CS Capstone Progress Report

May 6, 2018

eBay iOS eSports Application

CS463 Senior Software Engineering Project III

Spring 2018

Prepared for

eBay

Luther Boorn _____ _____
Signature *Date*

Prepared by

Group 17

Skill Capped IRL

Katherine Bajno _____ _____
Signature *Date*

Meagan Olsen _____ _____
Signature *Date*

William Sims _____ _____
Signature *Date*

Kiarash Teymoury _____ _____
Signature *Date*

Abstract

This document outlines the midterm progress made on the eBay iOS eSports application during Spring term. Included is a discussion of the project purpose and goals, current state, remaining work, problems encountered, interesting code, and user study results. We have also included updated images to show the current state of the application.

CONTENTS

1 Purpose

eBay would like to explore the eSports market and target millennial gamers with their products.They also have some public APIs that they have recently released and would like to test those APIs.Currently, there are no convenient, one stop shop applications that provide easy access to eSports merchandise for many different games.

2 Goals

Our goal with this project is to to provide a mutually beneficial application that will allow gamers access to consolidated information and merchandise for their favorite games and expose eBay to the millennial gamers which make up the eSports market. Specifically, this application includes a social component which displays tweets regarding upcoming events for a restricted number of eSports games and provides useful information including dates and times of upcoming eSports events. It utilizes eBay's recently released public APIs, and it provides users access to eSports merchandise related to specific games.

3 Current State

Currently, all of the high level features of our project are complete. Users can view merchandise for both events and games. Merchandise can be filtered and sorted from the both the browse event and browse game pages. The featured event is displayed on the home screen along with the associated twitter account. See more tweets correctly navigates to the twitter timeline of the targeted account and the featured event carousel is correctly populated with relevant merchandise. Users are able to sign up for an account with Firebase Authentication and favorite games from browse. Games that are marked as a favorite are correctly displayed on the home screen. Only minor implementation details and bugs remain which are described in the next section.

4 Remaining Work

4.1 Favorites button on browse screens

When a user is browsing merchandise, there is a blue heart for favoriting games that appears on the screen. Currently this heart does nothing. We need to take the code from the favoriting on the browse screen and use it for the browsing game merchandise screen to allow a user to favorite and unfavorite games.

4.2 Individual events

Currently the merchandise screens for individual events are not yet complete. If a user tries to select an event from browse screen, they will not get the proper event. The UI is complete so all that needs to be done is a call needs to be made for the proper event when the event is selected in the application.

4.3 Firebase database pulls information

Currently the information is being stored on Firebase Storage and is statically pulled from the application itself. The information instead needs to be stored on the Firebase Database. This would allow for the games and event information to change on the application whenever the information changes on the database rather than editing the code itself.

4.4 Favorites on home being alpabetical

Currently the favorites on the home screen are not alphabetical. This is due to the order of the static information for the events and games. This makes it confusing for the user. The games will be alphabetical when they are being pulled from the Firebase Database since it automatically alphabetizes the information stored.

4.5 No Internet notification

Our application relies on the user being connected to the Internet. Currently if the user is not connected to the Internet they receive no notification. A notification needs to be implemented.

4.6 Filter padding

On the merchandise filter some of the UI elements are off-center based on the screen size. This needs to be fixed so the UI is consistent on all screen sizes.

4.7 No favorite from browse event

Currently there are hearts on the browse event screens that are meant for the user to favorite them. Since we are only allowing for the favoriting of games, the heart needs to be removed on the browse event screen.

5 Problems and Solutions

5.1 Firebase Database

One problem faced when storing the information in the Firebase Database rather than Firebase Storage was the order that tasks happen in the application. The view loads before pulling the information from the Internet. This caused Katherine to run into some issues as the previous code that was written assumed the information would be there since it was static. The solution to this was changing the way that views load by having a base case that if there were no elements in the array yet, only one view would appear. This allows the program to initially load a default view and then load the view with the information from Firebase, which is all the user would see.

5.2 Twitter JSON

There were a few issues parsing the JSON for the most recent tweet because Will didnt realize that the JSON is still stored in a dictionary with one element when grabbing a single tweet. He kept trying to parse the JSON as a single string and the errors thrown by Xcode werent very helpful. He eventually realized that he needed to grab a single element of the dictionary it fixed the issue.

5.3 Dynamic tweet sizing

Will also ran into a few issues dynamically sizing the twitter cell and Kia helped him figure out a solution. The solution involved loading the tweet and calling the sizeThatFits() method on the tweet view prior to creating the twitter cell. We were unable to access the size of the tweet outside of the loadTweet function so we needed to call that function from within the home view controller which is where the cell is created.

5.4 Token Expiration

Kia ran into some issues retrieving an access token for the eBay API. The token expires every two hours and initially we would manually update the token string in order to authenticate the eBay API for displaying merchandise. After looking more into the token request call and eBay Auth, Kia was able to successfully implement retrieve token and use it within the application.

5.5 Merchandise Image Quality

During early iterations of the application we had some issues with the image quality of merchandise that was being returned by the eBay browse API. The reason for the blurry images was we were using the thumbnail image URL instead of the primary image URL from the item object. The problem was solved when Kia discovered the correct JSON parameter to use for the image URLs.

6 Interesting Pieces of Code

Checking if a user has favorited an item or not to change heart color on Browse.

```
let user = Auth.auth().currentUser?.uid
           //if user is signed in
           if user != nil{
               Database.database().reference().child("users").child(user!).child("favorites").observeSi
                   if let items = snapshot.value as? [String:Bool]{
                       if let check = items[self.curGame!.id!]
                       {
                           if check == true {
                               self.heartView.tintColor = UIColor.lightBlue
                           }
                           else if check == false{
                               self.heartView.tintColor = UIColor.softGray
                           }

                       }
                   }
               }, withCancel: nil)
```

Checking if a user is signed in to determine how to populate favorites on Home.

```
    if Auth.auth().currentUser != nil {
               if let f = userFavorites{
               if f.count > 0 {
                   print("user signed in with favorites")
                   self.errorText.isHidden = true
                   self.gameImage.isHidden = false
                   self.textLabel.isHidden = false
                   self.heartView.isHidden = false
                   self.carouselCollectionView.isHidden = false
                   self.merchButton.isHidden = false
```

```
                }
                else{
                    print("user signed in with no favorites")
                    self.gameImage.isHidden = true
                    self.textLabel.isHidden = true
                    self.heartView.isHidden = true
                    self.carouselCollectionView.isHidden = true
                    self.merchButton.isHidden = true
                    self.errorText.isHidden = false
                    self.errorText.text = "Go to browse to start favoriting games."

                }

                }} else {
                print("no user signed in")
                self.gameImage.isHidden = true
                self.textLabel.isHidden = true
                self.heartView.isHidden = true
                self.carouselCollectionView.isHidden = true
                self.merchButton.isHidden = true
                self.errorText.text = "Sign in to view your favorites."
            }
```

Loading the most tweet to be displayed.

```
func getTweet(completion: @escaping (TWTRTweet) -> ()) {
    let client = TWTRAPIClient()
    //Construct URL to query the most recent tweet from the featured event twitter account
    let statusesShowEndpoint = "https://api.twitter.com/1.1/statuses/user_timeline.json"
    let params = ["screen_name": "E3", "count": "1"]
    var clientError : NSError?

    let request = client.urlRequest(withMethod: "GET", urlString: statusesShowEndpoint, parameters: para

    client.sendTwitterRequest(request) { (response, data, connectionError) -> Void in
        if connectionError != nil { print("Error: with Connection") }

        do {
            guard let data = data else { return }

            guard
                let json = try JSONSerialization.jsonObject(with: data, options: .mutableContainers) as
                let dict = json[0] as? NSDictionary,
                let tweetID = dict["id_str"] as? String else { return }

            // Load the tweet in the Tweet view based on the ID retrieved
```

```
            client.loadTweet(withID: tweetID) { (tweet, error) in
                if let t = tweet {
                    DispatchQueue.main.async {
                        completion(t)
                    }
                }
            }

        } catch let jsonError as NSError {
            print("json error: \(jsonError.localizedDescription)")
        }
    }
}
```

Twitter Timeline View Controller

```
class Twitter_Timeline: TWTRTimelineViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        title = "Overwatch League"
        self.dataSource = TWTRUserTimelineDataSource(screenName:
"overwatchleague", apiClient: TWTRAPIClient())
    }
}
```

Networkinh Protocol Model

```
enum RequestRespnse: Error {
    case success(String)
    case error(String)
}

protocol Networking {
    associatedtype Model: Codable
    func requestData(forUrl url: URL, completion: @escaping (RequestRespnse, Model?) -> ())
    func decode(data: Data) -> Model?
}

extension Networking where Self == Model {
        func requestData(forUrl url: URL, completion: @escaping (RequestRespnse, Model?) -> ()) {

        //QUICK FIX FOR NOW
        Token().getToken {
            let authString = "Bearer \($0)"
            let config = URLSessionConfiguration.default
                config.httpAdditionalHeaders = ["Authorization" : authString]
            let urlsession = URLSession(configuration: config)
```

```
            DispatchQueue.global(qos: .userInteractive).async {
                let task = urlsession.dataTask(with: url) {
                    (dataObj, respnse, error) in
                    guard       error == nil,
                        let data = dataObj else { return completion(.error(error?.localizedDescription
}
                    DispatchQueue.main.async {
                        completion(.success("Successfully Requested Data"), self.decode(data: data))
                    }
                }
                task.resume()
            }
        }
    }

    func decode(data: Data) -> Model? {
        do {
            return try JSONDecoder().decode(Model.self, from: data)
        } catch let error {
            print("Error Decoding: \(error.localizedDescription)")
            return nil
        }
    }
}
```

BrowseAPI Protocol Model

```
protocol BrowseAPI: Networking where Model == Root {
    var keyWord: String? { get set }
    var filterBy: Filters.option? { get set }
    var sortBy: Sort.option? { get set }
    var fetchLimit: Int? { get set }
    var range: String? { get set }
    func retrieveDataByName(offset: Int,_ loadingIndicator: UIActivityIndicatorView, _ completion: @esca
}

extension BrowseAPI {

    private var endPoint: String {
        let baseUrl = "https://api.ebay.com/buy/browse/v1/item_summary/search?",
            query = "q=\(keyWord ?? "")&",
            groupBy = "category_ids=\(filterBy?.rawValue ?? "")&",
            limit = "limit=\(fetchLimit ?? 100)&",
            buyOption = "buyingOptions%3A%7BFIXED_PRICE%7D",
            condition = "conditions%3A%7BNEW%7D&",
            priceCurrency = "priceCurrency%3AUSD%2C",
            priceRange = "price%3A%5B\(range ?? "0..")%5D",
```

```
            filter = "filter=\(priceCurrency)\(priceRange),\(buyOption),\(condition)",
            sort = sortBy?.rawValue ?? ""
        return baseUrl + query + groupBy + filter + sort + limit
    }

    func retrieveDataByName(offset: Int,_ loadingIndicator: UIActivityIndicatorView, _ completion: @esca
        guard let url = URL(string: endPoint+"&offset=\(offset)") else { return }
        loadingIndicator.startAnimating()
        print("URL: \(url)")

        var merchendise:[Root]? = [Root]()
        requestData(forUrl: url) { (_response, _merchendise) in
            switch _response {
            case let .error(errorDesctiption):
                print(errorDesctiption)
                completion(nil)
            case let .success(successDesctiption):
                guard   let merchObj = _merchendise else { return }

                merchendise?.append(merchObj)
                DispatchQueue.main.async {
                    completion(merchendise)
                    loadingIndicator.stopAnimating()
                    loadingIndicator.removeFromSuperview()
                    print(successDesctiption)
                }
            }
        }
    }
}
```

Filtering Protocol Model

```
protocol FilterOptions {
    var sectionTitle: String { get }
    var options: [EnumTitles] { get }
}

protocol EnumTitles {
    var name: String { get }
}

extension FilterOptions {
    var sectionTitle: String {
        return String(describing: Self.self)
    }
}
```

```
extension EnumTitles {
    var name: String {
        return String(describing: self).replacingOccurrences(of: "_", with: " ")
    }
}

struct Sort: FilterOptions {
    var options: [EnumTitles] { return [option.Best_Match, option.Lowest_Price, option.Highest_Price] }

    enum option: String, Codable, EnumTitles {
        case Best_Match = "fieldgroups=MATCHING_ITEMS&"
        case Lowest_Price = "sort=price&"
        case Highest_Price = "sort=-price&"
    }
}

struct Filters: FilterOptions {
    var options: [EnumTitles] { return [option.All_Items, option.Toys, option.Clothing] }

    enum option: String, Codable, EnumTitles {
        case All_Items = "1249,1059,220,15687,155183,63859,155206" //Viedeo Games | Toys | Cothing
        case Toys = "220" //Toys and Hobbies
        case Clothing = "15687,155183,63859,155206" //Mens T-Shirts + Sweaters | Woments Clothing
    }
}

struct Price: FilterOptions {

    var options: [EnumTitles] { return [prices.range(NSRange(location: 0, length: 100))] }

    enum prices: EnumTitles {
        case range(NSRange)
    }
}
```

7 User Study Results

In general, all participants found this application moderately easy to use. A little under 50% of the participants found this application easy to use. All the participants claimed to have been involved in gaming. The test took between 15-30 minutes to complete.

7.1 Methodology

During each test session, the test administrator provided a quick briefing of the application and test and asked the participant to complete the tasks while the test administrator observed, recorded feedback, and answered questions.

After each task, the administrator asked the participant to rate the ease of use on a 10 point scale with 1 being super easy to use and 10 being difficult to use.The following scale was used with 1-3 being extremely easy, 3-6 moderately easy, 6-8 moderately difficult,and 8-10-extremely difficult. The tasks included:

- Creating an account
- Finding info about an event
- Finding tweets related to an event
- Browsing merchandise for a game and event
- Viewing the description of an item
- Favoriting and unfavoriting a game
- Signing out

After all tasks were completed, participants rated the app for four overall measures:

- Ease of use
- Overall Impression
- Feelings about interactions
- Look and feel
- Likeliness of reuse

In addition, the test administrator asked the following overall questions:

- What did you like most about the app?
- What did you like least about the app?
- What is one thing you would change?

The participants were then asked to give any other comments they might have about the app.

7.2 Participants

Houses

Courtyard	Antioch
3	2

7.3 Results

All participants completed tasks 1-8.

Task Completion Rates

0-success
X-failure

Participant	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
Success	5	5	5	5	5	5	5	5
Completion Rates	100%	100%	100%	100%	100%	100%	100%	100%

7.3.1 Task Ratings

Creating an account

40% of the participants found it moderately to extremely difficult to create an account while 60% of the participants found it extremely easy to create an account. *Finding info about an event*

60% of the participants found it extremely difficult to find the date and location of the gamescon event while 40% found it extremely easy. *Finding tweets related to an event*

40% of the participants found it moderately easy to find tweets related to the E3 game while 60% of the participants found it extremely easy to find the tweets related to the E3 event. *Browsing Merchandise for a game and event*

100% of the participants found it extremely easy to browse merchandise for the Overwatch game and PAX event. *Viewing the description of an item*

100% of the participants found it extremely easy to find view the detailed description of the PAX shirt. *Favoriting and unfavoriting a game*

60% of the participants found it extremely easy to favorite and unfavorite the League of Legends game while 20% found it moderately easy. *Signing out*

100% of the participants found it extremely easy to sign out of the app.

Mean of task ratings

Task	Mean
1-creating an account	2
2-find info about event	5
3-find tweets about an event	3
4-browse merch related to a game and event	1.5
5-view detailed description of item	1
6-favorite and unfavorite and event	2.25
7-sign out	1

Summary of Completion

Task	Completion
1	5
2	5
3	5
4	5
5	5
6	5
7	5
8	5

7.3.2 Overall Metrics

All the participants agreed that this app was easy to use. All had an extremely good overall impression of the app. All thought the app had extremely good interactions. All had an extremely good impression for the looks and feel of the app. Most(60%) said that they would reuse the app.

Liked Most

- Easy to use and info is presented in an easy to understand way.
- The games were easy to find and the merch was readily accessible.
- Clean UI, with surface level linking and making sense.
- The general feel of the app.

Liked Least

- The Reset button in the filter, useless hearts that cant be unclicked, weird sorting on home page for games.
- I wish browse was more than just shopping.
- Doesnt have a lot of games, but it has my games.
- You kind of have to look hard for some of the info you need.

Recommendations for improvement

- Change up the homepage to be more readable-way too much info. If E3 is the only available event, take the top section in the home page out. Make the games title on the home page selectable.
- I would have the link under the browse pull up a more general info tab about game/event and upcoming events or competitions.
- When logging in, press tab to go to the next field.
- Add a search bar.

Recommendations

Change	Justification	Severity
Make passwords match on registration page	Don't know which password it used to create the account	high
Make forgot button work	Can't reset password	high
Make error messages on sign in page more specific	Don't know how they screwed up	high
State requirements for password creation on register page	Don't know what password to choose	high
Put confirm and reset button at bottom of filter page instead of just reset	Accidently erase filter options	low
Delete the featured event picture on home page	Waste of space, more confusing, too much info	low
Remove the clickable heart from each event page	Not needed	high
Add profile page if using Facebook API	Would be nice, not necessary	low
Add to favorites alphabetically on home page	Confusing	high
Fix see more button on home page		high
Fix logo on browse page for specific events		high
Add confirmation on both home and browse page for favoriting and unfavoriting events		high
Fix merch on home page		high

7.4 Conclusion

Overall, all participants were happy with the overall look and feel of the app. There were only some slight reservations about usability, many having to do with some features being broken. Quickness of finding information is of key importance to users. Continuing to test with different user will ensure maximum usability.

8 IMAGES

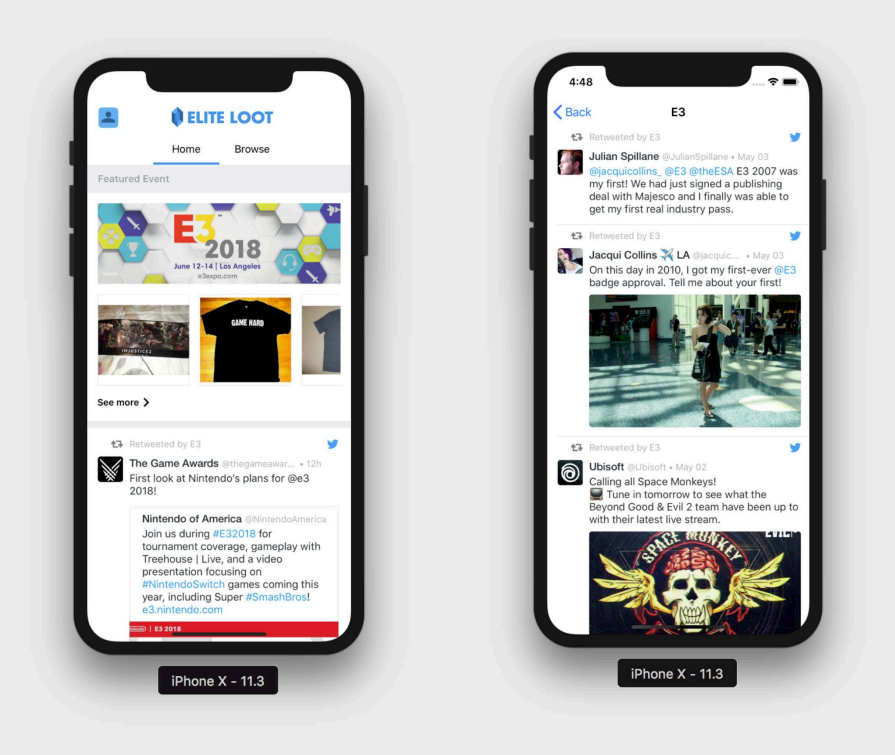

Fig. 1: Home screen and Twitter timeline.

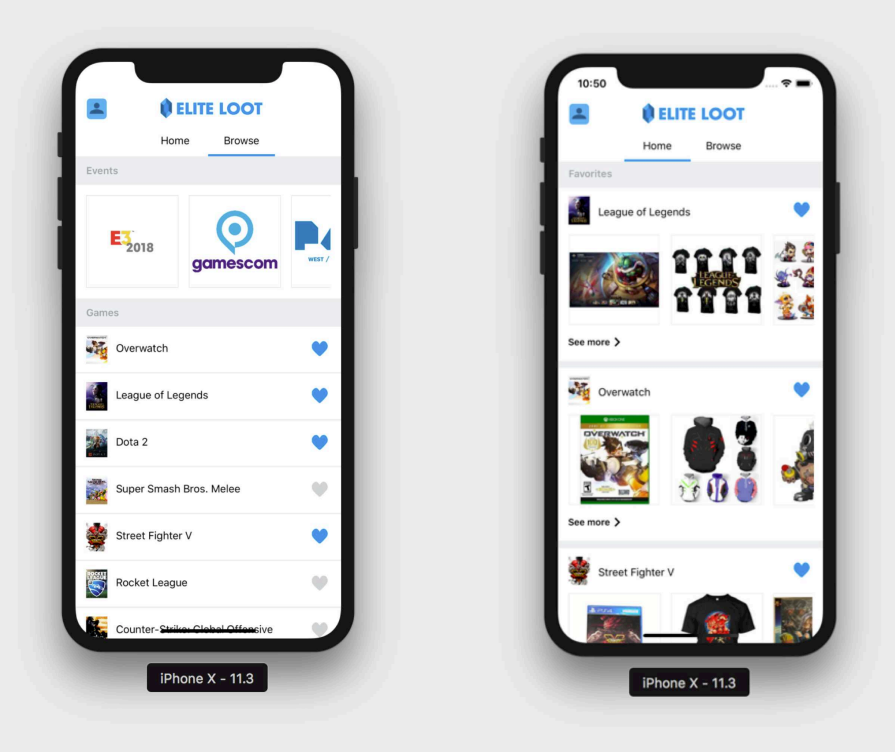

Fig. 2: Browse and favorites from the home screen.

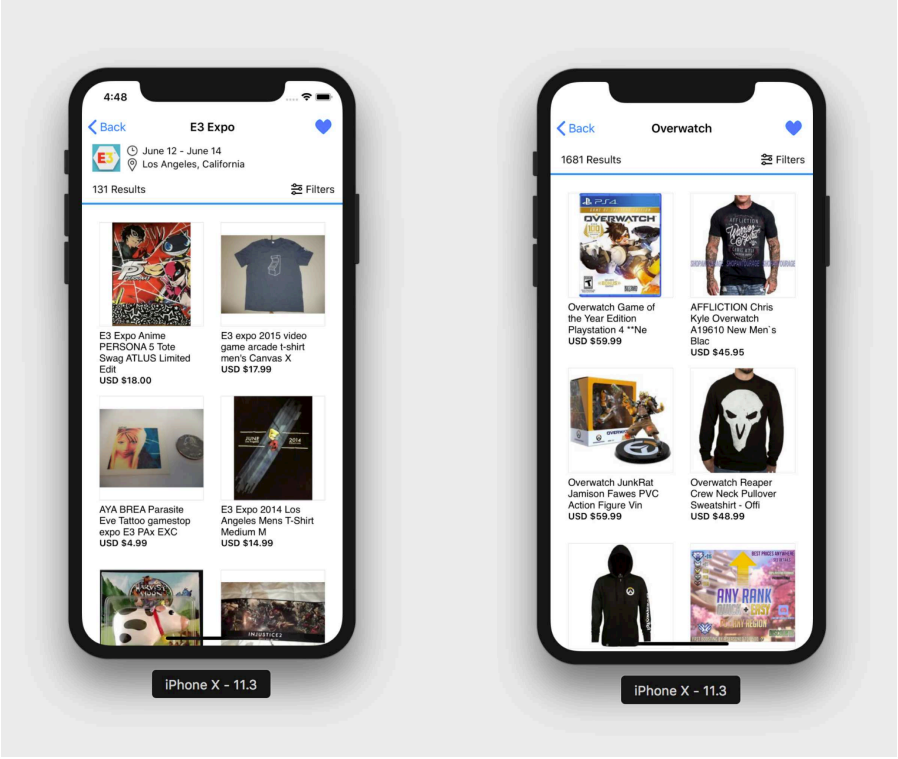

Fig. 3: Browse event and game pages.

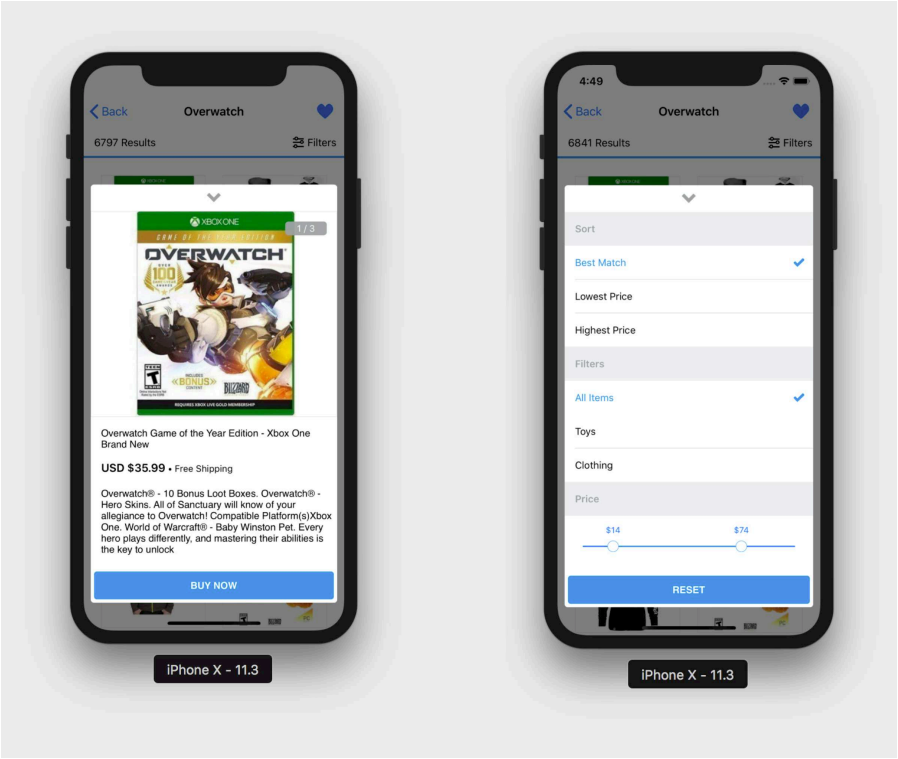

Fig. 4: Product details and filter pages.

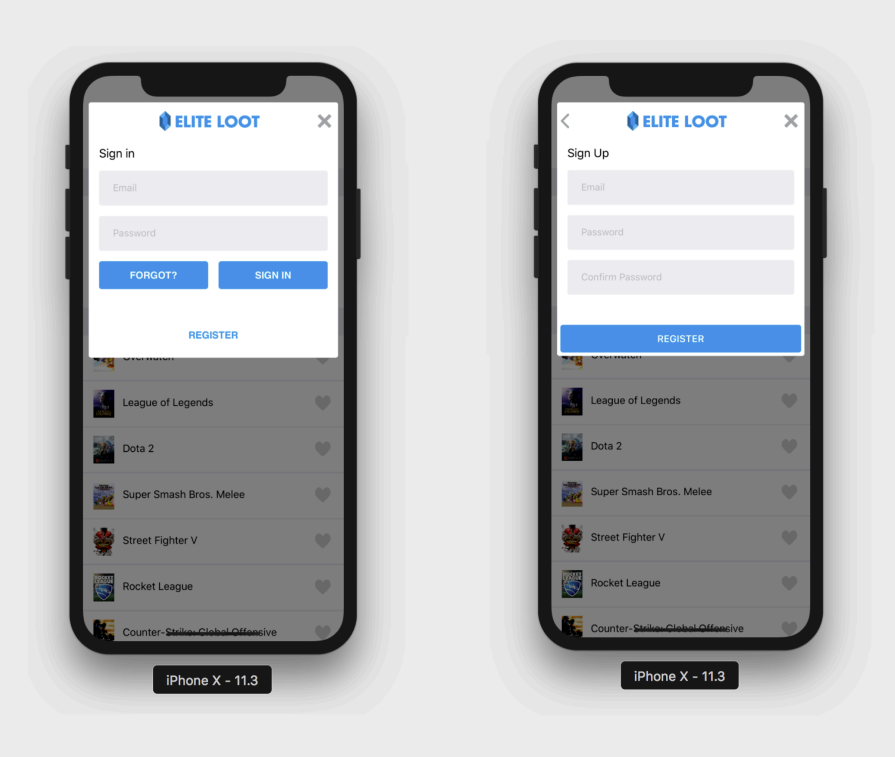

Fig. 5: Sign in and register pages.