DealDex Engineering Design

Github: https://github.com/coinfounded/dealdex
Figma: https://www.figma.com/file/G0TRAXjFgIw8RD6YuhHV1S/DealDex-UI

Prerequisites

1. Migrate from backend from Firebase to a more web3-friendly solution like Moralis
    a. Configure local Moralis support for other devs (if something special has to be done for this)
    b. Does Moralis support everything we need? Or will we need to have a hybrid firebase-moralis backend?
2. Frontend Data Layer Refactoring
    a. Need to split up data layer into Models and Services
    b. Services
        i. DealService
        ii. UserService
        iii. FirestoreService (or DBService)
        iv. AuthService
        v. All firebase calls should only be in FirestoreService and AuthService so we can swap it out easily
    c. Models
        i. Deal
        ii. User
        iii. NFT
3. IAM roles
    a. Create a new IAM role + permissions for new contributors

Smart Contract

Deal Factory

1. Implements the Clone Factory pattern
2. Is upgradeable (uses OpenZeppelin proxy smart contract pattern)

Events:

```
DealCreated(address indexed creator, Deal deal)
```

Interface:

```
createDeal(DealConfig memory _dealConfig)
```

- Creates a Deal contract and emits a DealCreated event

`calcInvestorTokenFee(...)`

- `Returns the number of tokens that we collect in fees - currently default this to 2.5%`

Deal

Events:

`Invested(address indexed investor, Deal deal, uint256 amount)`

Interface:

`init(DealConfig memory _dealConfig)`

- Initializes the dealConfig
    - Exchange rate: fraction
        - Project token / payment token
        - Note we may not have the project token decimals yet
        - Fixed point number? Or two units?
            - https://github.com/Uniswap/solidity-lib/blob/master/contracts/libraries/FixedPoint.sol
            - https://ortiz.sh/blockchain/2021/05/08/QMATH.html
    - Deal creator and project wallet addresses
    - Round min/max constraint
    - Investor max/min constraints
    - NFT address
    - Deadline
    - Project token address: optional
    - Syndication fee: % of project tokens, wallet that will receive fee
    - DealDex fee (calculated by calling *calcInvestorTokenFee*(...) in deal factory):
        - % of payment tokens,
        - % of project tokens // default this to 0 for now
        - wallet that will receive fee
    - Payment token address: required
    - Vesting schedule
        - Array of percentages: last one must be 100%
        - Array of timestamps
        - Enum of vesting strategy:
            - Proportional: percentage of tokens vested applies to all investors the same way
            - Time order: investors that invested first can withdraw tokens first
            - Small to high investment size:
- Sets the initial totalReceivedInvestment to be 0
- Locks the deal if the minimum total investment is 0

- Sets the number of project tokens to be 0

`invest(amount, nft_id)`

- Adds the invested amount and the sender to the total received investment and list of investors for the deal
- Use approve (happens on the frontend) + transferFrom (happens in the smart contract) on the payment token contract to receive investment
    - https://docs.openzeppelin.com/contracts/2.x/api/token/erc20
- Requires all of the following:
    - Investment amount is neither too low nor too high
    - Investment does not exceed the total investment cap
    - Investment deadline has not passed
    - Investor owns requisite NFT for gated deal
    - Max round size has not been reached yet
- Record the investment using the NFT id

`claimFunds()`

- Method used for project to claim its funds (in the payment token)
- Tokens are being sent from the Smart contract to the startup address
    - No need to call approve anymore (since we're sending our own money)
    - Only use transfer here
- DealDex fees get sent to DealDex wallet (in payment token)
- Requires all of the following:
    - User trying to claim funds is the project itself
    - Deal is valid
    - Deadline has passed

`claimRefund(nft_id)`

- Method used for investor to take back funds they have invested
- Tokens are being sent from the Smart contract to the investor address
    - Only use transfer here
- The refund amount depends on the amount invested using the NFT
- Requires one of the following:
    - Deal is invalid, meaning deal has fallen through
    - Investment deadline has NOT passed

`claimTokens(nft_id)`

- Method used for investor to claim their (vested) tokens
    - Vesting schedule is defined as array of percentages, array of timestamps + the enum. Defined in create deal.

- Tokens are being sent from the Smart contract to the investor address
    - No need to call approve anymore (since we're sending our own money)
    - Only use transfer here
- Requires the following:
    - Startup token must have been specified
    - Deal is valid (check based on lock status)
    - Investor must not have already claimed all their vested tokens (note this means that an investor can claim their tokens multiple times as long as the sum total of all token claimings is less than their vested # of tokens)
    - Contract has enough tokens to send to investor
    - Require # of tokens deposited >= amt to withdraw
    - Investors can ONLY claim tokens that were deposited by the depositTokens() function

`depositTokens()`
- Method used by project to distribute tokens to investors
- Send % fees to the syndicate wallet address (in project token)
- Send % of fees to the DealDex network wallet address (in project token)
- Set # of tokens deposited

`updateProjectToken(contract_address)`
- `Updates the project token to the new ERC20 defined in the contract address`
- `Only the project can call it`

`updateExchangeRate(new_rate)`
- `Updates the exchange rate between the project token and the payment token`
- `Only the project can call it`

Backend

Different blockchains (eth mainnet, polygon, localhost, etc.)

Need Error alert popup for wrong chain
1. Need to have a separate collection for each blockchain
2. Supported blockchains for V1:
    a. Mumbai testnet
    b. Polygon Mainnet
    c. Ropsten Testnet?
    d. Ethereum Mainnet?

Users

```
/users/{wallet_address}/
```

For each user (wallet address), store

```
Wallet_address: string
Username: string
Verified_status: boolean // default to false
Telegram_username: string
Deals_created: [string] // array of deal contract addresses
Pending_deals_created: [string] // array of transaction hashes
```

- Note that for pending deals we need to wait until the transaction is complete, then use the DealCreated event to get the deal contract address

NFTs

```
/nfts/{contract_address}/{nft_id}/
```

For each NFT, store

```
NFT_contract_address: string
NFT_id: string
Deals_invested: [string] // array of deal addresses this NFT was used to
invest in
Pending_investments: [string] // array of transaction hashes
```

Deals

```
/deals/{contract_address}/
```

For each deal, store

```
Deal_contract_address: string
Deal_name: string
Investor_payment_token: string
Deal_creator_address: string // wallet address of deal creator
```

```
Investment_deadline: int // Unix timestamp of investment deadline
```

Frontend

Navigation Bar

Figma:

- https://www.figma.com/file/G0TRAXjFgIw8RD6YuhHV1S/DealDex-UI?node-id=805%3A9258

Features:

- Connect Wallet
    - When this is clicked, metamask is opened and signing occurs. A loading indicator is displayed till the authentication has succeeded.
    - Should display loading indicator when you refresh the page and auth status is verified
    - Upon successful connection, wallet address is displayed
    - Wallet selection and connection state monitoring should be done with BlockNative Onboard: https://www.blocknative.com/onboard

- Change network
    - Can be changed to Ethereum Mainnet, Testnet, etc. (wrapper around metamask network change)
    - Display error for incorrect network (how to check for incorrect network?)

Backend Dependencies:

- Need to sync with user data stored in DB. either through firebase or moralis.

Create Deal Form

Figma:

- https://www.figma.com/file/G0TRAXjFgIw8RD6YuhHV1S/DealDex-UI?node-id=805%3A9307

Features:

- 5 Step process, visible even while logged out
    - Wallet connection is same as in Nav Bar
    - Step 1: Deal Name
        - Enter in string name of investment deal
    - Step 2: Investment Constraints

        - NFT Needed to invest in deal. Investors can claim project tokens with the NFT that was used to invest
        - Address of ERC Token used for investing
        - Min/Max Round Size (if min round size isn't reached, then investors can claim refund)
        - Min/Max Investment per investor
        - Investment Deadline
    - Step 3: Project Details
        - Project Wallet (required field)
        - Project Token Price (required)
        - Project Token (ERC20 Contract Address), can be specified later
            - Show if token has been validated. Show the Name, Symbol, and Balance
        - Vest Date, Vest Percentage, can be added to vesting schedule
            - Ensure that the last percentage is 100 after things have been added to vesting schedule
            - In a future version: Add some graphic that shows like some Pie chart where the vesting schedule gets more and more populated as new Vest Date/Percentage pairs are added to the vesting schedule
    - Step 4: Fees
        - Syndicate Wallet Address -> every deal can have atm ost one Syndicate associated with it, Wallet receives fees when project's tokens are deposited in the deal
        - Syndication Fee (as a percent)
    - Step 5: Review and Submit
        - Provides user the button to Create a Deal

Backend Dependencies:

- Need to be able to set human readable Deal name
- Make deal name unique

Smart Contract Dependencies

`init(DealConfig memory _dealConfig)`

- NFT requirement
- Payment token (erc20)
- min/max round size
- min/max invest per investor
- Investment deadline
- Project wallet
- Project token price
- Project token (erc20, optional)
- Vesting (optional)
    - A vesting schedule is two arrays
        - Array of Unix timestamps (milliseconds or seconds?) containing dates
        - Array of percentages (how do we store percentages)

- Both Syndication and Dealdex fees
    - DealDex fees (wallet + percentage) are specified in contract factory
    - Syndication fees: specify wallet + percentage

Notes:

- Can implement custom steps progress bar (or use https://github.com/jeanverster/chakra-ui-steps)

Home Page

Figma:

- https://www.figma.com/file/G0TRAXjFgIw8RD6YuhHV1S/DealDex-UI?node-id=0%3A1

Features:

- Display all deals
    - Show whether a syndicate is Verified
    - Display the syndicate name, required NFT, min investment amount, and deadline
        - Even show deals whose deadline has passed
        - Usernames will be unique, no duplicates
- Button for Create a Deal

Backend Dependencies:

- For each deal store
    - Deal Name
    - Syndicate Address
    - Required NFT
    - Payment token
    - Min investment amount
    - Deadline (how many days left, and whether it has passed, this computation will be done on the frontend)
- For each user store
    - Username
    - Enforce that username is unique

My Account

Figma:

- https://www.figma.com/file/G0TRAXjFgIw8RD6YuhHV1S/DealDex-UI?node-id=803%3A9204

Features:

- Information specific to an account:
    - Profile information:
        - Give the user editing capability for their username/email address and profile picture

            - NFT integration with profile picture image, similar to what twitter does with people's profile pictures being NFTs
    - Investments: Wallet holds NFTs which have invested in the following deals
        - Show the Deal Creator, My Investment amount, and Status (whether it is claimable or not)
    - Deals: Wallet has created the following deals
        - Show the requisite NFT, minimum round size, and deadline (number of days left till deadline)
    - Projects: Wallet has been the recipient of following deals
        - Show the NFT which the deal is gated by, funds raised so far, and whether the deal has closed or not (check whether the fundraising goal has been met before the deadline
    - Need to show pending deals + pending investments as well (using transaction hash + Moralis?)

Backend Dependencies:

What to store:

- Need to store all the NFTs used to invest on dealdex
- For each NFT
    - Store the contract addresses of the deals that the NFT was used to invest in
- For each user
    - Store the contract addresses of the deals that the user created
    - Store the contract addresses of the deals where the user is the project raising funds
- For each NFT in the user's wallet check if it is in the DB as a NFT used for investment
    - https://ethereum.stackexchange.com/questions/106942/what-process-does-opensea-use-to-get-all-the-nfts-of-a-wallet/106949
    - Use Moralis: /nft/wallet/{owner_address}

How to store it:

- Old way of creating deals (entirely done on frontend, could result in lag/delay):
    - Store a transaction hash on user doc
    - Once the user goes to My Account page, query for transaction and wait for transaction to complete
    - Delete transaction hash and add deal smart contract to firebase
- Firebase needed for storing metadata (including names/usernames)
- Moralis for keeping track of deal creation in smart contract
    - Moralis is cross-chain, could be beneficial
    - Use Moralis SDK to mark an event for deal occurring
        - https://docs.moralis.io/moralis-server/automatic-transaction-sync/smart-contract-events
        - Need an abi for the event, so that Moralis will parse fields and populate schema correctly
    -
- Infura? Firebase? Moralis?
-

How frontend will access it:

- Moralis claims they have some way of easily integrating with frontend

Smart Contract Dependencies

- Status (need to standardize what this is, maybe put some enums)
    - OPEN_TO_INVESTMENTS
    - CLOSED_TO_INVESTMENTS
    - CANCELED
    - VESTING
    - COMPLETED
    - etc.
- My investment amount (in payment token)
- Funds raised

Deal Details View

Note that this view will change in the event that the investment has occurred and also changes depending on who is viewing it: investor vs syndicate or manager vs project.

Figma:

- https://www.figma.com/file/G0TRAXjFgIw8RD6YuhHV1S/DealDex-UI?node-id=446%3A1052

Features:

- Summary
    - Status: Shows whether the deal has been completed, is still open to investments (but you haven't invested in it yet), or has been
    - Deadline: UTC time (date and HH:MM:SS) in which the deal expires by
    - Investment Per NFT: range of investments allowed for ownership of a single NFT
        - Note that in the database, we'll have to pair (user, NFT) in order to check in the Invest portion that the investment amount falls within the limits we show here for investment per NFT
    - Required NFT: Show the name of the NFT, both the short form and the full form
    - Token Price: Show in USDC, may have to do conversion or calculation
- The Project
    - Name, Token, Token Price, Total Raised
    - Token and Token price can be updated by project
- Invest
    - Select NFT (since there's an investment range per NFT, which we standardize currently. From a user interface and backend perspective it's much simpler to have one range for investment per NFT)
    - Ticker for amount in any ERC20
    - Expected number of tokens which changes based on the amount entered. Will need to do conversion between the Syndicate's token and the USDC amount entered. Number of decimal places for Expected token amount?

        - Connect wallet
            - Uses the same authentication mechanism as in the Connect Wallet in the Nav Bar
- Progress
    - Vesting
        - Step Progress bar which displays percentages (currently in quarters, but can have as many partitions as needed) and shows the date for each vesting period
    - Funding amount
        - Round Size and Amount raised
        - Progress bar for the amount of ERC20 remaining in the funding campaign, similar to how Kickstarter displays this
- Invest becomes Claim Tokens/Funds
    - Claim Tokens has the same interface as Invest but this time calls the smart contract function to claim tokens
- Other Information
    - Vesting is preserved from Summary section
    - Deposit Tokens allows the project to deposit a specified amount of their token
- If Syndicate is viewing it:
    - Can see the Subscribed Investors for the deal
        - Name and Investment amounts in table

Backend Dependencies:

- Deal Name
- Project Name
- Syndicate Name
- Investor Names

Smart Contract Dependencies:

- Deadline
- Subscribed investors
- Amt raised
- Investment amt per NFT, tokens claimed per NFT
- Round size, investment limits
- Vesting schedule
- Project token + token price
- Payment token
- Investor actions
    - Invest, Claim Tokens, Claim Refund
- Project actions
    - Claim Funds, Deposit Tokens, Update token, Update token price, Edit vesting
- Syndicate actions
    - Same as investor (+ they can see the subscribed investors)

DealDex V2.0 and beyond

- Need a KYC product like Jumio or Blockpass