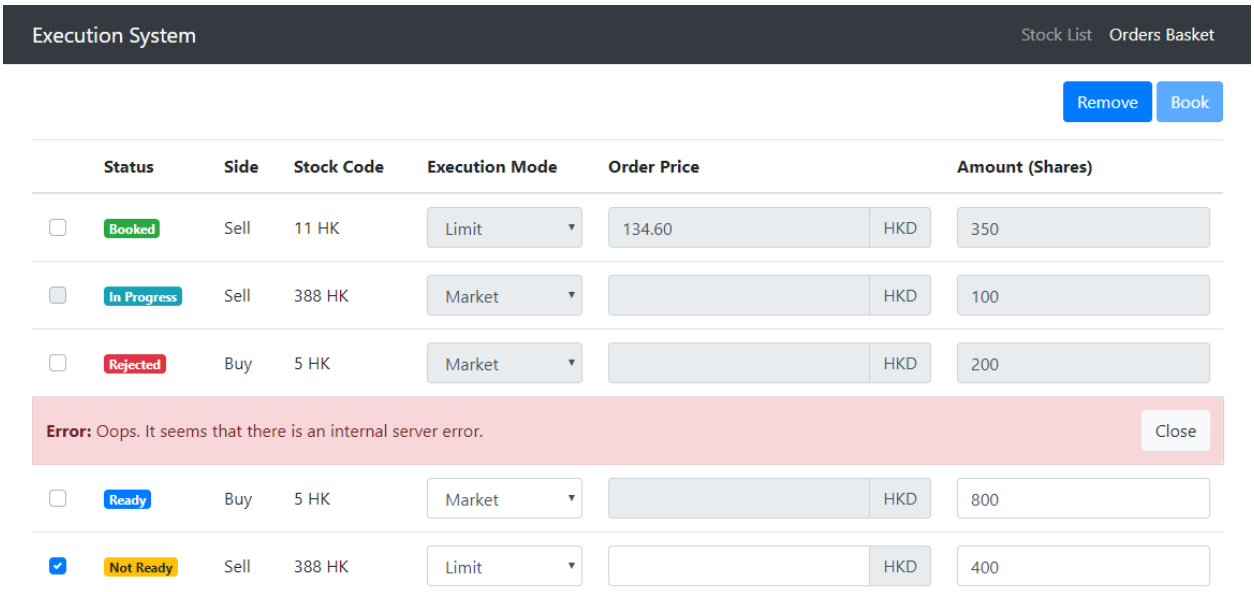
CODING EXERCISE

STOCK EXECUTION SYSTEM

Please take some time to go through the instructions below and produce your best implementation of the application requested.

You are going to develop a simple stock execution system that allows banking clients to place orders to buy or sell stocks.



YOUR TASKS

You are required to develop the Web GUI and the Web API for this execution system. Your Web GUI should communicate with your Web API for all data needed in your application and the connections to all other backend dependencies (if any).

See below the technological requirements for each of the component.

Web GUI

- Developed with Typescript, React, React-Redux and Bootstrap, with any other libraries and packages that you see appropriate.

Web API

- Developed with any technology that you are comfortable with.

GENERAL INSTRUCTIONS

Here are some general instructions on how to approach this exercise:

- **Clean code**
- It is not a must that you finish all the features requested. We do accept the submission of an incomplete implementation.
- You are encouraged to adopt a Test Driven Development (TDD) approach.
- You are encouraged to use a package manager to manage the dependencies in your components.
- All the pictures provided in this document are for your reference only, you may choose to implement any UI/UX you see fit.
- If there are any special cases that are not covered in this document, you may make your own assumptions and decisions on how those should be handled.

ASSESSMENT CRITERIA

Your implementation and test code will be assessed. The assessment will primarily but not exclusively base on:

- The design choices made regarding legibility, maintainability, refactorability and scalability;
- Test coverage;
- The time spent on implementation;
- Exception handling; and
- Special cases handling (if any).

SUBMISSION GUIDELINES

Reply to us via email when you are ready to submit your best implementation of the stock execution system.

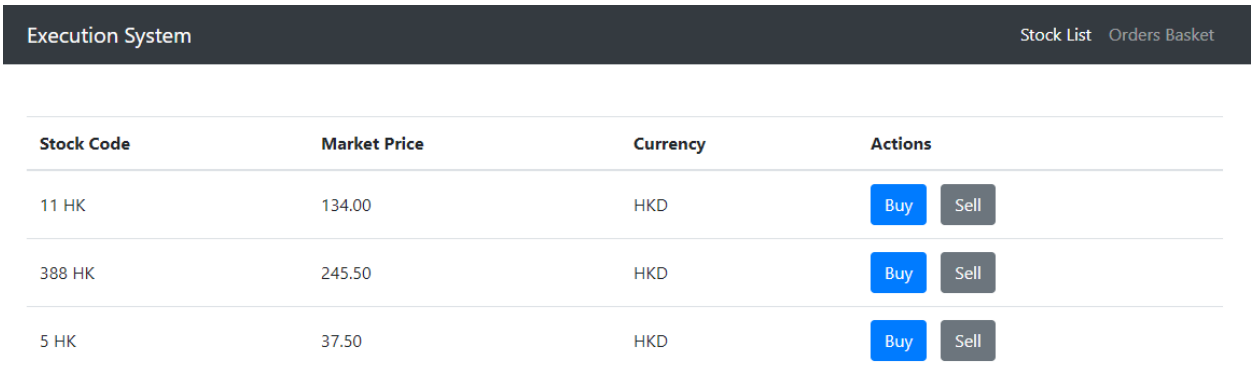
- **DO** upload your source code to a **private** repository on Github.com;
- **DO** add us to the list of the collaborator of your private repository created;
- **DO NOT** include your implementation's build files in your submission (e.g. **.exe*, **.dll*, **.class* files, etc.); and
- **DO NOT** include any dependency files in your submission.

SPECIFICATIONS

Web GUI

Stock List

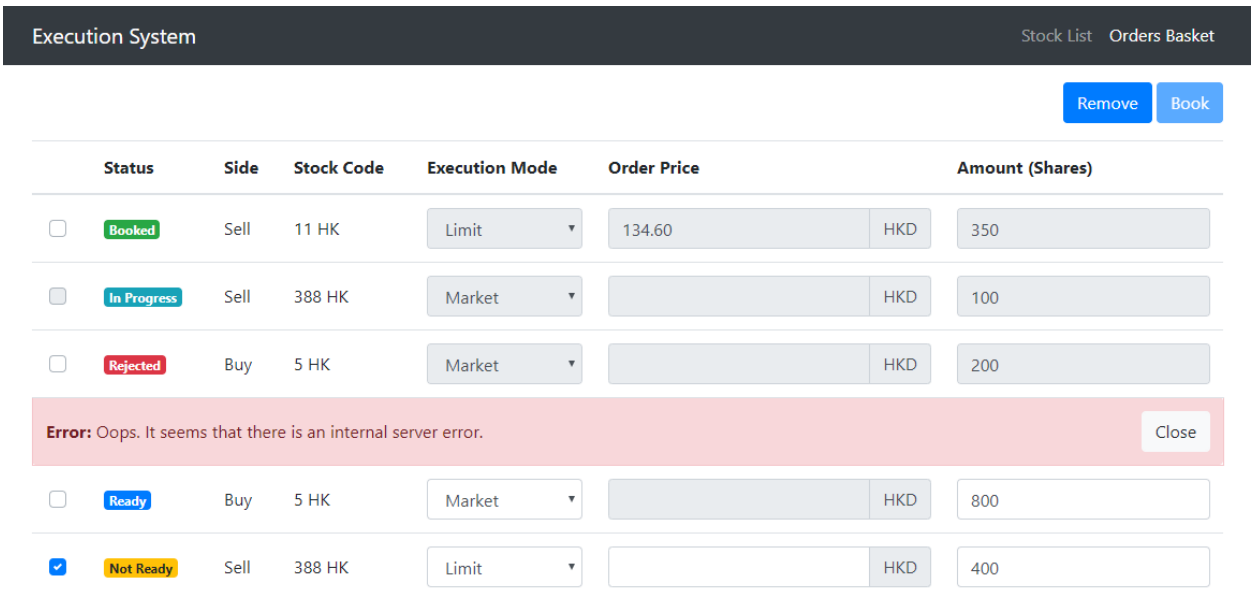
The is the first page of your web application. It displays a list of stocks with their current market level. The user can choose to perform a buy or sell action for a stock from this page.



- It should retrieve the list of stocks from the Web API.
- It should allow the users to view the details of the stocks.
- It should include the fields below for each of the stock:
  1. Stock Code
     - It should display the Bloomberg local ticker (*stock.bloombergTickerLocal*).
  2. Market Price
  3. Currency
  4. Actions
     - One *Buy* button to create a Buy order; and
     - One *Sell* button to create a Sell order.
     - A new Buy / Sell order should be added to the orders basket when the user clicks on the corresponding button.

Orders Basket

The second page of your web application is the orders basket – a basket of Buy / Sell orders that the user has added for execution. The user can modify some execution parameters of each order before submission.



- It should include the fields below:
  1. Selection Checkbox
     - A checkbox to select an order
     - Type: Checkbox
     - Should be disabled when:
       a. The order status is *In Progress*.
  2. Status
     - Indicates the status of an order
     - Type: Label, read only
     - Possible values: *Not Ready*, *Ready*, *In Progress*, *Booked* and *Rejected*
  3. Side
     - Indicates the Buy / Sell side of an order
     - Type: String, read only
     - Possible values: *Buy* and *Sell*
  4. Stock Code
     - The stock's Bloomberg local ticker (*stock.bloombergTickerLocal*)
     - Type: String, read only
  5. Execution Mode
     - Indicates the execution mode of the order

- Type: Select
- Available options: *Market* and *Limit*

6. Order Price
   - Type: Input group
   - Order Price
   - Indicates the order price of an order
   - Should be disabled when:
     a. The execution mode is *Market*; or
     b. The order is sent for execution; or
     c. There is an error with the order's last execution.
   - Features:
     a. Should display an empty value before execution when the execution mode is *Market*;
     b. Should not be disabled before execution when the execution mode is *Limit*; and
     c. Should display the order price returned by the Web API's order execution endpoint after execution.
7. Currency
   - Indicates the currency of the order
   - Type: String, read only
   - Example values: *HKD*, *USD*, *AUD*, *EUR*, *SGD*, etc.
8. Amount (Shares)
   - Indicates the number of shares to be executed in the order
   - Type: Number
   - Should be disabled when:
     a. The order is sent for execution; or
     b. There is an error with the order's last execution.

- It should include two actions buttons:

1. Remove
   - To remove the selected order(s) from the orders basket
   - When being clicked:
     a. Should remove the selected order(s) from the orders basket.
   - Should be disabled when:
     a. No order is selected.
2. Book
   - To book (execute) the selected order(s) in the orders basket
   - When being clicked:
     a. Should unselect the selected order(s); and
     b. Should send a request to the Web API to execute the orders; and

    c. Should change the status of the selected order(s) to *In Progress*.
  - Should be disabled when:
    a. No order is selected;
    b. Any of the selected order(s) is/are with the status of *Not Ready* or *Booked*.
- It should handle the parameter validation for each order to switch between the *Not Ready* and *Ready* status.
  - An order is said to be *Ready* when it is valid, *Not Ready* otherwise.
  - An order is valid when:
    a. Side is not null nor empty;
    b. Stock code is not null nor empty;
    c. Execution mode is selected;
    d. Order price is any decimal number greater than 0 (only when execution mode is *Limit*);
    e. Currency is not null nor empty;
    f. Amount is any integer greater than 0.
- It should update an order's status after receiving the response form the Web API's order execution endpoint.
  - When there is a successful response, should update order status to *Booked*.
  - When there is an error response, should update order status to *Error*.
  - It should display the error message(s) associated with an order from the Web API's order execution endpoint if there is any.

Web API

- Your API should be stateless.
- Your API should be a REST API.
- There is no strict requirement on how the API contract should be. You may implement the interfaces in any way you see appropriate.
- Your API should serve the features below:
  1. To retrieve all stocks available
     - A list of dummy stock data, *stocks.json*, is attached with this document.
     - Your API should return the data available in the dummy stock data file.
  2. To execute an order
     - Your API should take these request parameters:
       a. Sell / Buy side of your order;
       b. Stock ID;
       c. Stock code;
       d. Execution mode;
       e. Order price (only when the execution mode is Limit); and
       f. Amount (the number of shares).

- Your API should implement these response cases:
  a. Always throw a *500 Internal Server Error* when executing an order with the stock code *5 HK*;
  b. Always throw a *504 Gateway Timeout* when executing an order with the stock code *11 HK*;
  c. Always reject the request when executing an order with the stock code *388 HK*;
  d. Always return a successful response indicating the order has been booked for all other stock codes.
- Your successful response should include:
  a. The order price of the order executed
    - When the execution mode is *Market*, should return a random positive decimal number.
    - When the execution mode is *Limit*, should return the order price submitted to the API in the request.

README.md

You should attach a *README.md* file for each of your components. We expect at least two README.md files for your Web GUI and Web API, one each. If you see fit to implement some extra components for the stock execution system, you should include a *README.md* file in each of these extra components. Each of your *README.md* file should include, but not limited to:

1. Reason behind your choice of technology (programming languages, libraries, packages, etc.);
2. How to package (build) your component;
3. How to start your component;
4. Assumptions (if any) that you have made in your implementation;
5. Special cases handling (if any); and
6. Purpose / functionality of the component, if it is an extra component.

BONUS POINTS

You may choose to implement the optional features below to score some bonus points for your submission:

1. Pagination in the stock list;
2. A responsive design and implementation of your Web GUI; and
3. Any UI/UX enhancement that you see fit; and
4. Authentication between your Web GUI and Web API.