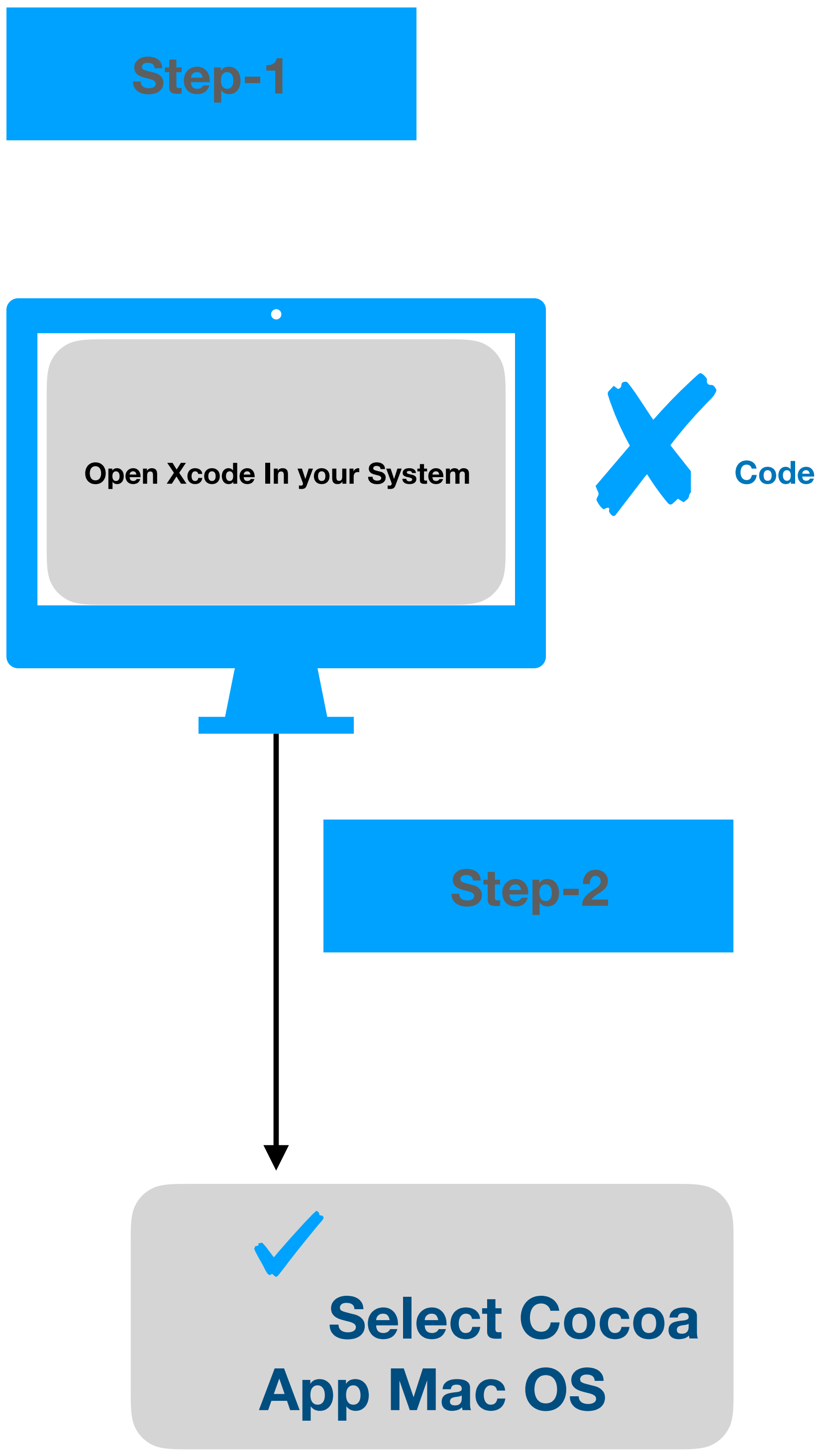
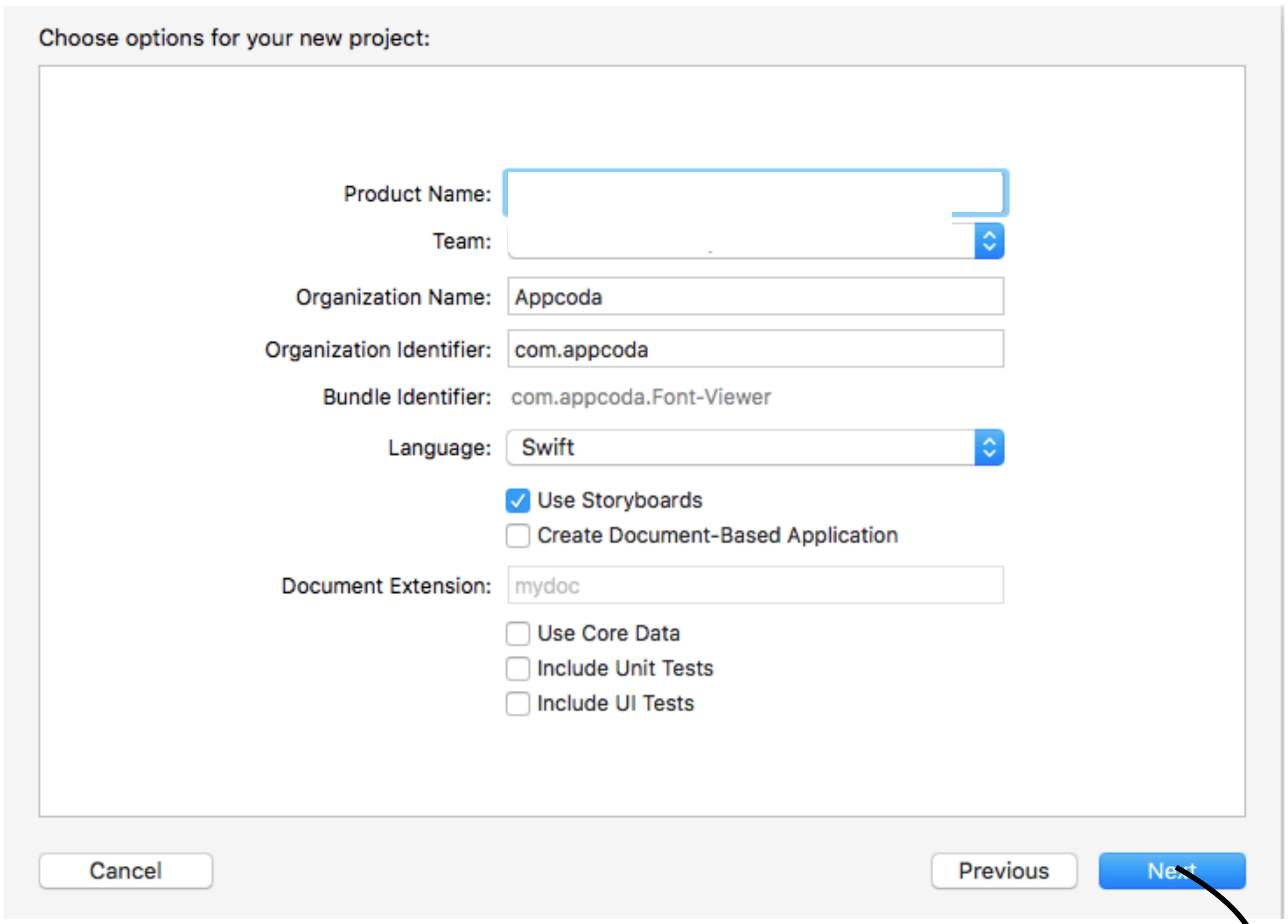
Data Flow Diagram



STEP-3


Choose options for your new project:

Product Name:

Team:

Organization Name: Appcoda

Organization Identifier: com.appcoda

Bundle Identifier: com.appcoda.Font-Viewer

Language: Swift

[✓] Use Storyboards

[ ] Create Document-Based Application

Document Extension: mydoc

[ ] Use Core Data

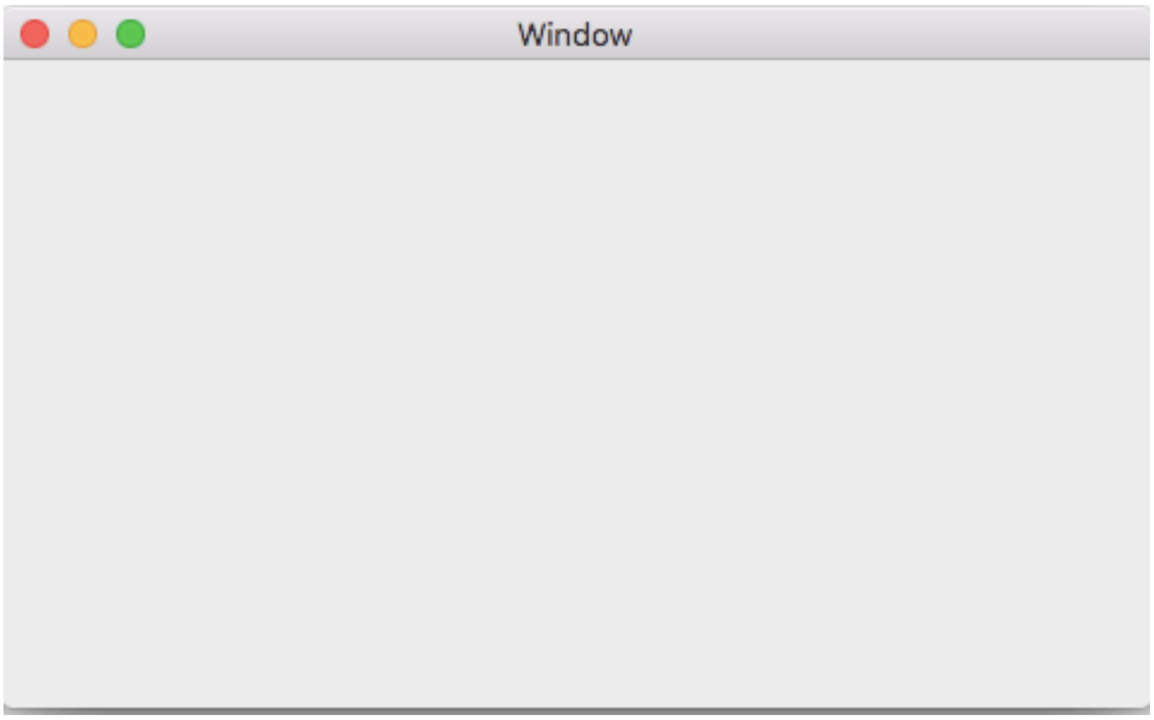
[ ] Include Unit Tests

[ ] Include UI Tests

Click On Next

Running The App :

While having Font Viewer project open in Xcode, press Cmd + R on your keyboard to run the app. Here’s what you’ll see appearing on your screen:



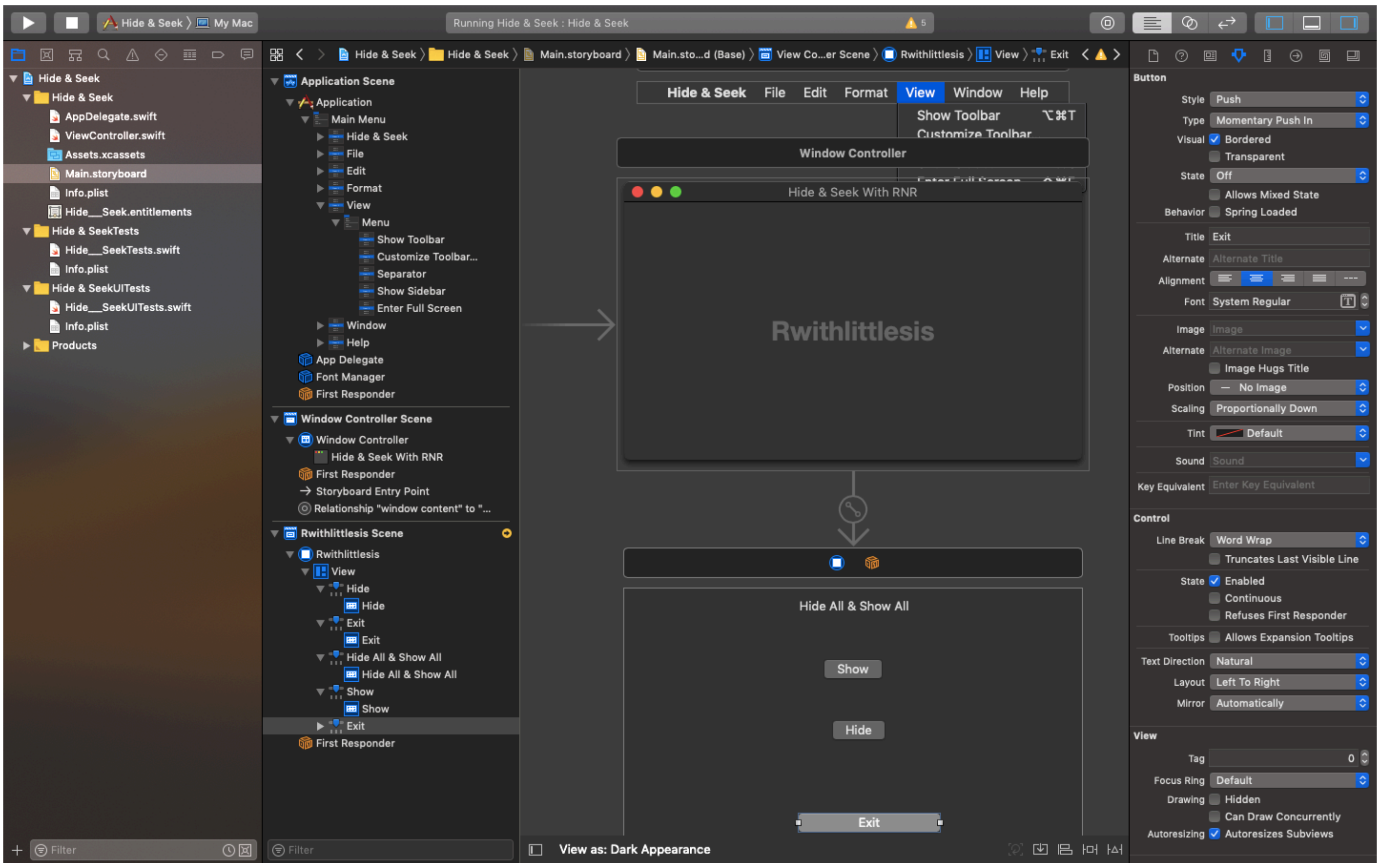
It’s an empty window for now, but it doesn’t matter as we will change that. Look and play around with it. You will notice that standard behaviour and attributes are assigned by default to the window, such as resizing, moving or going full screen. Also, there’s the main menu at the top bar, where some common functionalities are provided already, while some others are just disabled as they don’t trigger any action.

STEP-4

Exploring The Main.storyboard :

Clicking on the Main storyboard file in the Project Navigator, the automatically created, default user interface will be revealed in Interface Builder. Any Cocoa based app contains three default scenes:

- A window controller that manages the contained window (such as loading, showing and closing it, storing window's state, customize its title, and more).
- A view controller which is linked to the window controller and is being presented automatically along with the window. The View Controller scene is the place where Cocoa graphical controls are being added to.



Button

Style: Push
Type: Momentary Push In
Visual: ✓ Bordered
☐ Transparent
State: Off
☐ Allows Mixed State
Behavior: ☐ Spring Loaded
Title: Exit
Alternate: Alternate Title
Alignment
Font: System Regular
Image: Image
Alternate: Alternate Image
☐ Image Hugs Title
Position: — No Image
Scaling: Proportionally Down
Tint: Default
Sound: Sound
Key Equivalent: Enter Key Equivalent

Control

Line Break: Word Wrap
☐ Truncates Last Visible Line
State: ✓ Enabled
☐ Continuous
☐ Refuses First Responder
Tooltips: ☐ Allows Expansion Tooltips
Text Direction: Natural
Layout: Left To Right
Mirror: Automatically

View

Tag: 0
Focus Ring: Default
Drawing: ☐ Hidden
☐ Can Draw Concurrently
Autoresizing: ✓ Autoresizes Subviews

- The Application scene, which is the graphical representation of the `NSApplication` instance that manages the app's main event loop and resources used by all of that app's objects (**by Apple docs**). The main menu of the app with all submenus and menu items is contained in that scene.

STEP-5

Handling Actions

`NSFontManager` class seems to be quite useful, as there are more interesting properties and methods to explore and use. One such method that is particularly interesting is called `availableMembers(ofFontFamily:)`. It returns an array where each item is another array (`[[Any]]`) with four distinct values:

1. The PostScript name of the font as a String.
2. The type of the font, such as "Bold", "Italic", "Roman", "Light", etc.
3. The font's weight as an integer.
4. The font's traits as an unsigned integer (UInt).

An item-array is called a member of the font family. The following example taken from Apple Documentation illustrates what `availableMembers(ofFontFamily:)` returns:

```
(("Times-Roman", "Roman", 5, 4),
 ("Times-Italic", "Italic", 6, 5),
 ("Times-Bold", "Bold", 9, 2),
 ("Times-BoldItalic", "Bold Italic", 9, 3)
)
```

```
(("Times-Roman", "Roman", 5, 4),
 ("Times-Italic", "Italic", 6, 5),
 ("Times-Bold", "Bold", 9, 2),
 ("Times-BoldItalic", "Bold Italic", 9, 3)
)

```

Putting the above in different words, with `availableMembers(ofFontFamily:)` we can get all font variations that a font family supports, and display them on the `fontTypesPopup` popup button.
Before we make the actual use of the above method, let's declare the following two properties to the `ViewController` class, they'll become handy a bit later:
var selectedFontFamily: String?

var fontFamilyMembers = [[Any]]()

```
var selectedFontFamily: String?

var fontFamilyMembers = [[Any]]()

```

What we need to do now is to make it possible to keep the selected font family and get its contained members every time the `fontFamiliesPopup` selection gets changed. This must be done in the `handleFontFamilySelection(_:)` IBAction method:
@IBAction func handleFontFamilySelection(_ sender: Any) {
    if let fontFamily = fontFamiliesPopup.titleOfSelectedItem {

        selectedFontFamily = fontFamily

        if let members =
NSFontManager.shared.availableMembers(ofFontFamily: fontFamily) {
            fontFamilyMembers.removeAll()

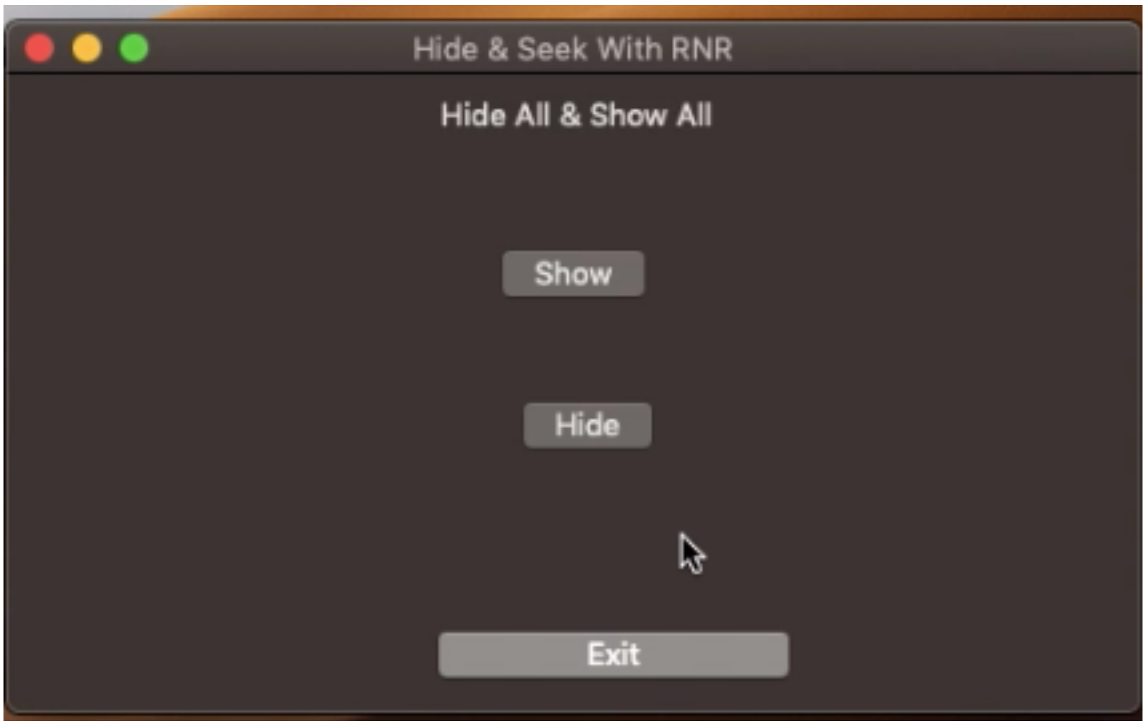
```
            fontFamilyMembers = members
        }
    }
}
```

```
@IBAction func handleFontFamilySelection(_ sender: Any) {
    if let fontFamily = fontFamiliesPopup.titleOfSelectedItem {

        selectedFontFamily = fontFamily

        if let members =
NSFontManager.shared.availableMembers(ofFontFamily:
fontFamily) {
            fontFamilyMembers.removeAll()
            fontFamilyMembers = members
        }
    }
}
```

STEP-6



Closing The Window

There is one last thing to do, and that is to enable the Close button so it's possible to close the window when clicking on it. Doing so is extremely easy as you can see right next:

```
@IBAction func closeWindow(_ sender: Any) {
   view.window?.close()
}
```

```
@IBAction func closeWindow(_ sender: Any) {
   view.window?.close()
}

```