Home Assignment #4

Due date: 27.07.2022, 23:59

Login

- after successful login inject `UserResponse` and `AuthInfo` to `HomeViewController` via simple property, before you navigate to home view.
- For `AuthInfo` please check Networking demo from lecture #4 - there you'll find how to parse it and store it. Backend will store it to response headers so it's a bit different way for parsing it.
- on login/register failure show `UIAlertController` with `.alert` style, error message and `Ok` action
  - UIAlertController
  - https://developer.apple.com/documentation/uikit/uialertcontroller
  - Google for more examples

Shows list

- on `HomeViewController` make API request for `GET /shows` and parse them to `ShowsResponse` using `Decodable`
  - **you need to set `AuthInfo` from login response to all requests after login or your API call will fail**
    - Token authorization is industry standard - you don't want to send username/password all the time in request
    - You log in once, and API generates some tokens to identify you - which expires after some time (if new request is not created in some time) - if you ever get message `Session expired` from Facebook or other services, it means that you did not make any request in defined time and server deleted your token :)

```
AF
  .request(
      "https://tv-shows.infinum.academy/shows",
      method: .get,
      parameters: ["page": "1", "items": "100"], // pagination arguments
      headers: HTTPHeaders(authInfo.headers)
  )
  .validate()
  .responseDecodable(of: ShowsResponse.self) { ... }
```

  - check API documentation for models :)

```
struct Show: Decodable {
    let id: String
    let title: String
    ... // all other properties
}

struct ShowsResponse: Decodable {
    let shows: [Show]
    ... // pagination metadata (optional, only needed for extra)
}
```

- show/hide spinner, everything you think is good to do :)
- add `UITableView` to `HomeViewController` (pin to 0-0-0-0)
- create custom `UITableViewCell` with title label for show name -- just title for now
    - remember using `tableView.reloadData()` when you set TV shows ;)
    - `reloadData()` tells table view to again invoke all those data source methods - how many items, cell for given index path...
- Set title of `HomeViewController` to `Shows`

Remove back button

- remove back button from `HomeViewController` (clear current navigation stack - nowhere to navigate back ;)
- hint:

```
navigationController?.setViewControllers([homeViewController], animated: true)
```

Extra home assignment (not mandatory):

If you take a closer look at a response which our backend serves us, you'll see that in Shows response we got some kind of metadata. There you'll see some information about current page `page` and number of items per page `items`. When communicating with backend and fetching some lists it is very inefficient to immediately fetch the whole list because user won't see it complete and response will take longer time. You want to optimize response time.

Our Shows list endpoint supports pagination - it's technique where you can specify which page you want to access and how many elements per page you want to fetch. So in our example above with `parameters: ["page": "1", "items": "100"]` we told our backend: please return me the first page of the shows list with 100 elements in it. Ideally that number of items shouldn't go over 20-30 in order to preserve time and data consumption.

So basic implementation would be to implement normal UITableView, like you did in home assignment above but with few modifications;

- You won't fetch all 100 shows, but some smaller number, around 20
- You'll have to implement a logic which will fetch next page, aka current_page + 1, when it makes sense
- It makes sense to fetch a new page only when user scrolls down to bottom

- For above, you could use delegate method called `willDisplayCell` at index path, and compare index path row with currently fetched shows count
- When you fetch the new page, you'll want to append it to current array of Shows ;)
- You'll have to somehow store what was the last page you fetched
- Don't forget to call `tableView.reloadData()`

Few things to consider:

- You can find same technique under few similar names: table view pagination, infinite scrolling...
- Ideally you will want to load/fetch next page only when user sees the last cell
- what if user tries to scroll down multiple times, won't that start multiple API calls? Take a closer look and what `responseDecodable(of:)` ... returns as return argument, probably will help you solving this edge case.