	
File upload:	Browse...
TextArea:	Textarea value
Display field:	Display field value
Number field:	5
Checkbox:	box label
Radio buttons:	radio 1
	radio 2
Date Field:	
Time Field:	

Examples: https://examples.sencha.com/extjs/latest/examples/kitchensink/#form-fieldtypes
Text Field: https://docs.sencha.com/extjs/latest/classic/Ext.form.field.Text.html
Base Field: https://docs.sencha.com/extjs/latest/classic/Ext.form.field.Base.html

Adding Fields to the Form

7. As previously mentioned, to include fields, we need to add the items property inside the form:

```
xtype: 'form',
title: 'Form',
width: 500,
height: 600,
border: true,
bodyPadding: 16,
buttons: [{
   text: 'Accept'
}, {
   text: 'Cancel'
}],
items: []
```

8. We will add three different types of fields: a **text field**, a **password field**, and a **date field**. This will help us explore different configurations.

9. **Text Field**: To add a text input field, we define the minimum required properties:

```
items: [{
   xtype: 'textfield',
   name: 'username',
   fieldLabel: 'Username',
   value: '',
   allowBlank: false,
   blankText: 'Error. Enter your username',
   emptyText:  'Your username'
}]
```

The following are the most common key properties for form fields:

a. **xtype**
   The type of field.

b. **name**
   The field's reference name, used when sending data to the backend.

c. **fieldLabel**
   The label displayed for the field.

d. **value**
   The default value.

e. **allowBlank**
   Determines whether the field is required (false means it must be filled)

f. **blankText**
   Error message shown if the field is empty.

g. **emptyText**
   Placeholder text displayed when the field is empty.

10. **Password Field**: The second field we will be adding is a password field. The password field is a **textfield** with the **inputType** property set to **password**.

```
{
   xtype: 'textfield',
   name: 'pwd',
   fieldLabel: 'Password',
   value: '',
   allowBlank: false,
   blankText: 'Enter a valid password',
   inputType: 'password'
}
```

a. **inputType**
   By default, it is set to **text**, but setting it to **password** hides the input characters.

11. **Date Field**: The third and last field we will be adding is a Date Field, which is configured as follows:

```
{
   xtype: 'datefield',
   name: 'dob',
   fieldLabel: 'Date of Birth',
   value: new Date(),
   maxValue: new Date(),
   allowBlank: false
}
```

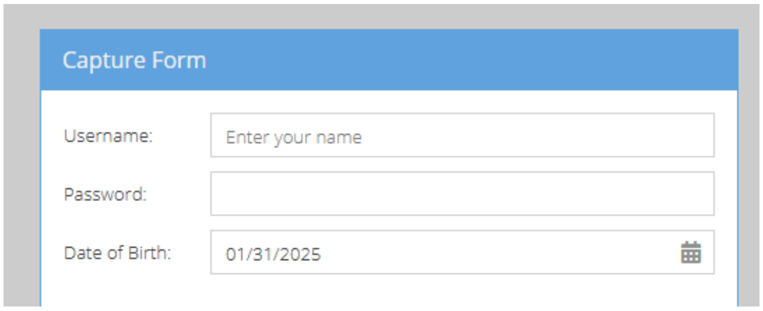
Additional properties for Date Fields:

a. **xtype**
   Defines it as a date field.

b. **value**
   Can be set using a JavaScript Date object.

c. **maxValue y minValue**
   Defines the valid date range.

12. Here's how the complete form looks with the three fields:



Username:	Enter your name
Password:	
Date of Birth:	01/31/2025

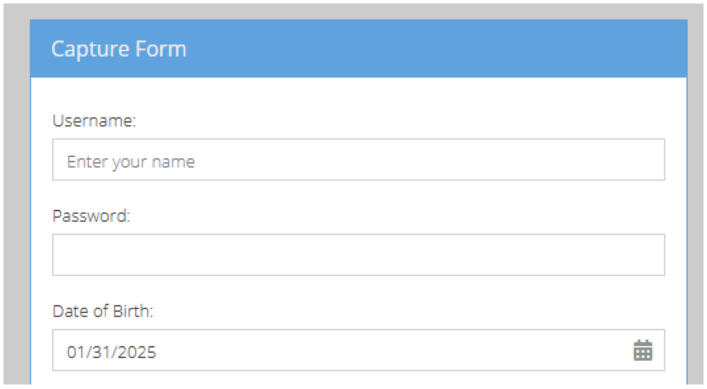
13. To refine the form layout, we add the defaults property to apply common configurations to all fields:

```
defaults: {
    labelAlign: 'top',
    width: '100%'
},
```

a. **labelAlign**
   Location of the field label.

b. **width**
   Adjust the field’s width, which includes the label and width.

These properties can also be set individually for each field.



14. Now that the form UI is complete, we will create the controller to handle form’s processes and events.

15. To link the view to the controller we must add the following lines:

a. Add the **`requires`** property to include the controller file.

b. Use the **`controller`** property to reference the controller.

```
Ext.define('MyApp.view.ejemplo.ExampleForm', {
    extend: 'Ext.panel.Panel',
    xtype: 'example-form',

    requires: [
        'MyApp.view.ejemplo.ExampleFormCtrl'
    ],

    controller: 'example-form-ctrl',

    layout: 'center',
```

16. To add a handler to the "Accept" Button, we assign an **event handler** to the button, referencing a function that will be defined in the controller.

```
{
   text: 'Accept',
   handler: 'SaveForm'
},
```

17. Now that the form is set up, we proceed to create the controller that will handle data submission and logic.

The Controller Class

18. Navigate to the **app/view/example** folder, and within it, create the file **ExampleFormCtrl.js**.
19. This file will be the controller, defining the class **`MyApp.view.example.ExampleFormCtrl`**. It is important that this name matches the one used in the requires section of the view.
20. The initial content will be:

```
Ext.define('MyApp.view.ejemplo.ExampleFormCtrl', {
    extend: 'Ext.app.ViewController',

    alias: 'controller.example-form-ctrl',

});
```

21. The controller class must extend from **`Ext.app.ViewController`** and include an alias, which must start with the **`controller`** prefix. This alias must match the name used in the controller property of the view.
22. Inside the controller, we will define all required methods. In this case, we will define the **`SaveForm`** method that was specified in the view.

```
Ext.define('MyApp.view.ejemplo.ExampleFormCtrl', {
    extend: 'Ext.app.ViewController',

    alias: 'controller.example-form-ctrl',

    SaveForm: function (btn,event) {
        let view=this.getView();
    }
});
```

We can observe that the method is defined as a property and takes a function with two parameters: **btn** (button) and **event**. However, in this case we will not use these parameters.

Inside the function, when we use the reserved word **`this`** we are referring to the controller itself. The **`getView()`** method retrieves the view associated with it, which in this case is the entire **`Ext.panel.Panel`**. Since we want to access the internal form, we need to reference the child elements of the main panel.

23. We will reference the **view** and search for the **form** with the following code:

```
GuardaForm: function (btn,event) {
    let view=this.getView();
    let form=view.down('form').getForm();
}
```

This way, we first get the **form** panel and then retrieve the **form** and its fields using the **`getForm()`** method.

24. Next, we validate the form. If there are no errors, we proceed with submission; otherwise, we display an error message.

```
let form = view.down('form').getForm();
if (form.isValid()) {
   // SUBMIT
} else {
   Ext.Msg.alert('There was an Error', 'Make sure you have filled out
all the required fields');
}
```

The **isValid()** method checks all fields and returns true if they are correctly filled. The **Ext.Msg.alert** function opens an Ext JS alert window displaying the error message.

25. To send the data to the backend, we can either make a **request** or use the form's **submit** method. Using submit simplifies the process since parameters are handled automatically.
    At this step, we will also add a mask to the view by using the mask() method while the call is being processed, and will remove it with unmask() upon completion.

```
// SUBMIT
view.mask('Saving');
form.submit({
   url: '/PATH/TO/ENDPOINT/',
   success: function () {
      view.unmask();
   },
   failure: function () {
      view.unmask();
   }
});
```

The submit method accepts four default properties:

- **url**: The endpoint URL that will process the submitted data.
- **method**: The REST method (POST, GET, DELETE, PUT) used to send the data.
- **success**: A callback function triggered if the submission is successful.
- **failure**: A callback function triggered if an error occurs while processing or connecting to the backend.

Within the callback methods, we can include additional actions, such as displaying an error message, closing the window, or switching to another view.

Final Notes

This concludes the **Getting Started with Sencha Ext JS** manual. This framework provides a vast array of features and components that were not covered in this guide.

For more information about **Sencha**, including licensing, features, and practical examples, we encourage you to explore the following useful links:

- **Visual Examples with Sencha's Kitchen Sink**
  https://examples.sencha.com/extjs/latest/examples/kitchensink/
- **Official Documentation**
  https://docs.sencha.com/
- **Sencha Official Website**
  https://www.sencha.com/
- **Sencha Trial**
  https://www.sencha.com/products/extjs/evaluate/
- **Templates and Examples**
  https://examples.sencha.com/
  https://examples.sencha.com/extjs/latest/

Additional Resources

Now that you're familiar with **Sencha Ext JS**, we invite you to explore more about this powerful tool through the following tutorials:

- Build an Email Client with Ext JS
  https://youtu.be/BxRvZ8alKpc?si=ipDOcMXVm-fQ54IB
- Create a Task Tracker (Time Clock) with ReExt
  https://www.youtube.com/watch?v=fNumYqaHGZU
- Sencha Official YouTube Channel
  https://www.youtube.com/@sencha
- Sencha Café (Spanish)
  https://youtu.be/lxnHgZsrJ00?si=8WuE5T1WaQphqh9w

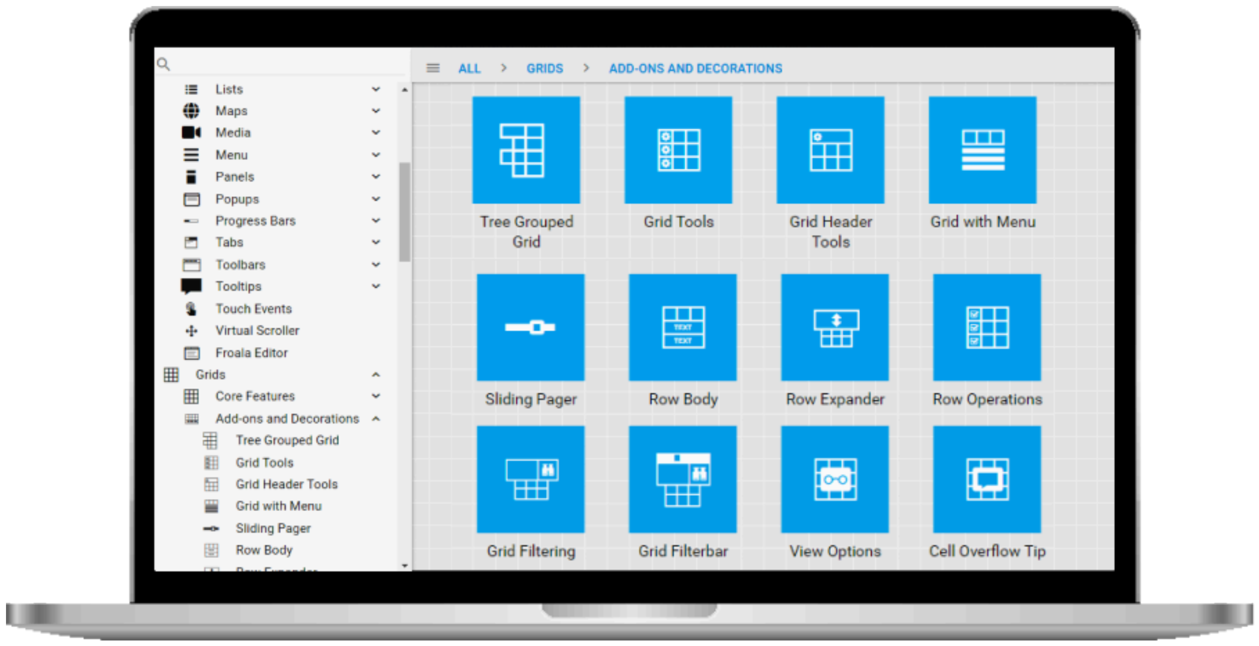
Copyright & Terms of Use

Acknowledgements

We sincerely thank all our readers for their time and dedication in following this manual. We also extend our gratitude to **Sencha** for providing such a powerful and flexible platform for application development.

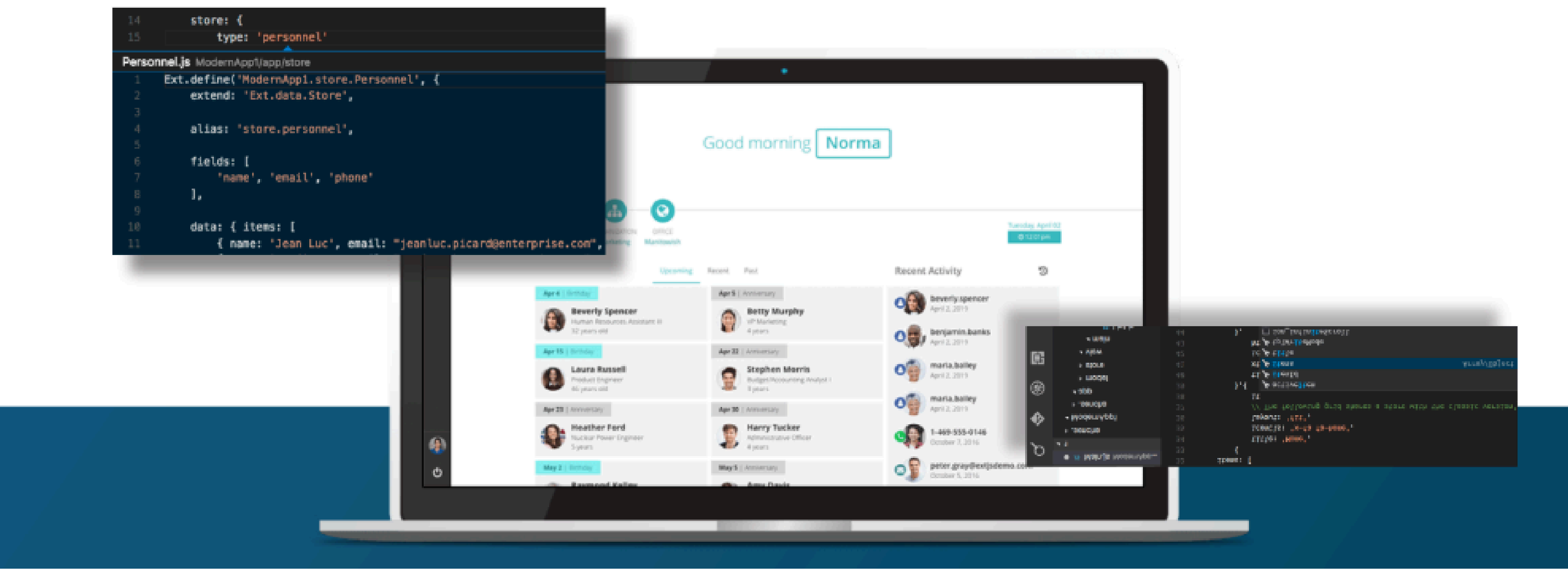
A special thanks to **Aumenta Software Development** for their continuous support in this project and for allowing us to share this knowledge with the community.

Finally, we appreciate the efforts and collaboration of the entire team involved in creating this manual—your hard work made this content possible. **Thank you all**!



Try Sencha Ext JS

FREE for 30 DAYS



```
    store: {
        type: 'personnel'
```

Personnel.js ModernApp1/app/store

```
Ext.define('ModernApp1.store.Personnel', {
    extend: 'Ext.data.Store',

    alias: 'store.personnel',

    fields: [
        'name', 'email', 'phone'
    ],

    data: { items: [
        { name: 'Jean Luc', email: "jeanluc.picard@enterprise.com",
```

Make the right decision for your business.

START YOUR FREE 30-DAY TRIAL

USEFUL RESOURCE LINKS

Get Started with Ext JS

View Examples and Documentation

Video Tutorials

Skill Sprints Series.

Email App Development - Tutorial.

Sencha