Bay Area Chess
Quality You Can Check On!

Bay Area Chess Technical Documentation

About

The Bay Area Chess iOS App is designed as a client server model. The client is the iOS application itself, written in Swift with XCode, Apple's development platform. The API server is a Node.js based web application that serves data from the MySQL database which is shared with the the Bay Area Chess website.

iOS Client Architecture

The client app, written in Swift, consists of several sections:

- The Storyboards
    - Storyboards are Apple's way of designing the views for the client application. They are designed graphically, meaning that as the App developer you are given tools to drag and drop graphical components as you see fit. For example text-boxes, labels, and pictures can be added simply by dragging in components from a toolbar in the Storyboard view. These elements can be bound using identifiers to items in the code base.
- The View Controllers
    - View Controllers are essentially the classes associated with each of the views designed in the Storyboard. They define, via listener methods, the behavior of the application when a user interacts or the server interacts with the App in some fashion.
- Utility Classes
    - The utility classes are collections of methods which statically support common functionality across the application and are often accessed for 'utility' for a wide variety of purposes in a general, supportive way.
- External Libraries
    - Several libraries and SDKs are linked into the application and thus are part of the source. One of the more notable examples of this is the PayPal SDK, which is written in Objective-C and linked in with a bridging header. Supporting libraries in this project reside in their own groupings, separate from the rest of the codebase.

Here is a rundown of all of the core view controllers, support libraries, and supporting modules:

- View Controllers
    - User
        - Login, handles logic associated with logging in.
            - Login route provides a POST endpoint to the client.
            - The server validates the user data and sends back a response as to whether it was successful in validating this request.
        - UserRegistration, handles registering new users.
            - Register route provides a PUT endpoint to the client
            - Client makes PUT request and sends registration data provided by the user to the client. A new record is added to the MySQL database.
        - UserUpdate, updates user information from their profile page.
            - User update provides a POST endpoint through which users can update their information.
            - Server stores updates a MySQL record upon a valid update.
        - User, loads user information into profile view.
            - GET endpoint provides access to user information, this is used to pull in information from the database about users on a valid login.
    - About
        - All about data is statically put into the application as it does not reside anywhere in the database.
        - About, handles main about page.
        - AboutLeft, handles left about page.
        - AboutRight, handles right about page.
    - Tournaments
        - Tournaments, pulls all tournament data into table view.
            - The server provides a GET endpoint for the client to request Tournament data from the database, this retrieves the tournament data records and feeds them to a Table View.
        - SpecificTournament, handles displaying data about a specific tournament.
            - Another GET endpoint is provided here which allows the client to obtain specific information from the MySQL database about a specific tournament, found by tournament id.
- Support Libraries
    - Utils, adds core JSON parsing functionality.
    - Constants, all constants are added to this class.
- External Libraries
    - PayPal SDK, adds support for PayPal payments in-App.

- HTTP Library, adds an abstraction layer for easier use of Apple's REST libraries.

Node.js API Server Architecture

The API server supports the client application by providing REST endpoints to access specific data and process specific request needed by the client. This data includes, but is not limited to, login verification and obtaining tournament data. The base route for all GET requests is structured as follows: *www.bayareachess.com/api/v1/*, thus a valid URL would look similar to this: *www.bayareachess.com/api/v1/all/*

Here are the current sections of the API:

- Login: */api/v1/login/*
- Tournaments: */api/v1/tournaments/*
- Register: */api/v1/register/*

If more endpoints are added they should all be added in a similar fashion to the pre-existing ones.

Another important note to make about the server is the structure of the directories and files. Please note that all of the routes are defined in route files under the 'routes' directory. The logic for each of these routes is defined in the controller file associated with it which can be found in the 'controllers' directory (located at the same level as the routes directory). Finally all of the SQL queries associated with a controller can be found within the query file associated with that controller, found in the queries directory.

Adding Features

If you are interested in adding new features you must first understand how all of the classes in the Client work with the server side of this application. Essentially all of the data pulled into the app or created from the app comes from or goes into the MySQL database that the server is hooked into. Thus adding a new feature is a sequence of four events:

1. Determine if you want to add a new page to the app, or if you want to add/change a pre-existing feature.
    a. If you are adding a new page to the app, you must first make a new Storyboard View Controller and tie this to a View Controller.
    b. If you are not, determine what feature you'd like to change in the app and determine what view controller it's tied to. Modify that view controller and Storyboard view as necessary.

2. If Step 1 part a or b require storing data, retrieving pre-existing data, or doing computationally intensive work, you'll need to have the server do this work.
    a. Look through the server routes above and add a new route, with the appropriate HTTP verb to the routes in the server.
        i. The routes directory is located at: api_server/routes
    b. Add the controller logic that accompanies this route. That is, the actual functionality that reads/writes to the database and how it does this.
        i. The controllers directory is located at: api_server/routes
    c. Add any SQL queries to a queries file in: api_server/routes
3. Once step 2 is finished you will need to create a method in the client to access the information processed by the server, this will be accomplished with an HTTP request.
4. Finally tie any necessary data from the client View Controllers to the Storyboard associated with it.