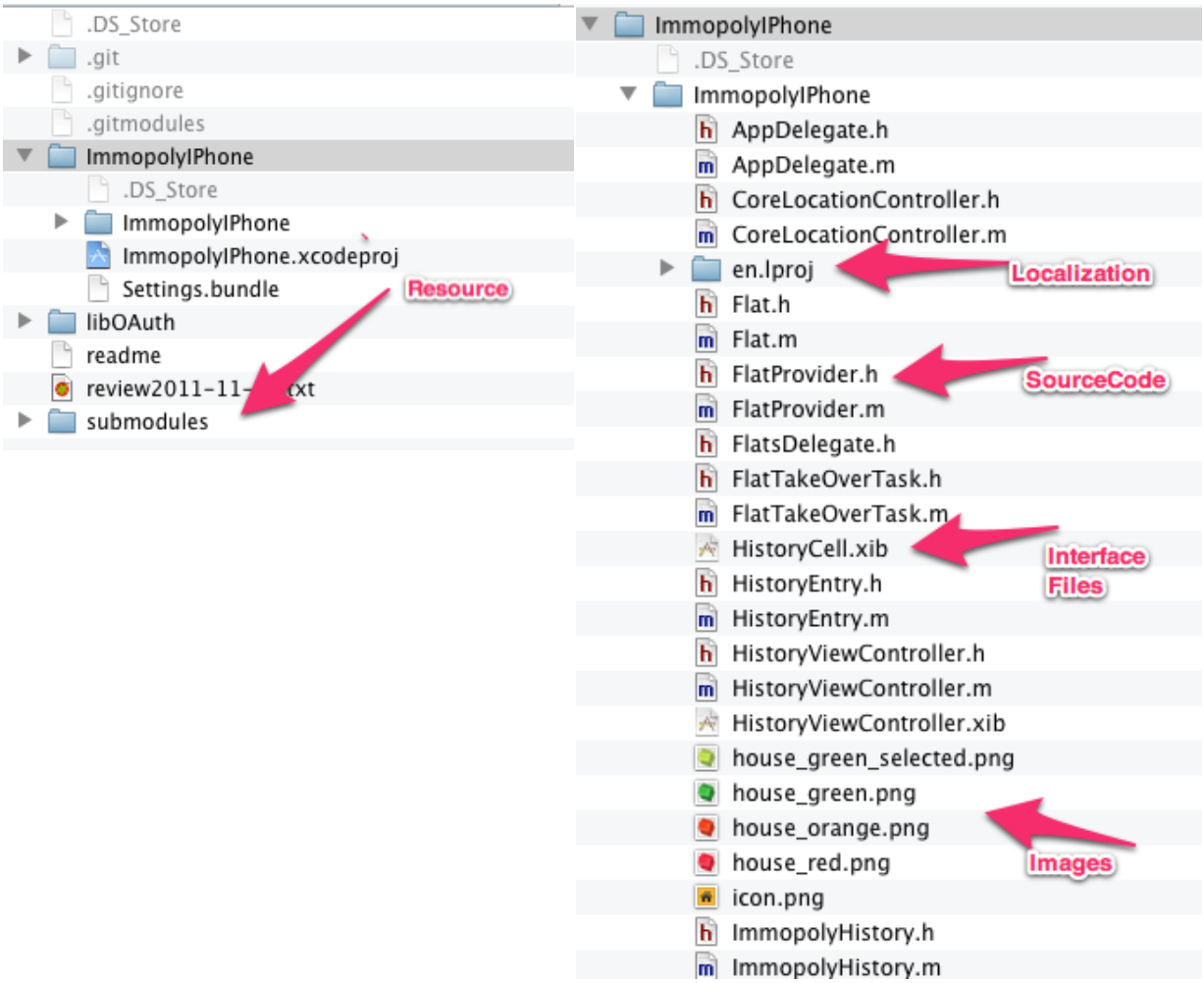
Immopoly iPhone

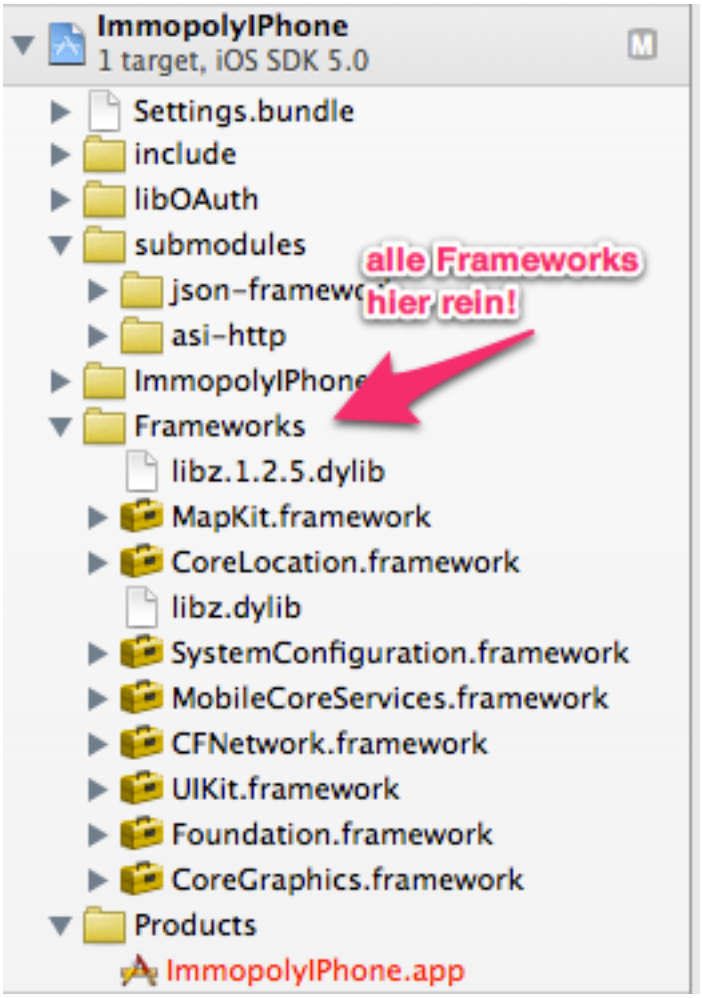
1. Organisiert euer Projekt auf Dateiebene



Unterordner für Resourcen (dort einen Unterordner für XIBs)
Unterordner für Quelltext
Unterordner für Localisierungsdatein

Macht das lieber jetzt, als irgendwann am Ende des Projekte wenn ihr den Überblick schon verloren habt.

Übernehmt die Struktur nach möglichkeit auch im XCode Projekt.
Nutzt git move (tower macht das automatisch) damit die History von Datein nicht verloren geht.
2. Organisiert euer XCode Projekt
XCode organisiert die Projektdatein nicht sonderlich gut, daher müsst ihr auch das selber machen :(



Gut ist, ihr habt schon „Packages“ also Ordner angelegt um die QuellcodeDatein übersichtlich zu verteilen.

3. Vermeidet Strings die keine Konstanten sind

```
// observer for login of user error
    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(handleErrorMsg:) name:@"user/login fail" object:nil];

    // observer for register of user error
    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(handleErrorMsg:) name:@"user/register fail"
object:nil];

    // observer for parsing flat data error
    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(handleErrorMsg:) name:@"flatProvider/parse fail"
object:nil];

    // observer for taking over a flat error
    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(handleHistoryResponse:) name:@"portfolio/add"
object:nil];
    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(handleErrorMsg:) name:@"portfolio/add fail"
object:nil];
Schreibt die am besten in eine Constants.h mit Constants.m
```

und definiert dort
Constants.h:
```
extern NSString* const myConstString;
```
Constants.m:
```
NSString* const myConstString = @"myConstString";;
```
4. Eindeutiges Shema für Variabel Namen:
```
CoreLocationController *CLController;
CLGeocoder *geocoder;
```

IVARS könnte man z.B. immer mit einem _ Anfangen lassen, ansonsten kleingeschrieben mit Camel Case.

5. Einheitliches Formatting von Methoden/Quelltext:
```
- (void) startLocationUpdate;
- (void) geocodeLocation:(CLLocation *)location;
-(void)handleHistoryResponse:(NSNotification *)notification;
-(void)handleErrorMsg:(NSNotification *)notification;
- (void)enableAutomaticLogin;
```

Gewöhnt euch jetzt schon an, alle einen Formatterstil zu verwenden, wir sind gerade in der Arbeit dabei da eine Lösung für automatisiertes Formatting zu finden, solange muss man selber ran und die Leerzeichen an den richtigen Stellen setzen

Ich bin auch ein Fan davon keine Abkürzungen in Methoden zu verwenden. also kein
```
handleErrorMsg
CLController.locMgr
```
sondern
```
handleErrorMessage
CLController.locationManager
```

Außerdem, macht vielleicht aus
```
enableAutomaticLogin
- (void)enableAutomaticLogin;
```
ein Property?

und wenn wir gerade dabei sind, Warum schreibt ihr den String „YES“ in die Properties:
```
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    NSDictionary *appDefaults = [NSDictionary
dictionaryWithObject:@"YES" forKey:@"saveToken"];
    [defaults registerDefaults:appDefaults];
    [defaults synchronize];
```

kann aber auch sein dass das einen höreren Sinn hat :) Auch hier wieder @“saveToken„ als Konstante auslagern weil ihr sie garantiert nochmal woanders benutzt

6. Helper Methoden auslagern:
Eurer AppDelegate hat folgende Methode:
```
// method for converting lat and long from location to user friendly
address
- (void)geocodeLocation:(CLLocation *)location{
```
Ich hab die mir mal angguckt, und die setzt aber ein Label:
```
[ImmopolyManager instance].delegate setAdressLabelText:[placemark
name]];
```

da muss im Methodennamen klarer kommuniziert werden was die Methode macht.

7. einheutliche Namen für Methoden:
```
- (void)performLogin:(NSString *)userName password:(NSString *)password;
- (void)performLoginWithToken:(NSString *) userToken;
```
wird zu:
```
- (void)performLoginWithUsername:(NSString *)userName andPassword:
(NSString *)password;
- (void)performLoginWithToken:(NSString *) userToken;
```

8. UserLoginTask
Schaut euch mal AFNetworking[1] an. Das ist ein schöner Wrapper um den NSURLRequest, damit könnt ihr euch den ganzen Quatsch mit `[NSURLConnection connectionWithRequest:request delegate:self]]` und den ganzen Delegate Methoden sparen und euch auf das wesentliche konzentrieren. Oder schaut euch ASIHTTPRequest mal genau an. Der ist schon im Projekt und mach die Arbeit mit HTTP Requests einfacher. ASI kann sehr viel, ist dadurch aber vielleicht auch verwirrender, AFNetwoking ist einfacher, reicht aber warscheinlich.

9. Andere Konstanten
```
    NSURLRequest *request = [NSURLRequest requestWithURL:url
cachePolicy:NSURLRequestReturnCacheDataElseLoad timeoutInterval:30.0];
```

macht zumindest ein #define aus der 30 falls ihr sie später Projektübergreifend ändern wollt.

10. Ich sehe gerade ihr habt viele „Tasks“
schaut euch mal die Klasse NSOperation an. Operations sind toll! Man kann sie in Queues werfen, Dependencies definieren, Thread Prioritäten definieren und noch vieles mehr. So sollte man Multithreading betreiben.

11. Header sauber halten
siehe meine Änderunden im Review Branch in der Klasse `FlatTakeOverTask`

Anonyme Kategorien heißt das Zauberwort.

Außerdem, wenn wir gerade bei der Sache sind:
warum:
```
FlatTakeOverTask *flatTask = [[FlatTakeOverTask alloc]init];
[flatTask takeOverFlat:[self selectedImmoscoutFlat]];
```
und nicht:
```
    FlatTakeOverTask *takeOverFlatTask = [[FlatTakeOverTask
alloc]initWithTakeoverFlat: [self selectedImmoscoutFlat]];
[takeOverFlatTask start];
```

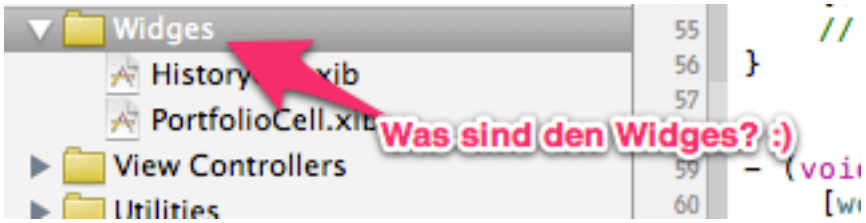
Außerdem :)
Warum:
```
-(void) displayUserData {
    FlatTakeOverTask *flatTask = [[FlatTakeOverTask alloc]init];
    [flatTask takeOverFlat:[self selectedImmoscoutFlat]];
    [self.view removeFromSuperview];
```

---

[1] https://github.com/AFNetworking/AFNetworking

}
heißt die Methode displayUserData wenn sie ein Flat übernimmt?

12. LOL



13. @synthesize

```
@synthesize tvCell, table, segmentedControl, portfolioMapView,
loginCheck,calloutBubble,isOutInCall,isCalloutBubbleIn,selectedExposeId,
selViewForHouseImage,selectedImmoScoutFlat,lbFlatDescription,lbFlatName,
lbFlatPrice,lbLivingSpace,adressLabel,lbNumberOfRooms,exposeWebViewContr
oller, spinner;
```

da sieht doch keiner mehr durch oder? Wir haben uns bei uns in der Firma angewöhnt pro @synthesize eine Zeile zu schreiben, das ist zwar aufwendiger aber übersichtlicher.

Wo wir gerade dabei sind, wenn ihr so viele Properties habt, warum sieht euer dealloc dann so aus:

```
- (void)dealloc {
    [segmentedControl release];
    [loginCheck release];
    [spinner release];
    [super dealloc];
}
```

und warum werden die ganzen anderen Properties nicht freigegeben?
und warum wird nichts freigegeben in viewDidUnload:

```
- (void)viewDidUnload
{
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}
Release any retained subviews of the main view.
e.g. self.myOutlet = nil;
```

14. Benutzt mehr Blocks, Blocks sind cool!

```
[UIView beginAnimations:nil context:NULL];
            [UIView setAnimationDuration:0.4];
            posMap = portfolioMapView.center;
            posTable = table.center;
            posMap.x = 480.0f;
            posTable.x = 160.0f;
            portfolioMapView.center = posMap;
            table.center = posTable;
            [UIView commitAnimations];
```

Ist okay, aber:

```
[UIView animateWithDuration:0.4 animations:^{
                posMap = portfolioMapView.center;
                posTable = table.center;
                posMap.x = 480.0f;
                posTable.x = 160.0f;
                portfolioMapView.center = posMap;
                table.center = posTable;
            }];
```

ist besser! Blocks werden immer wichtiger im Framework, also habt keine Angst davor, nutzt sie. Hab ich schon gesagt, dass Blocks eigentlich voll cool sind?!

Hab das auch mal commited... siehe review branch...

auch cool:
anstelle von

```
for(Flat *flat in [[ImmopolyManager instance] immoScoutFlats]) {
        [mapView addAnnotation: flat];
    }
```

die hier benutzen:

```
[[[ImmopolyManager instance] immoScoutFlats]
enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
        [mapView addAnnotation: obj];
    }];
```

dann muss ich mir auch keine Gedanken machen ob in der Foreach loop jedes mal [[ImmopolyManager instance] immoScoutFlats] für jede Iteration aufgerufen wird... das könne nämlich teuer sein.

15. Strings
Wir haben uns angewöhnt, nicht die Deutschen/Englischen Worte als String identifier zu verwenden....

```
    NSString *rooms = [NSString stringWithFormat:@"Zimmer: %d",
[selectedImmoScoutFlat numberOfRooms]];
    NSString *space = [NSString stringWithFormat:@"Fläche: %f qm",
[selectedImmoScoutFlat livingSpace]];
    NSString *price = [NSString stringWithFormat:@"Preis: %f €",
[selectedImmoScoutFlat price]];
```

Öhm, Wieso benutzt ihr eigentlich keine NSLocalizedStrings?

```
    NSString *rooms = [NSString stringWithFormat:@"Zimmer: %d",
[selectedImmoScoutFlat numberOfRooms]];
    NSString *space = [NSString stringWithFormat:@"Fläche: %f qm",
[selectedImmoScoutFlat livingSpace]];
    NSString *price = [NSString stringWithFormat:@"Preis: %f €",
[selectedImmoScoutFlat price]];
```

Wenn ihr dann NSLocalizedString benutzt, überlegt ob ihr den Text als Key nehmen wollt oder einen eineindeutigen Identifier (so wie man das bei Android auch macht)

```
NSLocalizedString(@"Map", @"First");
```

oder lieber

```
NSLocalizedString(@"title map view", @"Title of the map
viewcontroller");
```

16. vermeidet doppelten Code

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation
{
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
```

habe ich jetzt schön oft gesehen, macht doch lieber eine Mutterklasse von der alle eure ViewController erben und schreibt das dort einmal rein.

17. Singletons

es gibt direkt von Apple eine Patternempfehlung[2] wie man Singletons implementieren solle.

außerdem gibt es einen coolen neuen weg, mit Grand Central Dispatch, das hört sich nicht nur cool an, ist auch sehr pratisch.

http://stackoverflow.com/questions/5720029/create-singleton-using-gcds-dispatch-once-in-objective-c

```
+ (id)sharedFoo
{
    static dispatch_once_t once;
    static MyFoo *sharedFoo;
    dispatch_once(&once, ^ { sharedFoo = [[self alloc] init]; });
    return sharedFoo;
}
```

reicht erstmal :)

[2] http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaObjects/CocoaObjects.html#//apple_ref/doc/uid/TP40002974-CH4-SW32