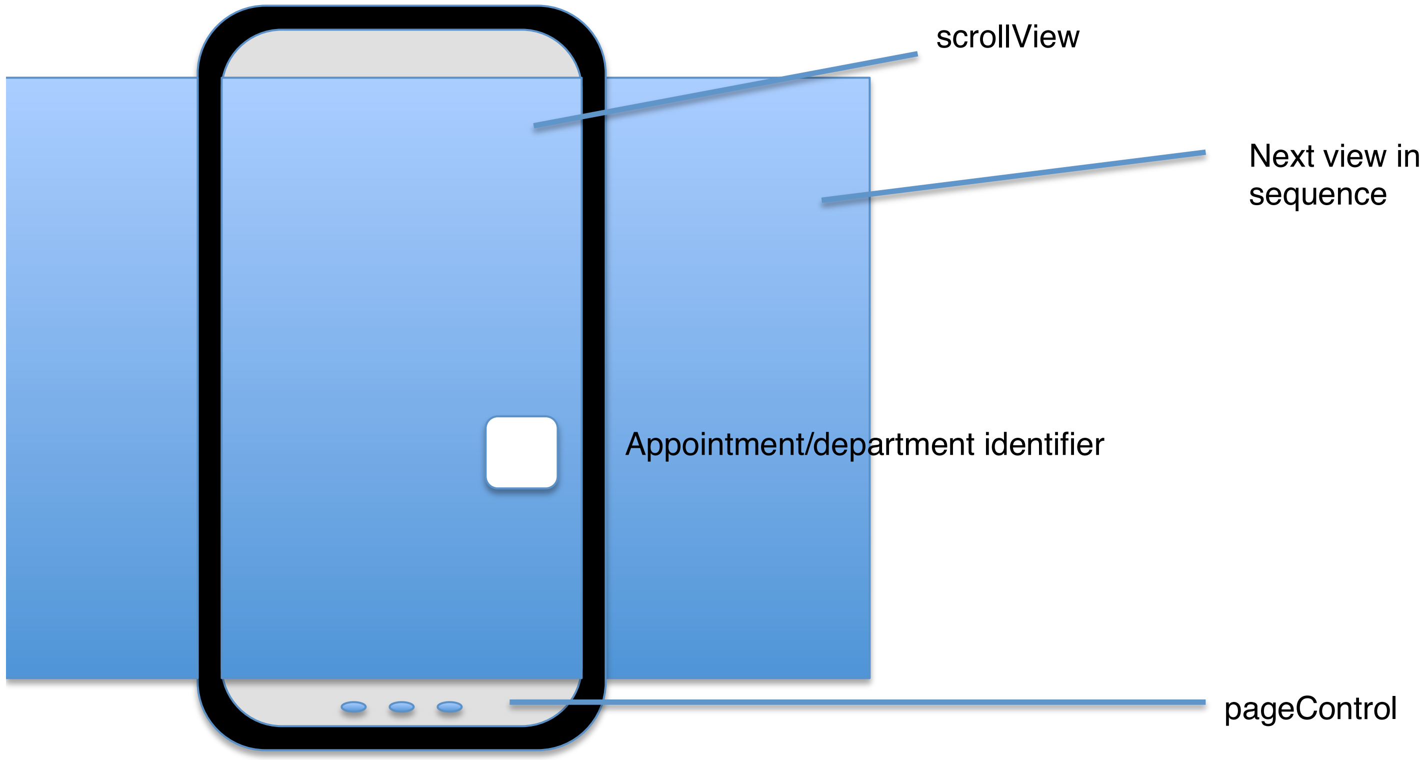
Student: Dani Rodriguez

Section: 5519



Project Description:

This user interface intentionally downplays the use of text/language, yet is not exactly a text-free UI. It is designed as a tool for case workers (Utah case workers in the case of the prototype) to demonstrate to newly arrived refugees the different types of documents that must accompany them to various appointments at social security administration offices and medical offices over the critical two week time period of initial resettlement. These two weeks are important to establishing status within the U.S., and understandably, may also be the introduction of culture shock.

The concept of this app came out of experience working across language differences with newly arrived refugee families when setting up these timely appointments and the particular difficulty in conveying which documents were necessary to bring to appointments a day or two away. Often the caseworker would need to remind and reinforce which documents were of importance at the time of pick-up, but without copies on hand, this could prove a challenging task at times. Striving to depart from the physical document, this app enforces the image and utilizes frameworks of the SDK that allow for ease of image browsing. In many circumstances, the image was the most fluid form of communication in lieu of Google mistranslations. This app is essentially a simple paged app that the user interacts with and that uses icons to signify it's importance to different departments/appointments.

Potential toward functionally assisting the caseworker to work across language barriers exist in this app through the camera and gallery function (in explaining non-familiar documents – in example, a lease or letter from the housing authority outlining aid awards are less familiar for some refugees than an I-94 or employee identification card, which must be taken to nearly every appointment). Future iterations could consider a visual display of time in coordination with different photo libraries. Ideally, audio translation would internationalize this app. However, that is out of scope of project 1 and likely this class.

In so far as complexity, the app is simple, but requires input from the user to select which office and service they would like to access, which promptly displays (in one view) a collection of images to swipe through. It could be designed in a couple of different effective ways. These varieties are detailed in the pseudocode below.

Pseudocode and Program Flow:

Though I could implement multiple UIImages that change with buttons pressed (different appointment types), similar to in the Beatles app from lab 4, I'd like to first try implementing a UICollectionView that is new to iOS 6.

Program Flow –

- Create subclass of UIViewController
- Set up an application delegate with collection view controller as root.
    - In AppDelegate.m

```
@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
   didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
- (BOOL)application:(UIApplication *)application
   didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
   CGRect frame = [[UIScreen mainScreen] bounds];
   self.window = [[UIWindow alloc] initWithFrame:frame];
   self.window.rootViewController = [
      [CollectionViewController alloc] init
   ];
   [self.window makeKeyAndVisible];
```

```
    return YES;
}

@end
```

- Create a custom collection view cell with a label
    - CustomCell.h

```
@interface CustomCell : UICollectionViewCell
@property (strong, nonatomic) UILabel *label;
@end
```

- Implement collection view controller and implement flow layout's delegate protocol to size the cells.
- Add protocol declaration to interface file, along with the delegate and data source protocols for the collection view.
    - In CollectionViewController.h

```
@interface CollectionViewController : UIViewController
<UICollectionViewDelegate,
    UICollectionViewDataSource,
    UICollectionViewDelegateFlowLayout>
@property (strong, nonatomic) UICollectionView *collectionView;
@end
```

- Use loadView method to create a base view.
- Make a paged interface by setting pagingEnabled to collection view

```
- (void)loadView
{
    self.view = [[UIView alloc]
        initWithFrame:[[UIScreen mainScreen] bounds]
    ];

    // Create a flow layout for the collection view that scrolls
    // horizontally and has no space between items
    UICollectionViewFlowLayout *flowLayout = [
        [UICollectionViewFlowLayout alloc] init
    ];
    flowLayout.scrollDirection = UICollectionViewScrollDirectionHorizontal;
    flowLayout.minimumLineSpacing = 0;
    flowLayout.minimumInteritemSpacing = 0;

    // Set up the collection view with no scrollbars, paging enabled
    // and the delegate and data source set to this view controller
    self.collectionView = [[UICollectionView alloc]
        initWithFrame:self.view.frame
        collectionViewLayout:flowLayout
    ];
    self.collectionView.showsHorizontalScrollIndicator = NO;
    self.collectionView.pagingEnabled = YES;
    self.collectionView.delegate = self;
    self.collectionView.dataSource = self;
    [self.view addSubview:self.collectionView];
}
```

- Add rest of setup in viewDidLoad method to register prototype cell class for the collection view
  - CollectionViewController.m

```
- (void)viewDidLoad
{
   [super viewDidLoad];

   // Register a prototype cell class for the collection view
   [self.collectionView registerClass:[CustomCell class]
      forCellWithReuseIdentifier:@"Cell"
   ];
}
```

- Feed collection view with our pages, implement methods of collection view's data source protocol.
  - In CollectionViewController.m

```
- (NSInteger)collectionView:(UICollectionView *)collectionView
   numberOfItemsInSection:(NSInteger)section
{
   return PAGES;
}

- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView
   cellForItemAtIndexPath:(NSIndexPath *)indexPath
{
   // Dequeue a prototype cell and set the label to indicate the page
   CustomCell *cell = [collectionView
      dequeueReusableCellWithReuseIdentifier:@"Cell"
      forIndexPath:indexPath
   ];
   cell.label.text = [NSString stringWithFormat:@"Page %d",
      indexPath.row + 1
   ];

   // To provide a good view of pages, set each one to a different color
   CGFloat hue = (CGFloat)indexPath.row / PAGES;
   cell.backgroundColor = [UIColor
      colorWithHue:hue saturation:1.0f brightness:0.5f alpha:1.0f
   ];

   return cell;
}
```

- Implement a method of the flow layout delegate to tell the collection view's layout manager how big the cells should be. Make the cells fill the collection view's bounds.
  - CollectionViewController.m

```
- (CGSize)collectionView:(UICollectionView *)collectionView
   layout:(UICollectionViewLayout *)collectionViewLayout
   sizeForItemAtIndexPath:(NSIndexPath *)indexPath
```

```
{
   return collectionView.bounds.size;
}

@end
```

Alternatively, if this framework proves too challenging, I will use a scrollView in the UIView and a PageControl object, enabling scrolling and paging.

Podcasted tutorials from the WWDC 2009 – 2013 sessions are my main resource in developing this application, in complement to the Developer pages and the tutorial listed below:

- http://iosmadesimple.blogspot.com/2013/01/page-control-for-switching-between-views.html

Project Testing:

As I don’t own any iOS device, testing will have to be done on a lab mac with a borrowed iPad. No devices are currently registered to my personal mac. Therefore, testing during office hours is my course of action.