EECS 581 Team 8 Project Proposal

Team Members:

- Cole Adam
- Aris Vinsant
- Beau Hodes
- Thomas Angles
- Justin Schreiner

Team No.
Team 8

Project Name:
Betliquid

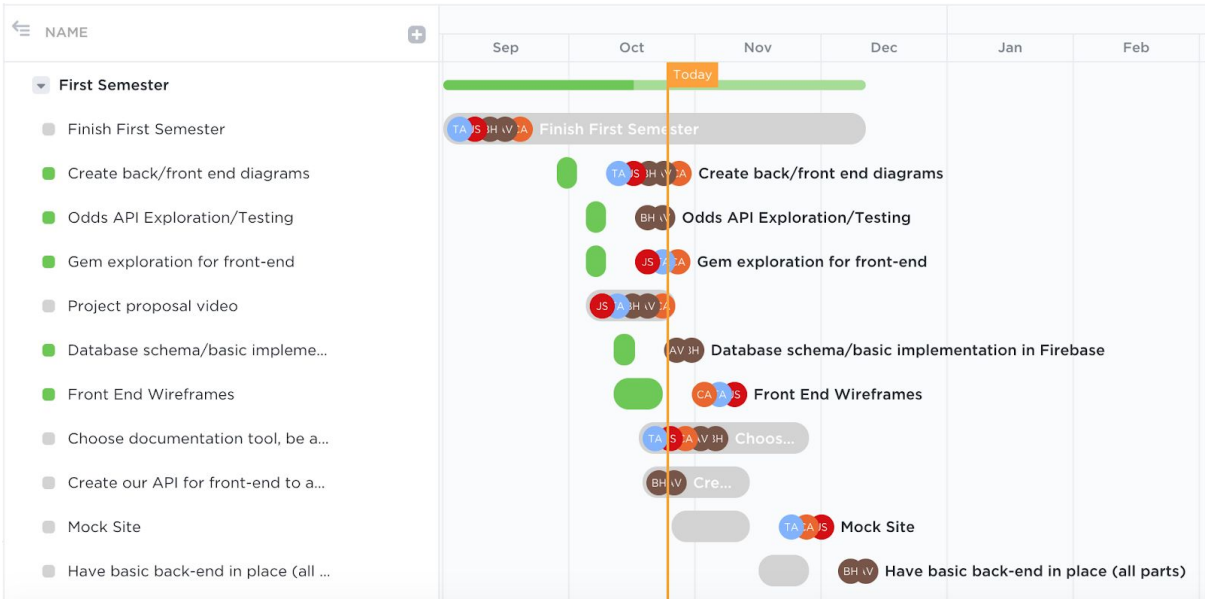
Project Synopsis (1-25 words):
Online exchange-based betting platform that allows users to predict the outcome of an event by buying shares in the end result.

Project Description (150-250 words):
In the last two years, the market for legal sports betting in the United States has exploded. This is largely the result of the supreme court's 2018 decision to overturn the federal ban on betting. Since then, many large players in the casino & gaming industry have fought for market share using the same outdated betting platforms with high margins for the house and low liquidity for gamers looking to actively enter and exit their bets in real time. Our project is to create an exchange that allows players to bet against each other at all times. Ultimately, the bettors are betting the hopes of receiving a fixed payout of $1 per "share". The price of each share is the implied probability of that proposition coming true (in cents), and a payout of $1 will be made for each share that cashes out correctly. Our platform will use traditional sports betting lines to create an "opening line", and the free market of players will be able to take it from there. Our primary focus is to create an exchange for sports betting, but our ultimate vision is to create one online exchange for all prediction markets, regardless of the event.

Project Milestones

- First semester:
    - Have back-end in place (Firebase set up, at least 2 endpoints in our own API finished/working such as adding users or updating bets). This will also include deciding between NodeJS, ExpressJS, Pyrebase, or C++ for our own API. Estimated completion date: 11/27

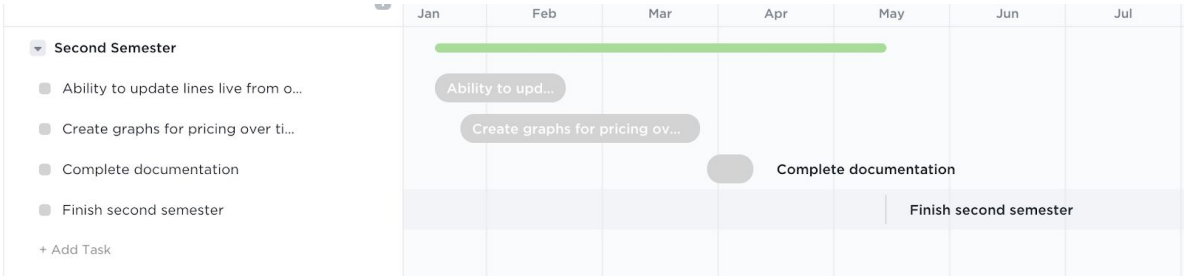
- Have documentation for back-end AND front-end in place (does not need to be fully written out, but deciding on a documentation tool and being able to generate documentation at will should be done).
  Estimated completion date: 11/27
- Ability to translate odds API data into implied probabilities before storing in Firebase (this will use Pyrebase).
  Estimated completion date: 10/16
- Create a comprehensive list of the gems that we will implement in the final design.
  Estimated completion date: 11/2
- Make github repo that incorporates continuous integration using TravisCI.
  Estimated completion date: 11/2
- Design the process of how the front end will communicate with the backend API (will likely need to add gems to accomplish this).
  Estimated completion date: 11/27
- Schedule the function for pulling odds to run every 10 minutes (can increase frequency when actually using the application).
  Estimated completion date: 10/31
- Schedule a function to run once per day that removes events and positions that have expired over 30 days ago.
  Estimated completion date: 11/7
- First Semester Gantt Chart:



- Second semester:
  - Be able to update lines directly from implied probabilities (which will be pulled from odds API) for testing/demo purposes. In real life, the market would decide

pricing, but we need to have this ability for testing/demo.
Estimated completion date: 2/19

- Create graphs for pricing over time for events (for instance, the price of the implied probability for the Chiefs to win a certain game, starting from a week before the game up until the game is over, at which point it would be $1 or $0).
  Estimated completion date: 3/26
- Complete documentation.
  Estimated completion date: 4/9 (roughly)
- Second Semester Gantt Chart:

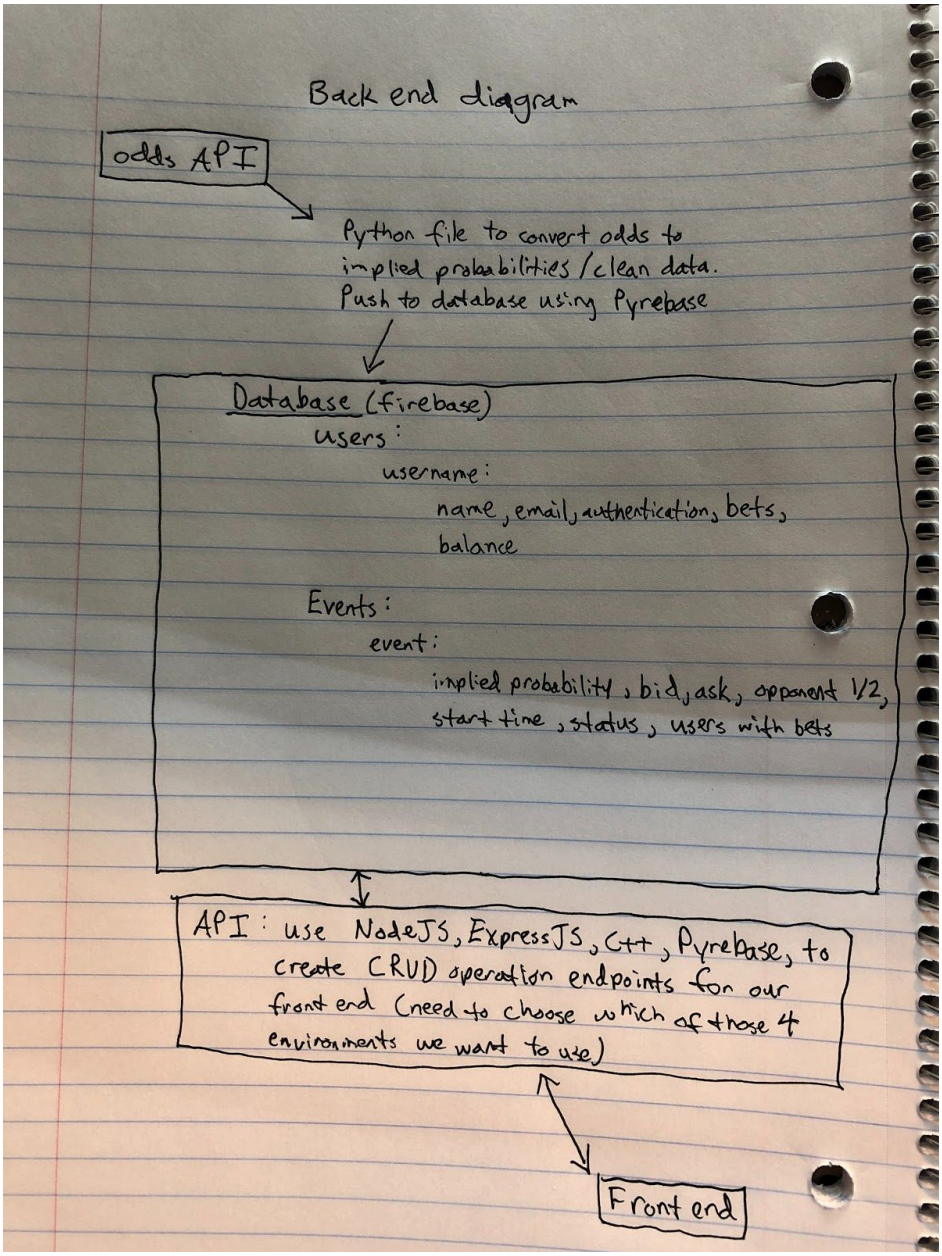


Project Budget:

- Domain Name - liquid.bet
  - $25
  - Godaddy.com
  - Date needed: ASAP
    - We've found a domain that we plan on using, so we hope to buy this as soon as possible so that we can secure the name.
- API requests - the Odds API, Rookie Subscription
  - $15/month
  - Date needed: January-May
  - We'll be able to scale down our project initially and reduce the scope of the available events which we'll allow users to place bets on. As the project gets closer to the final iterations, we'll need to expand the scope to allow the site to be fully functional.
- Google Cloud Platform compute credits
  - Used to pay for firebase datastore and app engine
  - Estimated cost: $10/month
- Total estimated costs:
  - $150

Preliminary Project Design: How the software works

- **Back-end**
    - Here is our back-end diagram:



    - Notable changes that we have made since creating the diagram are:
        - Replacing Pyrebase for pushing odds API data to Firebase with a Javascript function written in Google Cloud
            - Firebase has built in support for scheduled functions in JavaScript, so it made more sense for us to use JavaScript rather than Python with Pyrebase. We will be able to change the frequency with which the odds are pulled. For instance, we could set it to every 10

minutes if there are no events going on at that time, or every 10 seconds if there are live events happening.
    - Allowing Firebase to handle authentication with its built in functions
    - Adding a "positions" entity to the database
    - Deciding to use NodeJS for our front-end facing API
    - Deciding to use the odds from the odds API for pricing for demo purposes, since a market-based system would not have enough liquidity with our small user-base
    - Deciding to delete events/positions after 30 days
- The odds API is pulled by a scheduled Google cloud function, written in JavaScript. We convert the unix UTC dates into a "month/day/year 00:00" format and convert the odds (moneyline, spread, and totals) into implied probabilities, and then push the new odds to Firebase if the event already exists. If the event does not exist, we have to generate a new key and create the new event.
- Firebase uses Firestore to store all of our user, position, and event data. Here is our schema:

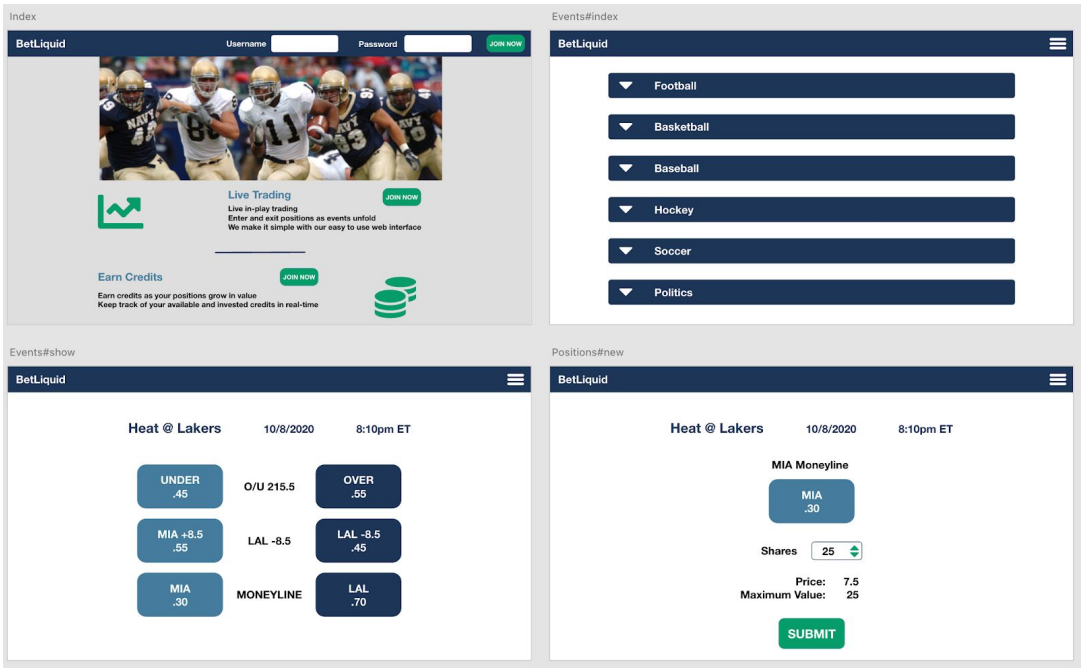
```
users: {
        [generated userID/key]: {
                username: [username]
                password : [pwd]
                email: [email]
                account balance

positions: {
        [positionId]: {
                userID
                eventId
                current price
                # of shares
                Share price at buy time
                time accepted
                outcome
                outcome time
events: {
        [eventId]: {
                Description
                Open date/time
                completion date/time
                Team1
                Team2
                Price per share for each bet (ML, spread, total) bid and ask
                Users (list of users who have positions on this. Firebase recommends this)
```

Users will also include all positions that the user has (up to 30 days ago).

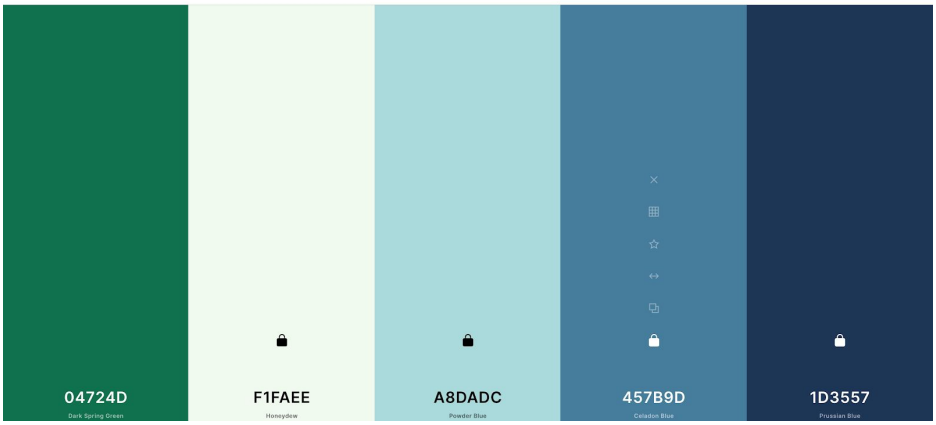
Firestore uses collections and documents to store data. These can be accessed, added, updated, or deleted through tools provided by Firebase that work with our JavaScript code. A collection holds multiple documents, and you can nest collections inside documents (and add documents to those collections) if

needed. For instance, “users” is a collection, and the positions that the user holds will be a document under “users” that holds a subcollection of positionId’s as documents. Collections and documents must always alternate in the hierarchy (for example, you cannot have a subcollection directly under “users”. The subcollection must be under a document which is under “users”.) “positions” will also be a collection of position documents, and “events” will be a collection of event documents.
        - Although it may seem strange to store positions for a user both in the “positions” collection and under the “user” collection for that user, Firebase actually recommends this practice in order to speed up access time.
    - The back-end will communicate with the front-end through API endpoints. The API will be written in Javascript using a NodeJS project. Endpoints for things like getting all positions for a user, authenticating users, initializing a position, updating a position (selling some shares or buying some shares), and exiting a position will be put in place. Firestore supports CRUD operations written in JavaScript, so we can access Firestore directly from the NodeJS project. Firestore also takes care of atomicity of transactions and generating user tokens to validate user sessions.
    - In order to keep the number of events and positions that we have stored in the database in check, we will delete events and positions 30 days after they end. To do this, we can schedule a function to run once per day that will scan all events and positions, remove them from Firebase if they are over 30 days old, and communicate this fact to the front-end.

- **Front-end**
  - Here is our front end mockup diagram:



  - Here is our front end color schema:



  - The front end will be run using the framework called Ruby on Rails. This Framework runs with the ruby language. Rails works with small tools called gems that aim to accomplish specific subtasks that are open-source and easy to integrate into the front end design stack. We are currently working on listing each of the gems in a comprehensive list. Gems are going to really help the front end design process because we can use specific gems to help with adding fonts, well structured odds displays, and all together helping the pages look much cleaner. Also, there is a gem that makes the utilization of Google Firebase with Rails extraordinarily simple and we plan on getting that gem set up as soon as possible which will allow us to link the front-end with the back-end of the project. Some other gems we plan on implementing is ChartKick for helping show the data from

the odds API as an updating graph image, this will allow users to have a simple view of the event they are looking at.
    - Rails is an easy to use MVC framework which will align nicely with our backend. For each model in the backend schema (users, positions, and events), there will be a corresponding model in Rails. Each model has specific actions that have corresponding views. For example, our 'events' model will have several actions including 'index' and 'show'. Each of these actions will have its own view, where the user will interact with that specific model.
    - The users will first be taken to a login page. After a successful login, they will be taken to the 'index' view for events. Here, they will be able to see all events under their respective category (baseball, football, etc.). Once they select an event, they'll be taken to the 'show' page for that event. Here, they will be able to see information about the event and positions they can take that correlate with that event. Once the user has selected a position to take for that event, they will be taken to the 'new' position page. Here, they will see metadata from this event and be able to select the number of shares they will hold. After submitting their position, users will be able to view their current positions by visiting the 'show' user page. Here, they can see their available and current balance, as well as a list of current positions they have taken. If a user wishes to either buy more shares or sell their current shares, they will have the option to visit the 'edit' position page. Once an event has finished, if the user held shares of the event their current balance will be updated accordingly.
- **Design Constraints**
    - Cross-Platform Viability: Our application is designed to run on the web. This may prove difficult if we ever decide to expand our product onto different platforms or operating systems.
    - Critical Mass of Users: In order to provide sufficient liquidity for our bets, we need to have enough users so that orders will be filled. Liquidity is the most important thing in any marketplace product, and a large number of active users is generally required to make this happen.
    - Media licensing: It may be difficult to display logos or images that provide a visual aid for our service. If we start with sports bets, we would have to set up licensing agreements to use the team and league logos. This could be very expensive, and may lie outside the initial purview of our implementation.
    - Limited team experience in the space: Our team members are very knowledgeable about the subject matter, and are quick learners, but many of the frameworks and platforms we are using to implement our software are new to the group and take some time to learn.

Ethical Issues:

- Ethical Consideration #1: Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing.
    - The first ethical issue in the ACM code of ethics revolves around the primary goal of creating a positive contribution to society. Many skeptics could argue that our service violates this purpose to foster a general well being because it involves the risk of losing money. We argue that our service does not violate this statement because:
        - A more liquid market is more financially rewarding for gamers because the "house" is taking a significantly lower rake on all bets. This is much better for our customers and adds up significantly in the long run.
        - We are willing to place wager limits on individual accounts and direct applicable help resources if a gamer develops a gambling problem.
        - We eventually hope to develop a "sandbox mode" where gamers can practice and become more knowledgeable about betting before falling prey to a larger casino business and losing more money than intended.
- Ethical Consideration #2: Honor confidentiality.
    - Because our service involves bets that represent actual currency, our service will have to be able to handle financial transactions such as cryptocurrency payments or bank account information. We pledge to honor the confidentiality of our customers by:
        - Using Google Cloud User Authentication to ensure login credentials are safe and secure.
        - Encrypting balance and transaction information for payments that happen on our website.
        - Encrypting login passwords so that they are secure and unable to fall into the wrong hands.
- Ethical Consideration #3: Perform work only in areas of competence.
    - Setting up a marketplace to function as a liquid exchange is inherently confusing. There are a lot of moving parts and back end calculations that are required to accurately display prices that ensure viability in our platform. Luckily, members of our own team have studied the functions of a marketplace using American stock exchanges as a good example. We are familiar with many sports betting and exchange-based concepts, and we feel as though we have more than sufficient competence to set up this platform correctly. We have divided our team into front end and back end development so that each team member is working within their purview of skills and expertise.

Intellectual Property Issues:

- Intellectual Property Consideration #1: Media content
    - Perhaps the most difficult obstacle for avoiding intellectual property violations revolves around the use of media content. Examples of this kind of content include designs, logos, and trademarks that may be the property of individual sports teams or leagues. In order to legally use this media, we would have to obtain licensing agreements to use those forms of media from their respective owners. This is something we would really like to set up in the future, but is probably not feasible for the scale of our project at the current time. Luckily, the odds that we pull from an API and statistics regarding the final outcome of the events are not protected as intellectual property, so our platform will be able to function with those data points.
- Intellectual Property Consideration #2: Exchange-based logic
    - Our exchange plans to use a bid/ask spread model for creating our liquid betting exchange. You can learn more about how the bid/ask spread model works by going here. This logical model has been implemented and used under a variety of different titles in many different stock exchanges across the world. As such, the logic that governs this strategy is not protected by intellectual property rights. There are some reserved words that are specific to some exchanges such as NASDAQ and NYSE, but we do not plan on using any of these keywords in our project.

Change Log:

- Added first semester milestone regarding scheduling the function to pull odds from the odds API.
- Added first semester milestone regarding scheduling the function to remove events and positions that are over 30 days old.
- Added budget information to explain where we'll be purchasing our domain name from.