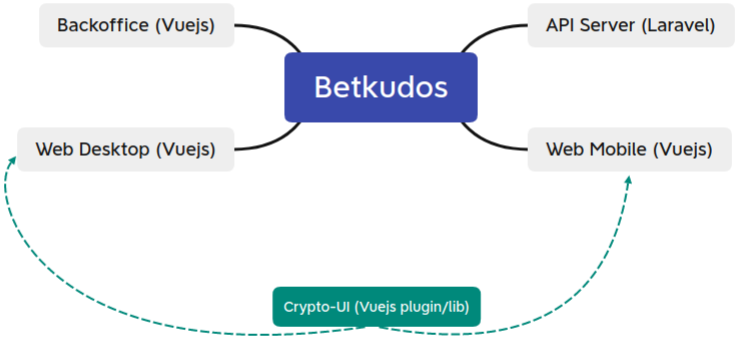
# BetKudos

This document covers all the BetKudos platform, from the client side to the API. The platform was built essentially on top Laravel Framework and Vuejs on the client side. The platform needs some third party integrations and at time of writing it also works with Bitcoin and Ethereum Integration.

## Overview

This is a very basic overview of the platform architecture, we will go deeper on each project to give you more details about each one and how it works.




## API Server

The API server is built on top of Laravel Framework which also uses some Laravel ecosystem tools like Octane + Swoole for Http Server, Horizon for background jobs, Passport of authentication and many more.

This is the main application structure.

```
├── app
│   ├── BetsAPI (part of the BetsAPI implementation required)
│   ├── Blockchain (Most of the blockchain logic is here, because it's easier to share with blockchain dev/team)
│   ├── Concerns (Any external concern not related with businnes logic)
│   ├── Console (Application console commands)
│   ├── Contracts (Interfaces that will be implement by the other layers)
│   ├── Events (Application events)
│   ├── Exceptions (Application custom exceptions)
│   ├── Http (Http layer, Controllers, etc)
│   ├── Jobs (Background jobs)
│   ├── Listeners (Application listeners)
│   ├── Models (Application models)
│   ├── Notifications (Application notifications ex.: emails)
│   ├── Processors (Sportsbook markets processors - will be used to set the outcome of a bet)
│   ├── Providers (Application providers)
│   ├── Repositories (Application repositories)
│   ├── Rules (Request rules)
│   ├── Services (Most of the business logic live here)
│   ├── Slotegrator (Slotegrator implementation)
│   ├── Support
│   ├── ValueObjects
│   ├── helpers.php
├── ... Laravel Stuff
```

These are some of the most important layers on API Server:

- BetsAPI
  - BetsAPI Http client
  - Inplay data compiler (Inplay market data is "obfuscated" so it needs to be transformed before inserted on databse)
- Blockchain
  - Bitcoin and Ethereum nodes implementation
  - CryptoWallet lib that's wrapper for BTC/ETH nodes
  - Jobs folder with Blockchain deposit/withdraw jobs
- Console
  - Kernel.php (All Scheduled jobs are defined here)
  - SB (Folder with sportsbook related commands)
  - SlotegratorGames command (Store slotegrator games into the database)
  - UserRoles command (add a role for a given user)
- Http (Http layer)
- Jobs (Most of the jobs are here)
- Models (All the application models)
- Processors (All the classes responsible for processing bets with bet365 results)
  - Processor.php (Entrypoint for processing a bet outcome)
- Services (Business logic)
- Slotegrator (Slotegrator implementation)

## Installing

To get the project running you will need to install the dependencies and also run the database seeders.

This project was built on top of Laravel Framework and uses Laravel Sail for development environment.

#### Project dependencies

- PHP 8.0 or higher
- composer 2.0 or higher
- Docker
- docker-compose

- Nodejs + yarn

Firstly you will need to install the composer dependencies by running the command bellow:

```
composer install
```

Then we will install NPM dependencies, which will allow Laravel Octane to reload the server workers on file changes by using Chokidar. To do so we will use yarn to install the dependencies.

```
yarn
```

Now you will need to create a **.env** file with all the required environment vars. You can just make copy from **.env.example** and fill the new **.env** file with the right values.

**Database vars on .env.example are already ready for using Laravel Sail, so you will don't need to change this**

To make a copy you can just run:

```
cp .env.example .env
```

Once all dependencies have been installed and .env filled with the right values we will generate our application key required by Laravel by running:

```
php artisan key:generate
```

Now we will run Laravel Sail to build the containers and get them running. To get more info about Laravel Sail check the docs

First, make the sail file executable

```
sudo chmod +x ./sail
```

and then run the containers

```
./sail up
```

Or to run the containers in detached mode use this instead:

```
./sail up -d
```

if you want to stop the containers (in detached mode) run the following

```
./sail down
```

### Database Setup

Now all containers should be up and running, then it's time to run the database seeders and insert all the data the project needs to work as expected. **As mentioned above you don't need to define database env vars to use *Sail* if you make a copy from .env.example. So you just need to run the migrations.**

First, you should migrate the database tables:

```
./sail artisan migrate
```

If everything worked as expected you should now run the database seeders. The command bellow will run the /database/seeders/DatabaseSeeder.php file which will insert all the data needed.

```
./sail artisan db:seed
```

Now we have everything we need on our database.

### Laravel Passport

All the authentication flow Login/Register was built on top of Laravel Passport which is a package to build API authentication with OAuth2. To get the authentication working you will need to create the Laravel Passport password client which will be used on our authentication flow from the SPA.

To create the Passport password client you should run the following command:

```
./sail artisan passport:client --password
```

Now you should fill the **PASSPORT_PASSWORD_CLIENT_ID** and **PASSPORT_PASSWORD_CLIENT_SECRET** vars on your .env file with the result of the command above. The ***PASSPORT_PASSWORD_CLIENT_ID*** value should be **1** if you are creating a fresh database.

## Running the project locally

Once you have installed the project dependencies and your containers are running You should start a Laravel Horizon instance and also the Laravel Scheduler. To start a new laravel horizon instance run this:

```
./sail art horizon
```

Keep the above command running. To access the Horizon dashboard open up http://localhost/horizon

### Setting up third-party services:

Now that you have a horizon instance running you should ensure all third-party services are working as expected.

### Third-party services used:

- BetsAPI (Bet365 API)
- Slotegrator (Casino)

#### BetsAPI (Bet365 API)

First you should define the `BETSAPI_TOKEN` environment variable into your .env file, you also have to make sure the API token is currently active.

Then you can run the following command:

```
./sail artisan sb:upcoming-matches 1 --all
```

This command will request soccer events on Bet365 and will dispatch jobs to store them on the database. The first argument `1` means the sport id, in this case `1` is the soccer id.

If everything worked as expected you will see jobs available on Horizon Dashboard. You can also check if any job failed.

#### Slotegrator (Casino)

First you should define these environment variables on your .env

```
SLOTEGRATOR_MERCHANT_ID=
SLOTEGRATOR_MERCHANT_KEY=
```

Casino games images are stored on our Digital Ocean CDN, so you will need to set this

```
DIGITALOCEAN_SPACES_KEY=
DIGITALOCEAN_SPACES_SECRET=
DIGITALOCEAN_SPACES_ENDPOINT=
DIGITALOCEAN_SPACES_REGION=
DIGITALOCEAN_SPACES_BUCKET=
DIGITALOCEAN_CASINO_CDN=
```

Now you will be able to communicate with slotegrator and serve game images.

### Scheduler

A lot of business logic depends on Laravel Scheduler, this includes blockchain tasks that will process deposits and withdrawals, so you will need to start it to see everything working. To do so we can run:

```
./sail artisan schedule:work
```

Keep this command running in addition to Horizon and you will have 100% of the application running.

### User Roles and Permissions

This project uses spatie/laravel-permission package to manage users permissions. This is useful to allow specific users to access the Backoffice or even allow just some permissions to a specific user.

To give a user full access to the BOST, you should run this command:

```
./sail artisan user:roles liam bookie
```

The first argument `liam` is the username of the user you will give a role. You can also give more than a role per time:

```
./sail artisan user:roles liam bookie example-role admin manager finances
```

The application uses a role based approach to restrict users to access a given resource.

These are the current available roles:

```
    'roles' => [
       [
           'name' => 'bookie',

           'permissions' => [
               'faqs' => 'create|edit|delete',
               'promotions' => 'create|edit|delete',
               'users' => 'list|search|filter|restrict',
               'sessions' => 'list|search|filter',
               'settings' => 'edit',
               'not allowed ips' => 'create|delete',
           ]
       ]
    ]
```

You can add more if required and then run the following command to create the new roles on database:

```
./sail artisan db:seed --class=RolesAndPermissionsSeeder
```

## HTTP API

The API

## USER AUTHENTICATION

#### - /auth/register [POST]

Attempts to Register a new user

#### - /auth/login [POST]

Attempts to log in a user

#### - /auth/verify/{id}/{hash} [GET]

Used to verify a user email by validating the hash sent

#### - /auth/forgot-password [POST]

Send a reset password link to the user email.

#### - /auth/reset-password/{method?} [POST]

Reset the user password using email or 2FA method. If 2FA is enabled it will require OTP code and will try to reset password.

#### - /auth/resend [POST]

Re-send a password reset email

#### - /auth/2fa/qr-code [GET]

Generate a new 2FA QrCode

#### - /auth/2fa/enable [POST]

Enable 2FA for the user account

#### - /auth/2fa/disable [POST]

Disable 2FA for the user account

#### - /auth/logout [POST]

Logout the current active user API token

## FEED

#### /feed/popular/{status}/{sport} [GET]

Get popular events for a given time status (live/upcoming) and a given sport

#### /feed/preview/today/{sport} [GET]

Get the feed preview for today's events

**/feed/preview/live/{sport} [GET]**

Get the feed preview for live events

**/feed/today/{sport} [GET]**

Get today's events by sport

**/feed/soon/{sport} [GET]**

Get events that will start in up to 1 hour

**/feed/live/{sport} [GET]**

Get live events by sport

**/feed/live/{sport} [GET]**

Get live events by sport

**/feed/upcoming/{sport} [GET]**

Get upcoming events by sport

**/feed/upcoming/{sport}/categories [GET]**

Get sport categories that has events, it's used for E-Sports and MMA/UFC

**/feed/upcoming/categories/{sportCategories} [GET]**

Get events from a sport category

**/feed/competitions/{country}/{sport} [GET]**

Get competitions from a country where has events for a given sport

**/feed/preview/upcoming [GET]**

Get feed preview for upcoming events

**/feed/preview/live [GET]**

Get feed preview for live events

**/matches/{match} [GET]**

Get details from a match

**/matches/{match}/odds [GET]**

Get the odds from a match

**/sports/ [GET]**

Get all the sports that has upcoming events

**/sports/live [GET]**

Get all the sports that has live events

**/sports/soon [GET]**

Get all the sports that has events that will start soon

**/countries/{sport} [GET]**

Get countries that has upcoming matches from a sport

## WEB CLIENT

**/localaes/{locale} [GET]**

Get the translations for the web client for a given locale

**/faqs/welcome [GET]**

Get welcome FAQs that will be shown when user is registered

**/user/account [POST]**

Get user account details

**/user/update [GET]**

Update user details

**/bets [POST]**

Try to place a bet

**/bets/open [GET]**

Get user open bets

**/bets/settled [GET]**

Get user settled bets

**/bet-slip/update [GET]**

Update bet slip selections

**/wallets/{wallet}/address/generate [GET]**

Generate a new deposit address to a wallet

**/withdrawals [POST]**

Create a new withdrawal request

**/transactions [GET]**

Get user transactions history

## BOST

**Bost routes has its own routes file which is routes/bost.php, however some user related routes are in /routes/api.php**

**The routes bellow are in the routes/api.php file. So they are in the API middleware:**

#### /users [GET]

Get all users

#### /users/search/{value}/{field}/{filter?} [GET]

Search for users based on a filter (username, ID, IP, Etc)

#### /users/filter/{type} [GET]

Filter users by top losers and top winners

#### /users/{id}/status [PUT]

Restrict/Unrestrict a user

#### /users/{user}/block/current-ip [PUT]

Blocks the user current IP Address

**The routes bellow are in the routes/bost.php file:**

#### /bost/login [POST]

Bost authentication

#### /bost/sessions/search/{value}/{field}/{filter?} [GET]

Search user sessions

#### /bost/sessions/filter/{filter?} [GET]

Filter user sessions

#### /bost/sessions/{session}/block-ip [PUT]

Block a user session IP Address

#### /bost/sessions [PUT]

Get all sessions

#### /bost/sessions/{session} [PUT]

Show details from sessions

#### /bost/settings [GET]

Get the platform settings

#### /bost/settings [PUT]

Update the platform settings

#### /bost/settings [GET]

Update the platform settings

#### /bost/settings [GET]

Update the platform settings

#### /bost/bet-monitor [GET]

Get the bet monitor

#### /bost/limits [GET]

Get the platform limits

#### /bost/risk-overview [GET]

Get the risk overview

#### /bost/user-profile/{user} [GET]

Get user profile details

#### /bost/user-profile/transactions/{user} [GET]

Get transactions from a user

#### /bost/user-profile/notes/{user} [POST]

Update user profile notes

## CASINO

#### /casino/games [GET]

List the casino games

#### /casino/games/{game} [GET]

Show a specific game

#### /casino/providers [GET]

Get providers list

#### /casino/init/{game}

Init a game on the game provider and get game URL

## Slotegrator Implementation

The slotegrator implementation currently covers ~60% of logic required.

What's working?

- Slotegrator requests sign
- Aggregator requests validation
- A Job + a Command to store slotegrator games (php artisan slotegrator:games or if using sail ./sail art slotegrator:games)

- Casino WebHook URL /casino/aggregator (needs to be tested) this route is at routes/web.php
- Aggregator user balance requests (needs to be tested)

What's not working?

- Aggregator bet request
- Aggregator win request
- Aggregator refund request
- Aggregator rollback request

To give more details of how to implement the requests above check the Slotegrator API docs.

# crypto-ui

## What's this package?

Crypto-ui was created with the intent of share our business logic although our front-end repositories. As we have the need of a web mobile application, and a web desktop application, rewrite some features, vue components and business logic is not an option.

So we've created this NPM private package currently published at @gustavoedny/crypto-ui to share all our vue components, api services, vuex store and much more.

It also works as a Vuejs plugin, so we can share on any Vuejs 2 app.

## Project setup

### Docker environment

We highly recommend use our Docker + docker-compose environment, you will need Docker and docker-compose installed and then follow these steps:

- Create the docker env file that will be used by docker-compose

```
cp .env.docker.example .env.docker
```

Then update the NPM_TOKEN value in your .env.docker file

- Build the docker image with docker-compose

```
docker-composer build --no-cache
```

### Running containers

- To start the containers:

```
docker-composer up
```

it will start the development server at localhost:8082

To STOP the container just type CTRL+C or CMD+C in your terminal

### Running containers in background

- Running in background (Note that it will take a few seconds to the development server start at localhost:8082)

```
docker-composer up -d
```

- To stop the container in background run:

```
docker-compose down
```

### Local environment

To set up this project you will have to set up the NPM private token firstly, so when you have the token run the command bellow and make sure to replace the "NPM_TOKEN" by the provided token:

```
npm config set //registry.npmjs.org/:_authToken "NPM_TOKEN"
```

Once you have the private token configured you should install the package dependencies:

```
yarn global add @vue/cli @vue/cli-service-global && yarn install
```

Once dependencies have been installed you can start the development server:

```
yarn dev
```

## Folder Structure

```
├── dev
│   ├── App.vue (The root component used on main.js and also the playground, where we can test /src components)
│   ├── main.js (Vue instance entrypoint, setup the plugin and dependencies including Vuetify)
│   ├── store.js (A fresh Vuex store)
│   ├── crypto-ui.js (Installs the crypto-ui plugin on the Vue instance)
│   ├── BetSlipWrapper.vue (BetSlip plug-and-play component to add on App.vue)
│   ├── betslip-state.json (Bet Slip dummy data)
├── dist (or build)
├── node_modules (NPM dependencies)
├── public (Developement server public folder)
├── src
│   ├── bus (Global event bus that can be used to comunicate events between components)
│   ├── components (All the library components, including Bet Slip)
│       ├── Small Component (Eg.: ShowMore)
│           ├── __tests__ (Component test suite)
│               ├── Component.spec.js
│           ├── Component.scss (Component styles)
│           ├── Component.vue (Component styles)
│           ├── index.js (Component entrypoint, will expose Component.vue publicly)
│       ├── Big Component (Eg.: BetSlip)
│           ├── SubComponent
│               ├── __tests__
│                   ├── SubComponent.spec.js
│               ├── SubComponent.vue
│               ├── SubComponent.scss
│               ├── index.js (SubComponent entrypoint)
│           ├── AnotherSubComponent
│           ├── BigComponent.vue
│           ├── BigComponent.scss
│           ├── index.js (BigComponent entrypoint, will make it public)
│   ├── exceptions (Http Errors and other exceptions)
│   ├── __tests__
│       ├── index.spec.hs
│       ├── index.js (All the exeptions)
│   ├── filters (Vuejs filters)
│       ├── index (Module entrypoint)
│       ├── install.js (Install module, allow globally install filters)
│   ├── input-rules (Form field validation rules, eg.: Password validation)
│       ├── field (example)
│           ├── __tests__
│               ├── field.spec.js
│           ├── index.js (Module entrypoint)
│           ├── field.js (Rules/Form field validation)
│   ├── mixins (Re-usable business logic and common features)
│       ├── Autencicable
│       ├── InteractsWithBetSlip
│       ├── Pagination
│       ├── Selectable
│       ├── Etc
│       ├── index.js (Module entrypoint)
│   ├── plugins (Plugins that should be shared on all platforms)
│       ├── i18n.js
│   ├── resources (Static Resources)
│       ├── img
│       ├── lang
│       ├── icons.js (Material Design icons mapper)
│   ├── services (HTTP API, Storage and Locale services, etc)
│       ├── API (The base API service)
│           ├── APIService.js
│           ├── index.js (Entrypoint)
│       ├── BetSlip (Bet Slip services)
│       ├── Etc
│       ├── index.js (Services entrypoint)
│   ├── store (Vuex Plugin module)
│       ├── modules
│           ├── auth.js
│           ├── betslip.js
│           ├── user.js
│       ├── index.js (Module entrypoint + Install function)
│   ├── styles
│       ├── main.sass (Global styles shared at all platforms)
│   ├── views (Shared base views)
│       ├── View
│           ├── __tests__ (Tests for the View component)
│               ├── Vue.spec.js
│       ├── View.vue
│       ├── View.scss
│       ├── index.js (Entrypoint - Will publicly exopose the components that should be shared, in that case View.vue)
│   ├── main.js
│   ├── setupTests.js (Setup the Vue instance for testing)
├── .env.local (Development server env vars)
├── README.md (This file)
├── package.json (Package dependencies and config)
└── .gitignore
└── .jest.config.js
└── Dockerfile
└── docker-compose.yml
```

### Dev server, testing components and making changes

In this section I'm going to show you how to use the development server to test components, see your code changes in real-time and also be able to test component behavior with fake data.

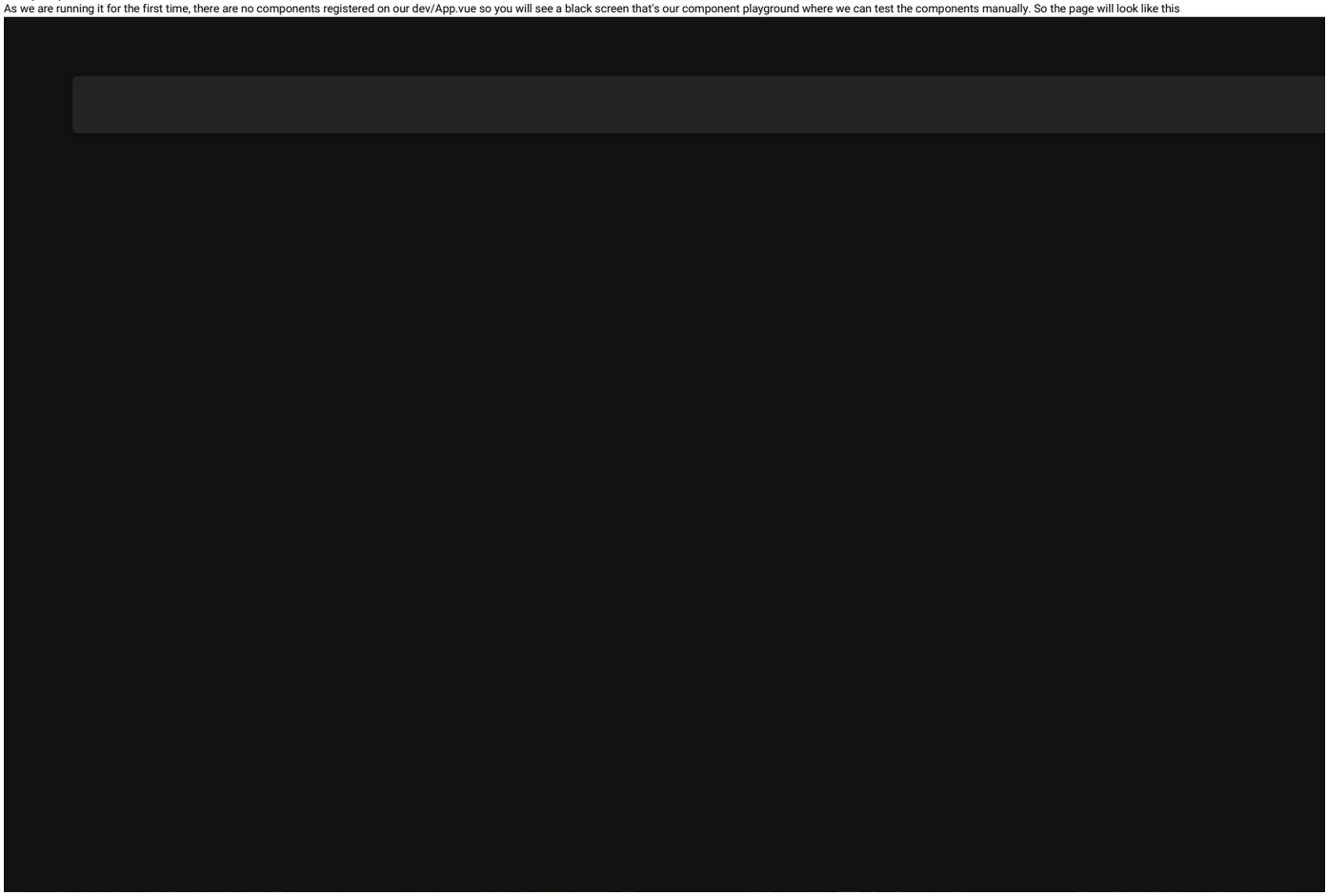
- So firstly let's start the development server (Skip this step if you are using Docker environment)

```
yarn dev
```

it will look like this

```
gustavo@nb:~/projetcs/betkudos/crypto-ui$ yarn dev
yarn run v1.22.5
warning package.json: No license field
$ vue-cli-service serve dev/main.js
 INFO  Starting development server...
40% building 29/38 modules 9 active /home/gustavo/projetcs/betkudos/crypto-ui/node_modules/vue-style-loader/lib/addStylesClient.jsBrowserslist: caniuse-lite is
npx browserslist@latest --update-db
98% after emitting CopyPlugin

 DONE  Compiled successfully in 4312ms


  App running at:
  - Local:   http://localhost:8080/
  - Network: http://10.0.0.141:8080/

  Note that the development build is not optimized.
  To create a production build, run yarn build.
```

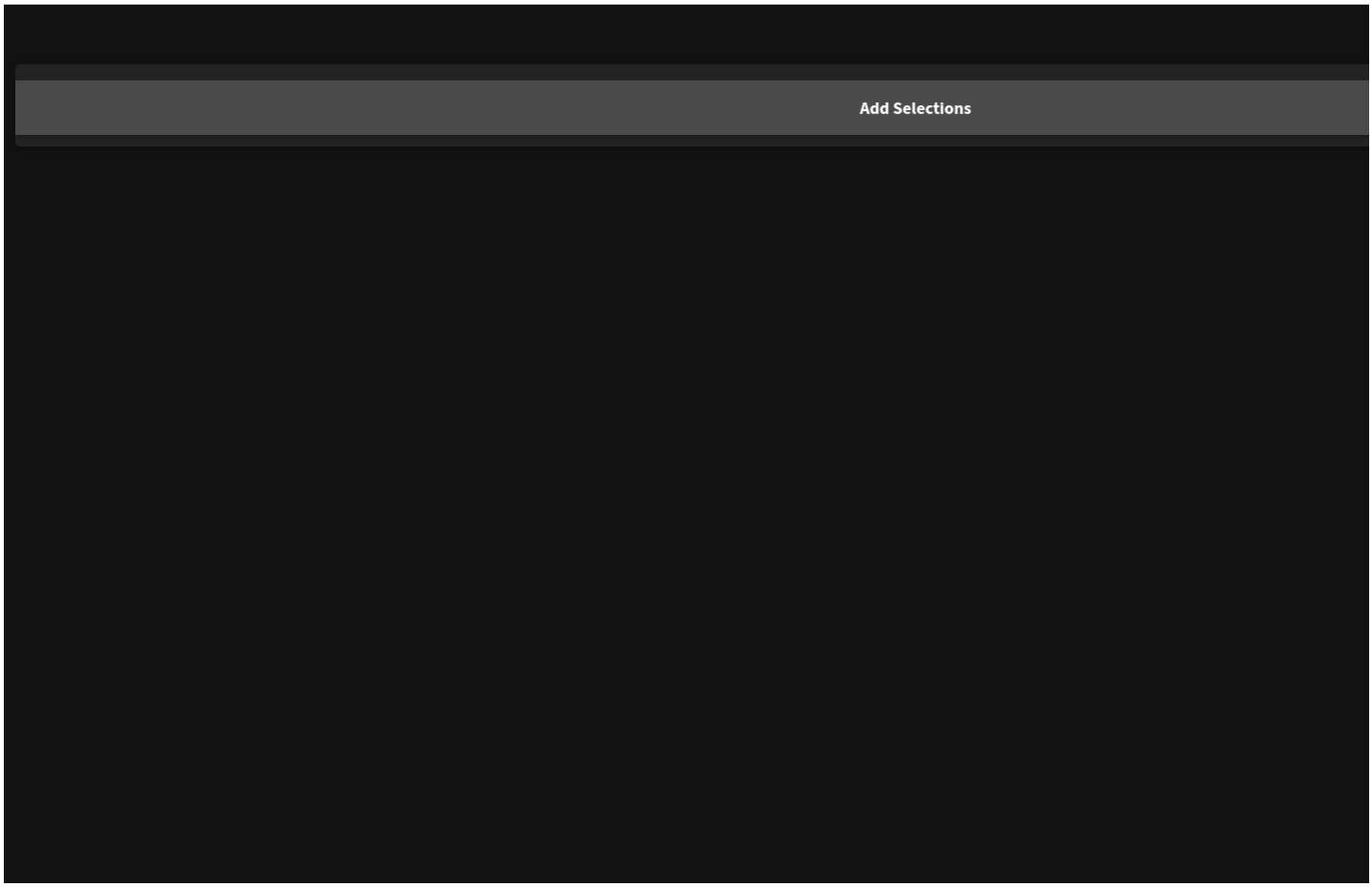
- Now go to your localhost server address un this case http://localhost:8080
- As we are running it for the first time, there are no components registered on our dev/App.vue so you will see a black screen that's our component playground where we can test the components manually. So the page will look like this



- So now open dev/App.vue in your code editor and make the following changes

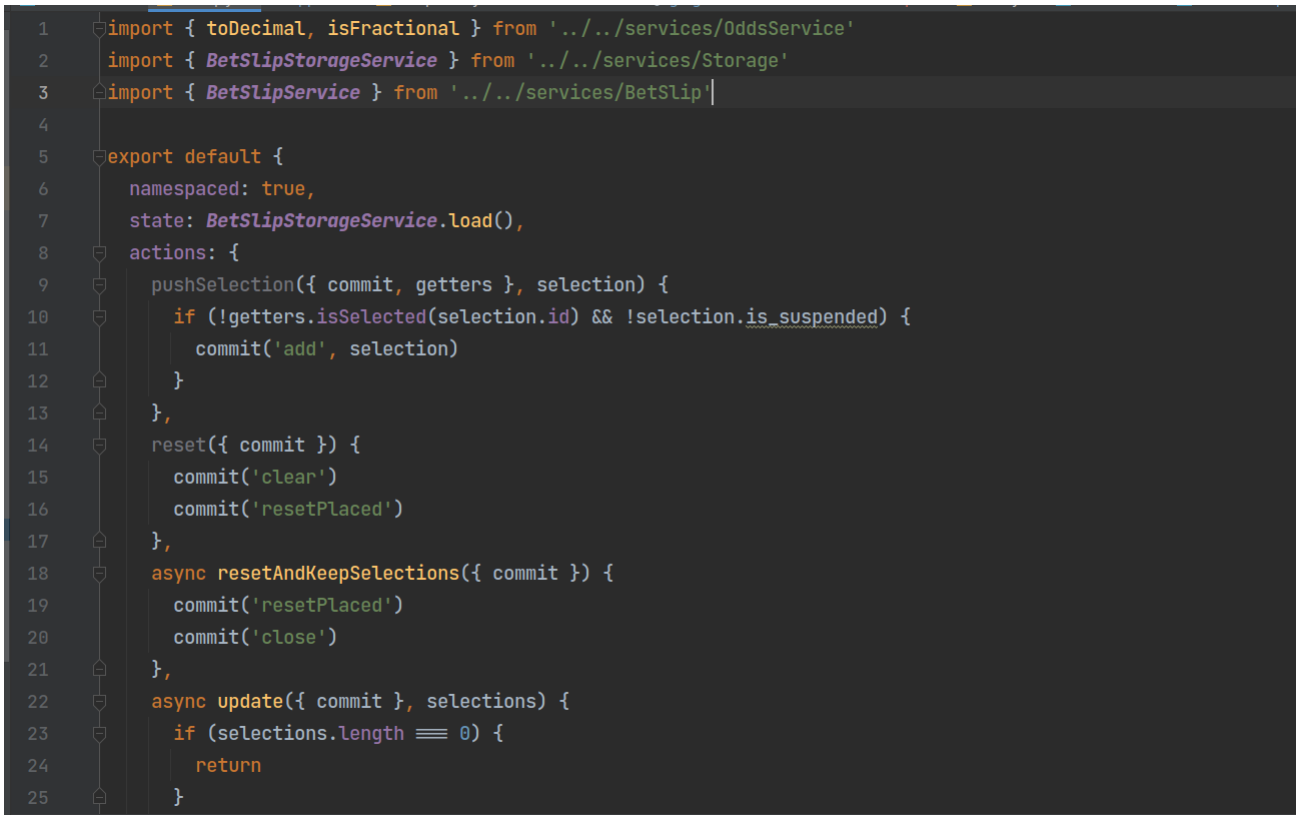
```
<template>
  <v-app :dark="true">
    <v-container fluid>
      <v-content>
        <div class="playground-wrapper elevation-3">
          <v-row>
            <v-col>
              <BetSlipWrapper></BetSlipWrapper>
            </v-col>
          </v-row>
        </div>
      </v-content>
    </v-container>
  </v-app>
</template>

<script>
import '@/styles/main.sass'
import BetSlipWrapper from './BetSlipWrapper'
export default {
  name: 'App.vue',

  components: { BetSlipWrapper },

  data: () => ({}),

  methods: {}
}
</script>

<style scoped>
.playground-wrapper {
  background-color: #232323;
  border-radius: 5px;
  padding: 0px;
  margin-top: 50px;
}
</style>
```

Now our page will load the BetSlipWrapper component and will look like this




As you can see the bet slip is empty, and we need to add some fake data to test it. If you to /dev folder you will see that we have a betslip-state.json file that has fake data to pupulate the betslip. The betslip data is managed by Vuex which is Vuejs official package to handle the application state. This package comes with Vuex Store module at /src/store, and there's a betslip module at /src/store/modules/betslip.js which has all the bet slip business logic and manage it's data.

So let's have a look at the Bet Slip Vuex module, it should look like this:

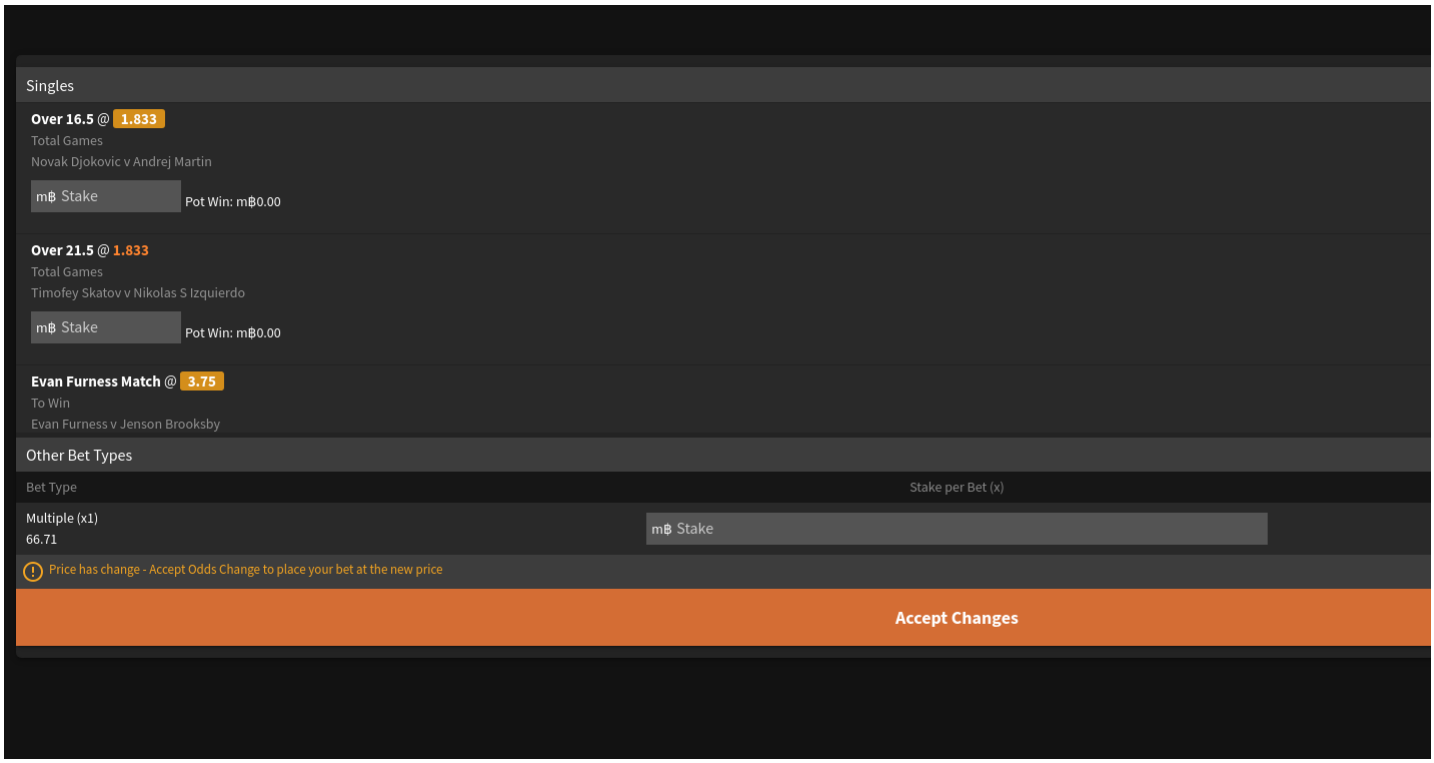


```
import { toDecimal, isFractional } from '../../services/OddsService'
import { BetSlipStorageService } from '../../services/Storage'
import { BetSlipService } from '../../services/BetSlip'

export default {
  namespaced: true,
  state: BetSlipStorageService.load(),
  actions: {
    pushSelection({ commit, getters }, selection) {
      if (!getters.isSelected(selection.id) && !selection.is_suspended) {
        commit('add', selection)
      }
    },
    reset({ commit }) {
      commit('clear')
      commit('resetPlaced')
    },
    async resetAndKeepSelections({ commit }) {
      commit('resetPlaced')
      commit('close')
    },
    async update({ commit }, selections) {
      if (selections.length === 0) {
        return
      }
```

As you can see the betslip.js module state is load by BetSlipStorageService.load() method. Which loads data from the localStorage API. So to populate the betslip data we just need to add it's data on our localStorage, So you have to run this on your browser:

```
localStorage.setItem('betslip', `{
  "placed": false,
  "visible": false,
  "selections": [
    {
      "eventName": "Novak Djokovic v Andrej Martin",
      "marketName": "Total Games",
      "id": 353062584,
      "market_id": 1606,
      "match_id": 122877,
      "match_market_id": 470280,
      "odds": 1.909,
      "name": "16.5",
      "header": "Over",
      "handicap": null,
      "is_suspended": false,
      "is_live": true,
      "order": 0
    },
    {
      "eventName": "Timofey Skatov v Nikolas S Izquierdo",
      "marketName": "Total Games",
      "id": 353062626,
      "market_id": 1606,
      "match_id": 122875,
      "match_market_id": 470282,
      "odds": 1.833,
      "name": "21.5",
      "header": "Over",
      "handicap": null,
      "is_suspended": false,
      "is_live": true,
      "order": 0
    },
    {
      "eventName": "Evan Furness v Jenson Brooksby",
      "marketName": "To Win",
      "id": 353048564,
      "market_id": 1605,
      "match_id": 122781,
      "match_market_id": 470275,
      "odds": 4.333,
      "name": "Match",
      "header": "Evan Furness",
      "handicap": null,
      "is_suspended": false,
      "is_live": true,
      "order": 0
    },
    {
      "eventName": "Kurumi Nara v Stefanie Voegele",
      "marketName": "To Win",
      "id": 353119101,
      "market_id": 1605,
      "match_id": 122821,
      "match_market_id": 470292,
      "odds": 2,
      "name": "Match",
      "header": "Kurumi Nara",
      "handicap": null,
      "is_suspended": false,
      "is_live": true,
      "order": 0
    },
    {
      "eventName": "Hua/Zhang v Cash/Stalder",
      "marketName": "To Win",
      "id": 353035402,
      "market_id": 1605,
      "match_id": 122706,
      "match_market_id": 470271,
      "odds": 2.2,
      "name": "Match",
      "header": "Hua/Zhang",
      "handicap": null,
      "is_suspended": false,
      "is_live": true,
      "order": 0
    }
  ],
  "changes": [
    {
      "id": 353062584,
      "match_id": 122877,
      "odds": 1.833,
      "changed": true,
      "suspended": false,
      "available": true
    },
    {
      "id": 353048564,
      "match_id": 122781,
      "odds": 3.75,
      "changed": true,
      "suspended": false,
      "available": true
    }
  ],
  "multipleStake": null
}`)
```

Now that you've set the betslip data on your localStorage, reload your page and the your betslip will look like this:

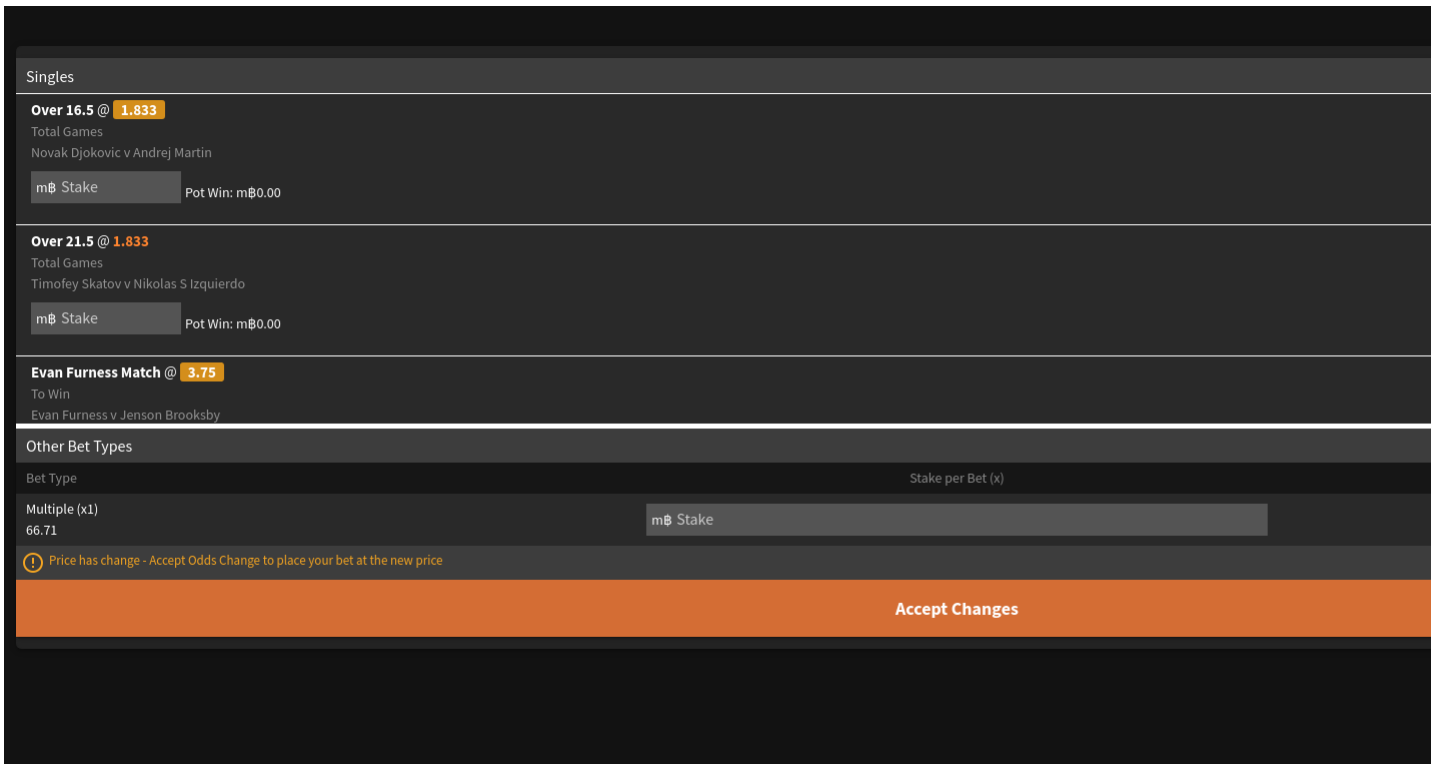



#### Making changes on components

Now that we have the betslip working on our dev server we can do any code change we need on it's components at /src/components/BetSlip, to give you a example let's change the betslip backgrond color to white by changing the BetSlip.scss file:

```
...
.bet-slip-wrapper {
  ...
  background-color: #ffffff !important;
  ...
}
...
```

It will look like this now:




To make changes on other components you should follow a similar process, but it will be easier than set up the betslip for testing since its one of our biggest components. So you have to:

- Import the component on /dev/App.vue
- Populate it with data if needed
- Do your changes on the components and test it, use unit testing if needed
- Commit your changes and make sure to do not commit the changes on /dev/App.vue. So this way other developers will have the playground always clean to test.

### Running your unit tests

```
yarn test:unit
```

### Lints and fixes files

```
yarn lint
```

# WEB DESKTOP CLIENT

### A vuejs desktop client for betkudos

## Project setup

### Dependencies

- Docker
- docker-compose
- Nodejs, npm and yarn

There's two ways to setup the project and run the development server

- Using docker-compose
- Using yarn/npm

### Docker Setup

To start with docker ensure you have Docker and docker-compose installed then you can just use docker-compose to build the container and install the NPM dependencies once.

Run

```
docker-compose up
```

This will build the container and install the NPM dependencies. The NPM dependencies will be installed at the first time you run `docker-compose up`.

Once the container is running it will start the development server, it can take a few seconds. So when the development server is running it will be listening for http://localhost:8081 on your machine.

### Yarn Setup

if you don't want to use Docker you will need to use yarn or npm to start the development server. To do so you will need to set up the NPM private package token because crypto-ui is a private package.

```
npm config set //registry.npmjs.org/:_authToken '82674aca-0053-4125-856a-8d445029922a'
```

Install dependencies

```
yarn
```

### Compiles and hot-reloads for development

```
yarn serve
```

### Compiles and minifies for production

```
yarn build
```

### Run your tests

```
yarn test
```

### Lints and fixes files

```
yarn lint
```

### Run your unit tests

```
yarn test:unit
```

# WEB MOBILE CLIENT

### A vuejs mobile client for betkudos

## Project setup

### Dependencies

- Docker
- docker-compose
- Nodejs, npm and yarn

There's two ways to setup the project and run the development server

- Using docker-compose
- Using yarn/npm

### Docker Setup

To start with docker ensure you have Docker and docker-compose installed then you can just use docker-compose to build the container and install the NPM dependencies once.

Run

```
docker-compose up
```

This will build the container and install the NPM dependencies. The NPM dependencies will be installed at the first time you run `docker-compose up`.

Once the container is running it will start the development server, it can take a few seconds. So when the development server is running it will be listening for http://localhost:8080 on your machine.

### Yarn Setup

if you don't want to use Docker you will need to use yarn or npm to start the development server. To do so you will need to set up the NPM private package token because crypto-ui is a private package.

```
npm config set //registry.npmjs.org/:_authToken '82674aca-0053-4125-856a-8d445029922a'
```

Install dependencies

```
yarn
```

### Compiles and hot-reloads for development

```
yarn serve
```

### Compiles and minifies for production

```
yarn build
```

### Run your tests

```
yarn test
```

### Lints and fixes files

```
yarn lint
```

### Run your unit tests

```
yarn test:unit
```

### Customize configuration

See Configuration Reference.

# BOST - Backoffice System Tool

## Project setup

```
yarn install
```

### Compiles and hot-reloads for development

```
yarn serve
```

### Compiles and minifies for production

```
yarn build
```

### Run your unit tests

```
yarn test:unit
```

### Lints and fixes files

```
yarn lint
```