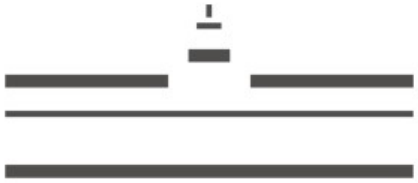


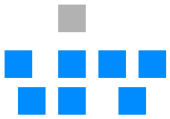
Westfälische
Wilhelms-Universität
Münster

Programmierung für mobile Endgeräte

Core Data (III) und nutzerdefinierte View-Bausteine

wissen.leben
WWU Münster

Name: Patrick Förster



ZENTRUM FÜR
INFORMATIONS
VERARBEITUNG

Wiederholung Xib-Bausteine

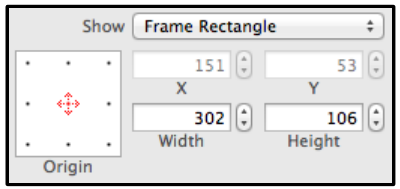
- Vor den Spielplan in der EM-Applikation wurde ein komplett eigener View entworfen, der über ein Delegate Daten geliefert bekommt
- Diese Daten wurden für die Anzeige auf Zellen verteilt, die die Darstellung und das Layout der Daten vorschrieben
- Da der Spielplan so designed wurde, dass er eine dynamische Anzahl an Daten darstellen kann, können die nötigen Zellen nicht im Storyboard vordefiniert werden
- Die Zellen müssen dynamisch passend zur Datenmenge erzeugt werden
- Um ein Prototyp für die Datenzelle festzulegen wurde eine Xib-Datei verwendet, die – analog zum Storyboard – über den Interface-Builder designed werden kann
- Das Design wurde um eigenen View-Klassen bereichert, die per Outlets und Properties Zugriff auf die Layoutelemente zulassen
- Zur Laufzeit werden dann die Xib-Bausteine dynamisch erzeugt und mit den nötigen Daten gefüllt
- Alternativ zu den Xib-Dateien können Bausteine natürlich auch programmatisch zur Laufzeit zusammengestellt werden

Interaktion

- Bisher war jegliche Interaktion mit dem Dargestellten auf vordefiniertes Verhalten der Views und unter Umständen auf das Implementieren eines Delegates zurückzuführen
- Mit dem eigenen View-Baustein existieren keine vordefinierten Interaktionsmöglichkeiten (es gibt zurzeit keinerlei Buttons o.Ä.)
- Jeder Baustein erbt von UIView und damit von UIResponder und ist somit in der Lage auf einfache Touch-Ereignisse zu reagieren:

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event;
```

- Den Platz den eine UI-Komponenten in Anspruch nehmen kann, wird durch eine Bounding-Box (Frame) definiert
- Die oberste Komponente, deren Bounding-Box die Position eines Touches einschließt und die touchBegan-Methode implementiert bekommt den Zuschlag



Storyboard: Size-Inspector

Interaktion (Beispiel)

- Als Einführungsbeispiel wird für den Spielplan ein „Highlight“-Effekt implementiert:
  - Bei einem Touch soll die Datenzelle des berührten Spiels kurz „abdunkeln“
  - Um solcherlei grafische Effekte umsetzen zu können, muss das QuartzCore-Frame in das Projekt eingebunden werden
  - Der PGSchedulerSubcell-Klasse wird – um zur Ausgangsfarbe zurückkehren zu können – ein neues Attribute vom Typ UIColor* hinzugefügt und letztendlich die touchesBegan/Ended: withEvent Methoden implementiert

```
#import <QuartzCore/QuartzCore.h>...
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    _b = [UIColor colorWithCGColor:[self.backgroundColor CGColor]];
    const CGFloat* components = CGColorGetComponents([_b CGColor]);
    CGFloat red = components[0];
    CGFloat green, blue, alpha = components[0];
    self.backgroundColor = [UIColor colorWithRed:red-0.3f green:green-0.3f blue:blue-0.3f alpha:alpha];}

-(void) touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
        self.layer.backgroundColor = [_b CGColor];
}
```

Interaktion (Beispiel II)

- Sobald ein Touch auf einer Datenzelle ausgelöst wurde, wird die aktuelle Hintergrundfarbe der Zelle zwischengespeichert und der Hintergrund mit einer etwas dunkleren Farbe (-0.3) neu gesetzt
- Sobald der Touch endet wird der Hintergrund wieder zurückgesetzt
- Auf den ersten Blick mag nicht deutlich sein, wo der Unterschied zwischen touchEnded und touchCancelled ist, vor allem weil der Nutzer keine Möglichkeit hat aktiv zwischen dem regulärem Beenden und dem Abbrechen eines Touches zu unterschieden
- touchCancelled wird immer dann aufgerufen, wenn das iOS entscheidet, dass die aktuelle Eingabe nicht mehr valide beendet werden kann, z.B. wenn während einer Berührung die „Home“ Taste gedrückt wird
- Im Beispiel würde die Hintergrundfarbe nie zurückgesetzt werden, weshalb die touchCancelled-Methode analog zu touchEnded implementiert werden sollte
- Man sollte also stets unterscheiden, ob die Logik hinter einer Eingabe bereits bei der Berührung oder erst beim Loslassen des Bildschirms auszuführen ist

Die EM-App (Match-Details)

- Das Highlighting eines Spiels soll nicht das Ende der Interaktion im Spielplan gewesen sein
- Mit dem Touch, soll jetzt eine Detailansicht als Popover zu dem ausgewählten Spiel angezeigt werden
- Um erneut eine klare Trennung zwischen View und Controller zu haben, wird dem Delegate-Protokoll eine neue Methode hinzugefügt, die dann bei einem Klick auf eine Datenzelle aufgerufen wird, um die Login an den Controller zu delegieren:

```
-(void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
        PGSchedulerView* pgView = (PGSchedulerView*) self.superview.superview;;
        CGFloat x = self.superview.frame.origin.x + self.frame.origin.x;
        CGFloat y = self.superview.frame.origin.y + self.frame.origin.y;
        CGRect frame = CGRectMake(x, y, self.frame.size.width, self.frame.size.height);
        [pgView.delegate schedulerView:pgView didSelectRowAtCellIndex:self.index withinFrame:frame];
    }
```

- Die neue Protokoll-Methode heißt didSelectRowAtCellIndex:withFrame und ihr wird zum einen der Index der berührten Zelle, sowie die Größe und die Position der Zelle (relative zum Root-View) mitgegeben

Die EM-App (Match-Details II)

- Im zugehörigen Controller kann nun die Protokoll-Methode implementiert und ein Übergang zur Detailansicht eingeleitet werden
- Dazu wird ein neuer (getaggter) ViewController im Storyboard, sowie eine neue Controller- und für die Outlets eine neue UIView-Klasse benötigt
- Da später über diese Ansicht auch Ergebnisse eingetragen werden sollen, wird die Ansicht nicht direkt im Storyboard, sondern analog zu den Datenzellen über eine neuen Xib-Baustein erstellt
- Bisher wurde den Views immer der gesamte Anzeigebereich zur Verfügung gestellt und dabei vorherige ausgeblendet; mit dem Popover-Controller werden Views *innerhalb* einer ausgewählten Ansicht angezeigt:

```
    UIViewController* controller = [self.storyboard instantiateViewControllerWithIdentifier:@"vc:details:match"];
    PGMatchDetails* details = (PGMatchDetails*) [PGUIFactory createViewFromXib:@"schedule_match"
widthTag:PG_SCHEDULE_MATCHVIEW andFrame:CGRectMake(0, 0, 200, 200) forOwner:self];
    [controller.view addSubview:details];
    _popover = [[UIPopoverController alloc] initWithContentViewController:controller];
    [_popover setPopoverContentSize:rect.size];
    [_popover presentPopoverFromRect: frame inView:self.view permittedArrowDirections:UIPopoverArrowDirectionAny animated:YES];
```

Die EM-App (Match-Details III)

- Beim Erzeugen des Popover-Controllers ist Vorsicht geboten: Damit der Controller den Methodenaufruf übersteht, muss an irgendeiner Stelle eine Referenz auf den Controller gehalten werden!
- Der ViewController für die Spielansicht bietet sich hier an, da man später vielleicht über den Controller das Ausblenden des Popoveres kontrollieren möchte (Der Popover-View wird automatisch ausgeblendet, sobald ein Punkt außerhalb seines Bereichs berührt wird)
- Damit die nötigen Spieldaten in die Detailansicht übertragen werden können, muss der zugehörige Controller über die Daten informiert werden
- Innerhalb der didSelectRowAtCellIndex Methode kann man über den mitgelieferten Zellenindex das zugehörige Match herausgefunden und weitergeben

Edit-Ansicht

- Damit im Spielplan zwischen normaler Ansicht und editierbarer Ansicht unterschieden werden kann, wird dem View ein Toolbar und diesem ein Button mit der Aufschrift „Edit“ hinzugefügt
- Der Button wird durch eine IBAction im ScheduleViewController verbunden

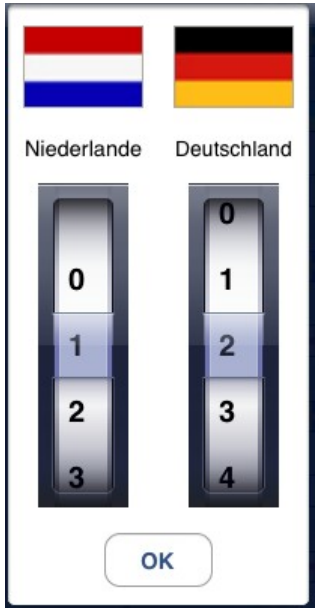
```
UIBarButtonItem* button = (UIBarButtonItem*) sender;

[self setEditing: !self.editing];

if (self.editing) {
    button.title = @"Edit beenden";
    button.tintColor = [UIColor redColor];
}
else {
    button.title = @"Edit";
    button.tintColor = nil;
}
```

- Jede UIView-Komponente verfügt von Haus aus über das Flag „editing“, dessen Wert hier einfach „getoggelt“ und der Button-Text dementsprechen angepasst wird

UIPickerView

- Im Edit-Modus soll über das Popover das Ergebnis eines Spiels eintragen werden
- Dazu wird die UI-Komponente UIPickerView genutzt
- Mit der neuen Komponente kann nun ein neuer Baustein in der Xib-Datei für die Match-Ansicht definiert und eindeutig getagged werden
- Innerhalb des Match-Controllers kann anhand des Editstatus der richtige Baustein mit Hilfe des vergebenen Tags geladen werden

```
-(PGMatchDetails*) loadDetailsViewWithFrame: (CGRect) frame {
    NSUInteger tag = self.editing
        ? PG_SCHEDULE_MATCHVIEW_EDIT
        : PG_SCHEDULE_MATCHVIEW;
    return (PGMatchDetails*) [PGUIFactory createViewFromXib:@"schedule_match"
withTag: tag andFrame:frame forOwner:self];
}
```



- Auch der UIPickerView erlangt seine Daten und auch sein Aussehen über Delegates (UIPickerViewDataSource, UIPickerViewDelegate), die durch einen Controller implementiert werden sollten

File's Owner

- Nach dem MVC-Prinzip sollten die Delegates Controllern sein
- Im Interface-Builder wurden bislang allerdings lediglich View-Bausteine ohne Verbindung zu einem Controller erstellt
- Erst beim Laden wird über das owner-Attribut gesetzt, wer für den Baustein verantwortlich ist (nach MVC hoffentlich ein Controller!)
- Im Interface-Builder wird der Eigentümer des Xibs über den Platzhalter „File's Owner“ repräsentiert

- Im Interface-Builder wird der Eigentümer des Xibs über den Platzhalter „File's Owner“ repräsentiert
- Über den „Identity-Inspector“ kann die erwartete Klasse definiert werden, so dass der Interface-Builder evtl. vorhandene IBOutlets und IBActions des Controllers erkennt
- Der Eigentümer des Match-Xib ist der PGMatchViewController sein, der nun als Delegate für die UIPickerViews über den File's Owner Platzhalter gesetzt werden kann

UIPickerViewDataSource, UIPickerViewDelegate

- Über die Methoden für das DataSource-Delegate wird definiert, wie viele Komponenten der UIPickerView haben soll und wie viele Einträge pro Komponente vorhanden sind
- Für die Match-Ansicht sind beide trivial: Es gibt eine Komponente, in die Werte von 1 – 10 zugelassen sein sollen

```
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView {return 1;}

- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:(NSInteger)component {return 10;}
```

- Für das View-Delegate wird nur eine Methode implementiert, die den anzuzeigenden Text für eine bestimmte Reihe in einer Komponente definiert
- Auch dies ist für die Match-Ansicht trivial, denn angezeigt werden soll einfach der Index der aktuellen Reihe:

```
- (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row forComponent:(NSInteger)component {
    return [NSString stringWithFormat:@"%i", row];
}
```

Speichern

- Um den Ok-Button mit einer IBAction aus dem Match-Controller zu verbinden, muss die zugehörige Klasse als File's Owner eingetragen sein
- Damit das eingetragene Ergebnis persistiert wird, müssen die Werte der UIPickerViews auf das angezeigte Match übertragen...

```
PGMatchDetails* details = (PGMatchDetails*) [self.view viewWithTag:PG_SCHEDULER_MATCHVIEW_EDIT];
if (_match) {
    _match.scoreHome = [NSNumber numberWithInt: [view.homePicker selectedRowInComponent:0]];
    _match.scoreGuest = [NSNumber numberWithInt: [view.guestPicker selectedRowInComponent:0]];
}
```

- … und der geänderte Status des Spiels im Kontext gespeichert werden

```
NSManagedObjectContext* ctx = [PGEntitySetup sharedInstance].managedObjectContext;
NSError* error;
if ([ctx save:&error]) {  /*...*/ }
```

- Das Spielergebnis wird jetzt zwar gespeichert, allerdings wird der Spielplan nicht aktualisiert

Aktualisieren der Ansicht

- Damit die Ansicht aktualisiert werden kann, benötigt der Match-Controller Zugriff auf den View des Spielplan-Controllers
- Der Match-Controller muss also um eine neue Eigenschaft erweitert werden, die in der schedulerView:didSelectRowAtCellIndex:withFrame-Methode des Spielplan-Controllers gesetzt werden sollte
- Des Weiteren wird der PGScheduleView um eine Methode erweitert, die zu einem bestimmten Zellenindex die zugehörige Datenzelle heraussucht:

```
-(PGScheduleSubcell*) getCellWithIndex: (PGCellIndex*) index {
    for (UIView* temp in self.subviews) {
        PGScheduleSubcell* cell = (PGScheduleSubcell*) [temp viewWithTag:PG_SCHEDULE_SUBCELL];
        if (cell.index.row == index.row && cell.index.column == index.column &&  cell.index.index == index.index) {
            return cell;
        }
    }

    return nil;
}
```

Aktualisieren der Ansicht (II)

- Die Grundlage der Aktualisierungsroutine wurde bereits für den ersten Aufbau der Ansicht definiert und kann übernommen werden, in dem die Operationen zur Berechnung des Layouts ausgelassen werden
- Unter zur Hilfenahme der getCellWithIndex Methode, wird das Aktualisieren dann einfach an die Delegate Methode setup weitergeleitet und somit wieder an den Spielplan-Controller übergeben

```
-(void) reloadData {
    ...
            PGCellIndex* index = [[PGCellIndex alloc] initWithRow:row column:i andIndex:j];
            Match* match = [_delegate scheduleView:self matchForIndex:index];
            if (match) {
                NSDictionary* details = [_delegate scheduleView:self detailsForIndex:index];
                PGScheduleSubcell* subcell = [self getCellWithIndex:index];
                [_delegate setup:subcell withMatch:match andDetails:details];
            }
}
```

- Die reloadData-Methode muss noch in der IBAction des Match-Controllers aufgerufen werden

Editieren in Tabellen

- Wie bereits erwähnt, verfügt jeder View über das Flag editing, über das festgelegt wird, ob ein bestimmter View editiert werden darf
- Für die Spielplan-Ansicht wurde dieses Verhalten per Hand programmiert
- Der schon des Öfteren eingesetzte TableView besitzt von Haus aus eine Editiermöglichkeit zum Löschen von Einträgen
- Der zugehörige Edit-Button ist bereits vordefiniert und muss nur aktiviert werden (hier angenommen, der View befindet sich in einer Navigation-Hierarchie):


```
- (void)viewDidLoad {
    ...
    self.navigationItem.rightBarButtonItem = self.editButtonItem;
}
```

Editieren in Tabellen (II)

- Möchte man das Verhalten beim Aktivieren des Edit-Modus selbst steuern, so kann die setEditing:animated-Methode überschrieben werden
- Mit dem Einbinden des Edit-Buttons kann der Nutzer bereits in eine Ansicht wechseln, die die Möglichkeit bietet einen bestimmten Tabelleneintrag zum Löschen auszuwählen
- Das Löschen an sich muss allerdings noch implementiert werden (woher soll die Ansicht auch wissen, was Löschen für die angezeigten Daten bedeutet)
- Mit dem Klick auf den „Delete“-Button wird die Delegate-Methode tableView:commitEditingStyle:forRowAtIndexPath aufgerufen, in der dann das eigentliche Löschen durchgeführt werden kann
- Ähnlich wie beim Speichern im Spielplan, muss nach einer Löschoperation auf der Persistenzschicht auch die Ansicht aktualisiert werden
- Zum Löschen einer Zeile aus der Tabelle verfügt die TableView-Klasse über:

```
- (void)deleteRowsAtIndexPaths:(NSArray *)indexPaths withRowAnimation:(UITableViewRowAnimation)animation;
```

- Analog verfügt die Klasse über Methoden zum Einfügen, Neuladen, usw.

Löschen in der Team-Ansicht

- Die Team-Ansicht ist kein reiner TableView, sondern wurde durch eine eigene View-Klasse repräsentiert, die mittig einen TableView über die Spieler anbietet
- D.h. um bei einem Klick auf den Edit-Button auch die mittige Tabelle in die Editieransicht umzuschalten muss die setEditing:animated-Methode überschrieben werden

```
- (void)setEditing:(BOOL)editing animated:(BOOL)animated {
    [super setEditing:editing animated:animated];      //   zu erst super.. .aufrufen!
    PGTeamView* view = (PGTeamView*) self.view;
    [view.playersView setEditing:editing animated:animated];
}
```

- In der Delegate-Methode kann relativ leicht die ausgewählte Zeile aus der Ansicht gelöscht werden (der withRowAnimation-Parameter legt die Animation fest)

```
    [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath] withRowAnimation:..];
```

- Führt man die Applikation jetzt aus und testet das Löschverhalten führt dies zu einem Absturz des Programmes: Das Löschen einer Zeile muss sich in den Daten widerspiegeln; ohne Löschen aus der Persistenzschicht geht es also nicht

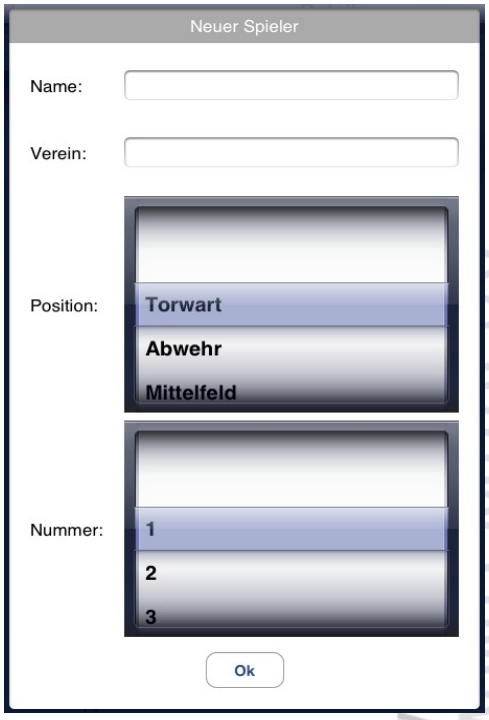
Löschen in der Team-Ansicht (II)

```
if (editingStyle == UITableViewCellEditingStyleDelete) {
    NSManagedObjectContext* ctx = [PGEntitySetup sharedInstance].managedObjectContext;
    [ctx deleteObject: [self fetchPlayerWithIndexPath:indexPath]];                    // (A)
    NSError* error;
    if (![ctx save:&error]) { /*..*/ }                                                // (B)
    else {
        [ctx refreshObject:_team mergeChanges:YES];                                   // (C)
        [tableView deleteRowsAtIndexPaths:  [NSArray arrayWithObject:indexPath]  withRowAnimation:...];   // (D)
    }
}
```

- (A): Zum Löschen aus der Persistenzschicht muss eine Entität zu erst einmal aus dem Kontext gelöscht werden, der sie verwaltet
- (B): Wie schon beim Editieren des Spielplans muss die Änderung am Kontext gespeichert werden; dadurch wird das Löschen an den Persistenz-Koordinator weitergeleitet und die Entität aus der SQLite-Datenbank gelöscht
- (C) Da ein Spieler aus dem Team gelöscht wurde, hat sich auch das Team geändert; diese Änderung muss aktiv „refresht“ werden
- (D) Jetzt kann auch die Zeile aus der Ansicht entfernt werden

Hinzufügen in der Team-Ansicht

- Der Edit-Modus in der Team-Ansicht bietet bisher lediglich nur das Löschen von Datensätzen an
- Zum Einfügen von neuen Datensätze wird im Storyboard ein neue ViewController hinzugefügt und mit einer neuen Controller-Klasse verbunden
- In der Team-Ansicht wird an beliebiger Stelle ein neuer Button einbaut der nur angezeigt werden soll, wenn der Editmodus aktiviert wurde
- Der Button wird zudem im Storyboard per Popover-Segue mit dem neuen Controller verbunden
- Die Einfügen-Ansicht soll über zwei Texteingaben für den Namen und den Verein verfügen, sowie über zwei Picker für die Position und die Rücknummer
- Das Edit-Popover wird durch einen Ok-Button geschlossen, der per IBAction mit dem Controller verbunden ist



Hinzufügen in der Team-Ansicht (II)

- Die Delegats für die UIPickerViews lassen sich wieder relativ trivial im Controller implementieren; einzig für die Position muss ein Mapping von Reihe auf Positions-String festgelegt werden
- Damit das Popover bei einem Klick auf Ok geschlossen werden kann, muss der neue Controller den Popover-Controller kennen
- Eine gute Gelegenheit die beiden miteinander bekannt zu machen ist, die prepareForSegue:sender Methode im PGTeamViewController, die aufgerufen wird, sobald auf den neuen Einfügen-Button geklickt wird
- Um nach dem Einfügen die Team-Ansicht aktualisieren zu können wird bei der Gelegenheit auch gleich der PGTeamViewController mit dem neuen Controller befreundet:

```
-(void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    PGNewPlayerController* newController = (PGNewPlayerController*) [segue destinationViewController];

    newController.teamController = self;
    newController.popController = ((UIStoryboardPopoverSegue*)segue).popoverController;
}
```

Hinzufügen in der Team-Ansicht (III)

- Jetzt fehlt nur noch das Einfügen eines neuen Datensatzes

```
NSInteger selected = [_positionPicker selectedRowInComponent:0];
NSString* position = [self pickerView:_positionPicker titleForRow:selected forComponent:0];
NSInteger number = [_numberPicker selectedRowInComponent:0] + 1;
NSMutableDictionary* dict = [[NSMutableDictionary alloc] initWithCapacity:5];
[dict setValue:_nameText.text forKey:@"name"];
[dict setValue:position forKey:@"position"];
[dict setValue:_clubText.text forKey:@"club"];
[dict setValue:[NSNumber numberWithInt:number] forKey:@"number"];

Player* player = [[PGEntitySetup sharedInstance] createPlayerFromDictionary:dict];
player.team = _teamController.team;

NSManagedObjectContext* ctx = [PGEntitySetup sharedInstance].managedObjectContext;
NSError* error;
if ([ctx save:&error]) {  ...  }

[ctx refreshObject:_teamController.team mergeChanges:YES];

PGTeamView* view = (PGTeamView*) _teamController.view;
[view.playersView reloadData];
[_popController dismissPopoverAnimated:YES];
```