
MVC-Clock

Features

The main features of the application are Displaying an updating a clock interface. The application provides three tabs when it is started for the first time, the first is called the Time Controller. This tab is responsible for allowing the user to update the time from the current time to any other time. This is handled in three parts. The first is allowing the user to update the time, when this option is selected the user is provided options for updating the hour and minute that the clock is set to. The second option is update the seconds, because mobile time pickers do not allow the user to update the seconds. This was done separately and will provide the user a selection between 1 and 59 to set for the the seconds on a clock. The third option allows the user to update the date, this is a full date picker, allowing selection of the year, month and day. One more option is provided to update the time, which is to set the date back to the current time and date.

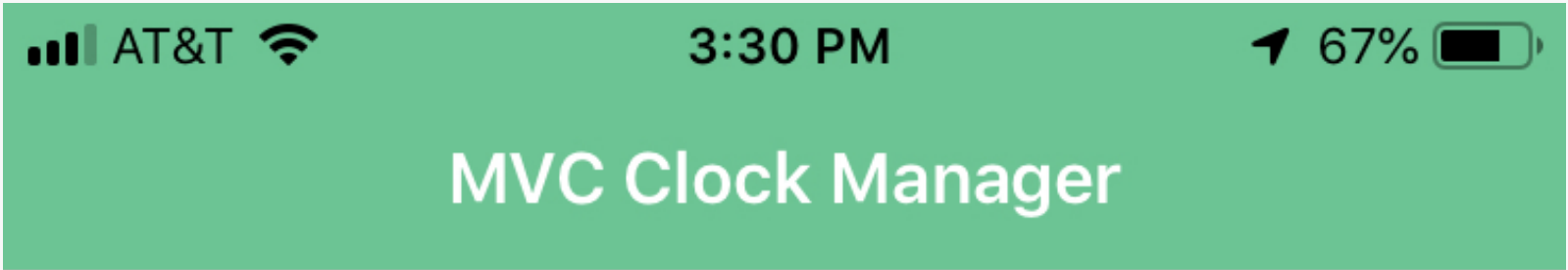
This application also provides the ability to undo and redo any time changes that are made using the 'Time Controller' interface. So those options are displayed on this screen for the user. Lastly, there are two options to add new tabs, which will correspond to clocks. The clocks that can be added are the same as the two already open below on the application. An analog clock and a digital clock, screenshots of both can be seen below.

The other two tabs open by default in the application display the time and are updated every second. The two tabs display the time in digital and analog fashion respectively. As previously discussed the user can create more of these two tabs to their hearts content.

Experience

I have often used the MVC patten in web development, however, doing this project I realized the formal design pattern and the usage in web frameworks is very different. I think much of this has to do with the fact that in web frameworks you are working with a stateless protocol, the result is that the controller does not need to receive updates from the model on changes, because it will not be updating the view with changes. I can see however, where in development in a stateful setting where the MVC pattern is very useful.

I choose to implement my MVC also including an event class, which handles much of the inter-class communication. I did this to further seperate the interactions of the classes and for a cleaner design. This made updating the controller with infromation from the model much more seamless and allowed the model to not have any prior knowledge about the controller. I personally think this is a good thing because it helped me enforce seperation between the different domains. The events also made communication from the views to the controller cleaner in that the view didn't have to worry about anything beyond having a function which would notify the controller. I enjoyed implementing the MVC pattern in Javascript, which was a little bit of challenge since Javascript thinks of classes different than Java. But it really helped me understand more of what is happening under the hood in Javascript.



UpdateTime

UpdateSeconds

UpdateDate

UpdateNow

Undo

Redo

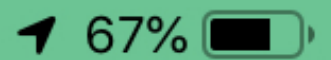
Add Digital Clock

Add Analog Clock

[Time Controller](#) Digital Clock Analog Clock Digital Clock Analog Clock



3:30 PM



MVC Clock Manager

October 6, 2018
12:12:52 PM

Time Controller [Digital Clock](#) Analog Clock Digital Clock Analog Clock

AT&T

3:30 PM

67%

MVC Clock Manager



Time Controller Digital Clock [Analog Clock](#) Digital Clock Analog Clock

UML Diagram

```

classDiagram
    class Event {
        _sender: Object
        _listeners: Array [Functions]
        +attach(listener):void
        +notify(args):void
    }
    class TimeModel {
        hour: Number
        minute: Number
        second: Number
        year: Number
        month: Number
        day: Number
        updateTimeEvent: Event
        updateTime(timeObj): void
        getTime(): Object
        setDate(year, month, day): void
        setTime(hour, minute): void
        setSeconds(seconds): void
        setNow(): void
        _increaseTime(): void
        init(): void
    }
    class TimeController {
        model: TimeModel
        views: Array [Object]
        redo: Array [Commands]
        redoPointer: Number
        init(): void
        setupHandlers(): TimeController
        enableModel(): void
        enableView(view): TimeController
        addView(view): void
        sendUpdate(sender, args): void
        setTime(sender, args): void
        setSecond(sender, args): void
        setDate(sender, args): void
        setNow(sender, args): void
        undo(sender, args): void
        redoIt(sender, args): void
    }
    class App {
        Controller: TimeController
        Tabs: Array
        addTab(Object): void
    }
    class DigitalClock {
        Controller: TimeController
        Date: Object
        dateDisplay(): String
        timeDisplay(): String
        updateDate(): void
    }
    class TimeInterface {
        setTimeEvent: Event
        setSecondEvent: Event
        setDateEvent: Event
        setNowEvent: Event
        undoEvent: Event
        redoEvent: Event
        datetimestpicker: ModalPicker
        Controller: TimeController
        setTime(): void
        setSecond(): void
        setDate(): void
        setNow(): void
        undo(): void
        redo(): void
        addDigital(): void
        addAnalog(): void
    }
    class AnalogClock {
        Controller: TimeController
        Date: Object
        hourValue(): Number
        dateDisplay(): String
        timeDisplay(): String
        updateDate(): void
    }
    class SecondDialog {
        selectedItem: Number
        listOffItems: Array
        selectedIndexChanged(): void
    }
    class TimeCommand {
        <<Interface>>
        +controller: TimeController
        +currentTime: Object
        +doIt(): void
        +undoIt(): void
    }
    class SetSecondsCommand {
        newSecond: Number
        Second: Number
        controller: TimeController
        doIt(): void
        undoIt(): void
    }
    class SetTimeCommand {
        newHour: Number
        newMinute: Number
        currentTime: Object
        controller: TimeController
        doIt(): void
        undoIt(): void
    }
    class SetDateCommand {
        newYear: Number
        newMonth: Number
        newDay: Number
        currentTime: Object
        controller: TimeController
        doIt(): void
        undoIt(): void
    }
    class SetNowCommand {
        currentTime: Object
        controller: TimeController
        doIt(): void
        undoIt(): void
    }

    Event <|-- TimeModel
    Event <|-- TimeController
    Event <|-- App
    Event <|-- DigitalClock
    Event <|-- TimeInterface
    Event <|-- AnalogClock
    Event <|-- SecondDialog

    TimeModel --> TimeController : Use
    TimeController --> Event : Use
    App --> TimeController : Use
    App --> Event : Use
    App --> DigitalClock : child
    App --> TimeInterface : child
    App --> AnalogClock : child
    TimeController --> TimeCommand : 1
    TimeCommand <|.. SetSecondsCommand
    TimeCommand <|.. SetTimeCommand
    TimeCommand <|.. SetDateCommand
    TimeCommand <|.. SetNowCommand
    TimeInterface --> TimeController : parent
    TimeInterface --> AnalogClock : parent
    SecondDialog --> TimeInterface : child
  
```

The diagram illustrates the architecture of a clock application. It features several classes and interfaces with their attributes, methods, and relationships.

- Event**: A base class for events, containing attributes `_sender: Object` and `_listeners: Array [Functions]`, and methods `+ attach(listener):void` and `+ notify(args):void`.
- TimeModel**: Inherits from **Event**. Attributes include `hour: Number`, `minute: Number`, `second: Number`, `year: Number`, `month: Number`, `day: Number`, and `updateTimeEvent: Event`. Methods include `updateTime(timeObj): void`, `getTime(): Object`, `setDate(year, month, day): void`, `setTime(hour, minute): void`, `setSeconds(seconds): void`, `setNow(): void`, `_increaseTime(): void`, and `init(): void`.
- TimeController**: Inherits from **Event**. Attributes include `model: TimeModel`, `views: Array [Object]`, `redo: Array [Commands]`, and `redoPointer: Number`. Methods include `init(): void`, `setupHandlers(): TimeController`, `enableModel(): void`, `enableView(view): TimeController`, `addView(view): void`, `sendUpdate(sender, args): void`, `setTime(sender, args): void`, `setSecond(sender, args): void`, `setDate(sender, args): void`, `setNow(sender, args): void`, `undo(sender, args): void`, and `redoIt(sender, args): void`.
- App**: Contains attributes `Controller: TimeController` and `Tabs: Array`, and a method `addTab(Object): void`. It has **Use** relationships with **TimeController** and **Event**, and **child** relationships with **DigitalClock**, **TimeInterface**, and **AnalogClock**.
- DigitalClock**: Contains attributes `Controller: TimeController` and `Date: Object`, and methods `dateDisplay(): String`, `timeDisplay(): String`, and `updateDate(): void`.
- TimeInterface**: Contains attributes `setTimeEvent: Event`, `setSecondEvent: Event`, `setDateEvent: Event`, `setNowEvent: Event`, `undoEvent: Event`, `redoEvent: Event`, `datetimestpicker: ModalPicker`, and `Controller: TimeController`. Methods include `setTime(): void`, `setSecond(): void`, `setDate(): void`, `setNow(): void`, `undo(): void`, `redo(): void`, `addDigital(): void`, and `addAnalog(): void`. It has a **parent** relationship with **AnalogClock**.
- AnalogClock**: Contains attributes `Controller: TimeController` and `Date: Object`, and methods `hourValue(): Number`, `dateDisplay(): String`, `timeDisplay(): String`, and `updateDate(): void`.
- SecondDialog**: Contains attributes `selectedItem: Number` and `listOffItems: Array`, and a method `selectedIndexChanged(): void`. It has a **child** relationship with **TimeInterface**.
- TimeCommand**: An interface (indicated by `<<Interface>>`) with attributes `+ controller: TimeController` and `+ currentTime: Object`, and methods `+ doIt(): void` and `+ undoIt(): void`. It has a **1** relationship with **TimeController** and is implemented by **SetSecondsCommand**, **SetTimeCommand**, **SetDateCommand**, and **SetNowCommand**.
- SetSecondsCommand**: Contains attributes `newSecond: Number`, `Second: Number`, and `controller: TimeController`, and methods `doIt(): void` and `undoIt(): void`.
- SetTimeCommand**: Contains attributes `newHour: Number`, `newMinute: Number`, `currentTime: Object`, and `controller: TimeController`, and methods `doIt(): void` and `undoIt(): void`.
- SetDateCommand**: Contains attributes `newYear: Number`, `newMonth: Number`, `newDay: Number`, `currentTime: Object`, and `controller: TimeController`, and methods `doIt(): void` and `undoIt(): void`.
- SetNowCommand**: Contains attributes `currentTime: Object` and `controller: TimeController`, and methods `doIt(): void` and `undoIt(): void`.

download this project, change the nativescript id in the package.json file to the value of your provisioning profile, and use the following commands:

```
# Install dependencies
npm install

# Build, watch for changes and run the application
tns run ios --bundle
```