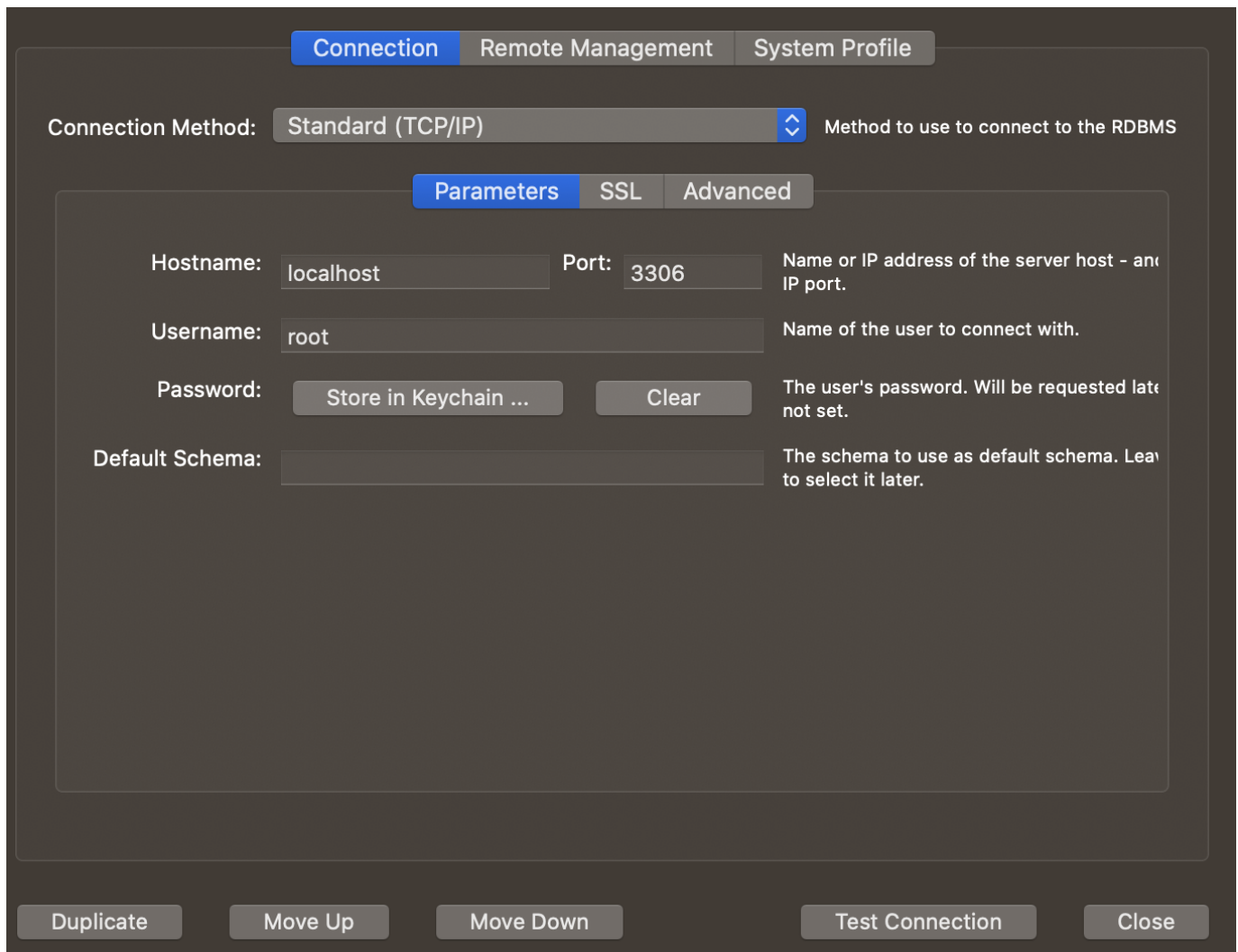


Instruction

Before you start this module, you need a DB running with a schema named "HaruDB"
For those of you who don't how to set up DB, simply run this command

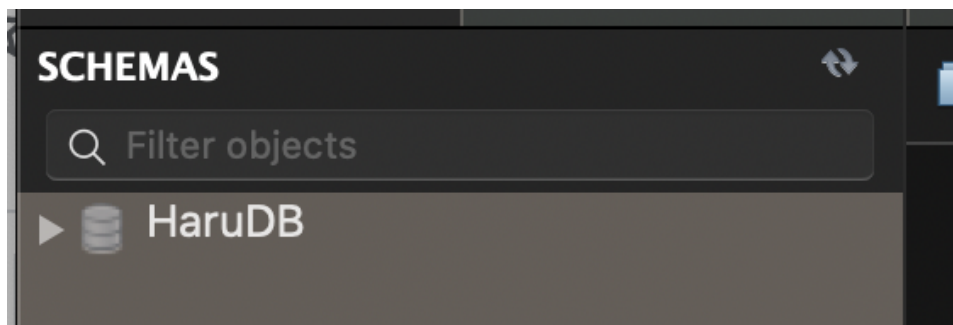
```
sudo docker run -d --name haruDB -p 3377:3306 -v /opt/datadir:/var/lib/mariadb -e  
MYSQL_ROOT_PASSWORD=BillionDollar12! mariadb:10.2
```



The screenshot shows the MySQL Workbench Connection dialog box. The 'Connection' tab is selected. The 'Connection Method' is set to 'Standard (TCP/IP)'. Below this, the 'Parameters' sub-tab is active. The fields are filled as follows: Hostname: localhost, Port: 3306, Username: root, Password: (with 'Store in Keychain ...' and 'Clear' buttons), and Default Schema: (empty). To the right of each field is a descriptive text. At the bottom of the dialog are buttons for 'Duplicate', 'Move Up', 'Move Down', 'Test Connection', and 'Close'.

Field	Value	Description
Connection Method	Standard (TCP/IP)	Method to use to connect to the RDBMS
Hostname	localhost	Name or IP address of the server host - and IP port.
Port	3306	
Username	root	Name of the user to connect with.
Password	[Buttons: Store in Keychain ..., Clear]	The user's password. Will be requested later if not set.
Default Schema		The schema to use as default schema. Leave empty to select it later.

The command above will download and run Maria DB's image with a port listening on 3377
Once the Database starts running, you need to create a schema called "HaruDB"
You may ask your fellow developer who sits next to you for help



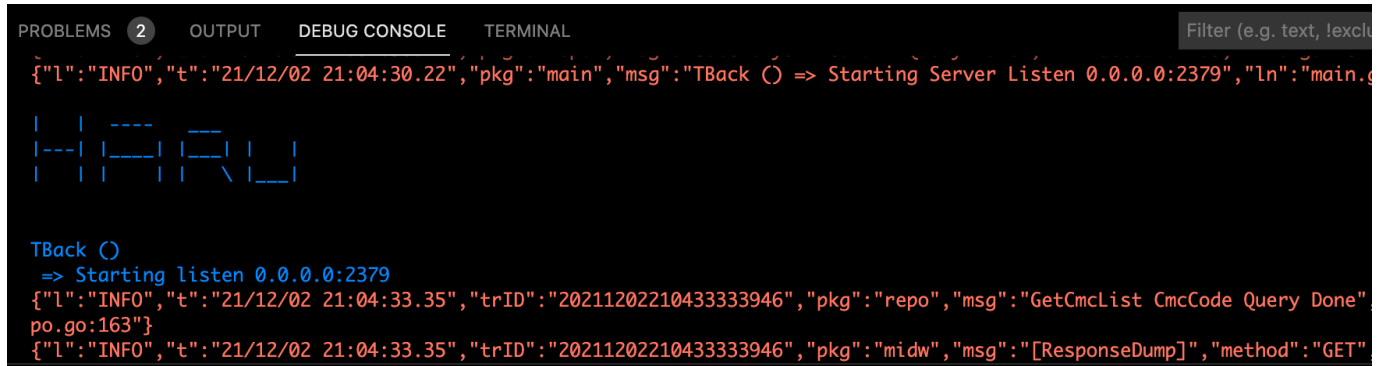
Once DB is completely set up, open TBack's module and run these commands:

dep init

dep ensure

Press F5 to run the module (I'm using Visual Studio Code)

Once you see "HARU" on the Debug Console, the backend is running successfully



The screenshot shows the Visual Studio Code interface with the 'DEBUG CONSOLE' tab active. It displays several log messages in JSON format. The first message indicates the server is starting and listening on 0.0.0.0:2379. Subsequent messages show a successful database query for CMC data and a response dump. A small ASCII art logo is also visible in the console output.

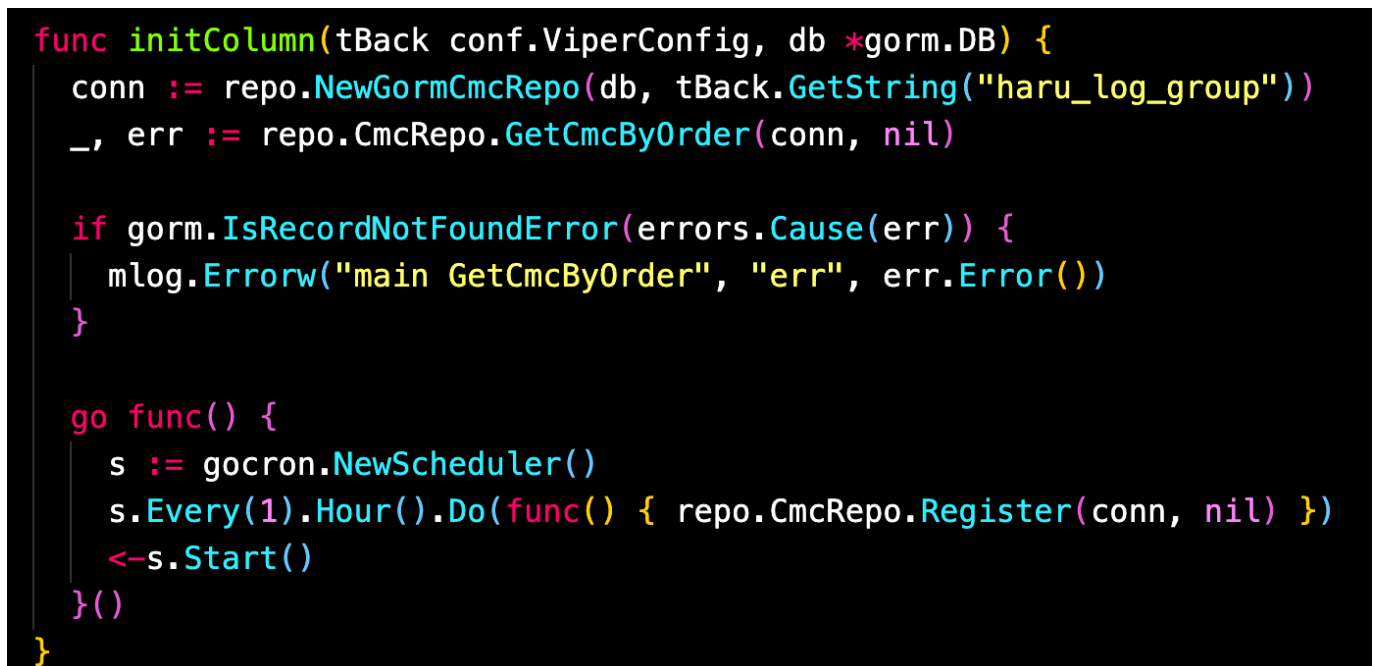
```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, !excl)
{"l":"INFO","t":"21/12/02 21:04:30.22","pkg":"main","msg":"TBack () => Starting Server Listen 0.0.0.0:2379","ln":"main.g
| | | | | | |
|---| |---| |---| |
| | | | | \ |---|

TBack ()
=> Starting listen 0.0.0.0:2379
{"l":"INFO","t":"21/12/02 21:04:33.35","trID":"20211202210433333946","pkg":"repo","msg":"GetCmcList CmcCode Query Done"
po.go:163"}
{"l":"INFO","t":"21/12/02 21:04:33.35","trID":"20211202210433333946","pkg":"midw","msg":"[ResponseDump]","method":"GET"
```

Go program starts from main.go

The initColumn() method checks whether the Table is empty and if it's empty, it fetches the first data from CoinMarketCap and stores the current market prices of ATOM, BTC, ETH, LTC, and XRP in the database.

The "go func()" schedules a batch every hour and stores the next current market prices of the coins. The logic runs in an asynchronous method.



The screenshot shows a Go code snippet. The initColumn function initializes a database connection and checks for existing data. If no data is found, it logs an error. A go func() block schedules a task using gocron to run the repo.CmcRepo.Register function every hour.

```
func initColumn(tBack conf.ViperConfig, db *gorm.DB) {
    conn := repo.NewGormCmcRepo(db, tBack.GetString("haru_log_group"))
    _, err := repo.CmcRepo.GetCmcByOrder(conn, nil)

    if gorm.IsRecordNotFoundError(errors.Cause(err)) {
        mlog.Errorw("main GetCmcByOrder", "err", err.Error())
    }

    go func() {
        s := gocron.NewScheduler()
        s.Every(1).Hour().Do(func() { repo.CmcRepo.Register(conn, nil) })
        <-s.Start()
    }()
}
```

Once the api is called, the request passes through controller.go

```
main.go M  controller.go 1, M  cmcCtl.go M
pi > controller > controller.go > tbackHTTPHandler > tbackSvc
84 func newCmcHTTPHandler(eg *echo.Group, scd service.CmcSvc
85     handler := cmcHTTPHandler{
86         cmcSvc:  scd,
87         jwtMode: jm,
88     }
89
90     eg.GET("/symbol", handler.getCmcListBySymbol)
91     eg.GET("/conversion", handler.getConversion)
92 }
```

getCmcListBySymbol and getConversion are the two apis that can be requested from the frontend.

```
func (a cmcHTTPHandler) getCmcListBySymbol(c echo.Context) error {
    trID := c.Response().Header().Get(echo.HeaderXRequestID)
    ctx := c.Request().Context()
    if ctx == nil {
        ctx = context.Background()
    }

    symbol := new(model.GetBySymbolHTTPRequest)

    if err := c.Bind(symbol); err != nil {
        mlog.Warnw("cardCtrl getCmcListBySymbol Bind error", "trID", trID, "err", err.Error())
        return c.JSON(http.StatusBadRequest, model.CommonHTTPResponse{
            ResultCode: "1001",
            ResultMsg:  fmt.Sprintf("Invalid Parameter err[%s]", err.Error()),
            TrID:       trID,
        })
    }

    result, err := a.cmcSvc.GetCmcListBySymbol(ctx, trID, symbol)
    if err != nil {
        mlog.Errorw("cmcCtrl getCmcListBySymbol cmcSvc GetCmcList", "trID", trID, "err", err.Error())
        return c.JSON(http.StatusInternalServerError, model.CommonHTTPResponse{
            ResultCode: "1101",
            ResultMsg:  "Internal Server Error",
            TrID:       trID,
        })
    }
}
```

when getCmcListBySymbol is an api to fetch the stored market prices of the coins. Once the function is called, it binds the parameters with the premade struct and passes through the model to service then to repository. When data is transferred from one file to another, it leaves a unique trace called the transaction ID to later retrace once the errors occur. Every action leaves a footprint of logs printed on the console with exception handlings.

```

ain.go M    controller.go 1, M    cmcCtl.go M    gormCmcRepo.go 5 X
sitory > gormCmcRepo.go > (gormCmcRepo).GetCmcListBySymbol
1  }
2
3  func (a gormCmcRepo) GetCmcListBySymbol(ctx context.Context, trID string, symbol *model.GetBySymbolHTTPRequest) ([]*model.Cmc, error) {
4      card := []*model.Cmc{}
5      offset := 0
6      limit := 1000
7      orderby := "desc"
8      query := a.conn.Offset(offset).Limit(limit).
9          Select("*").
10         Where("deleted_at IS NULL")
11
12     if symbol.Symbol != "" {
13         query = query.Where("symbol = ?", symbol.Symbol)
14     }
15
16     if symbol.FromDate != "" && symbol.ToDate != "" {
17         query = query.Where("last_updated between ? and ?", symbol.FromDate, symbol.ToDate)
18     } else if symbol.FromDate != "" && symbol.ToDate == "" {
19         query = query.Where("last_updated >= ?", symbol.FromDate)
20     } else if symbol.FromDate == "" && symbol.ToDate != "" {
21         query = query.Where("last_updated <= ?", symbol.ToDate)
22     }
23

```

The query is built with the ORM called gorm in order to take advantage of abstracting the object relational mapping based on different parameter settings. Once the query is running, it maps with the struct model and sends the response back in json format.

Basically, it'll fetch this data:

seq	created_at	updated_at	deleted_at	cmc_id	symbol	price	last_updated
1	2021-12-02 10:25:17	0000-00-00 00:00:00	NULL	74276a84-4eee-483f-819d-95ae0da00f46	BTC	57031.85186885341	2021-12-02 10:24:06
2	2021-12-02 10:25:17	0000-00-00 00:00:00	NULL	8f97f3ff-9a3d-4f48-8447-bc5902dc6ac8	ETH	4573.116780199948	2021-12-02 10:24:06
3	2021-12-02 10:25:17	0000-00-00 00:00:00	NULL	c0c2e48-ffc-4322-a188-914e9e023fb4	ATOM	26.434249510088097	2021-12-02 10:23:33
4	2021-12-02 10:25:17	0000-00-00 00:00:00	NULL	c8f7cc21-95d2-4dfc-81b3-130ed0495943	LTC	207.81234588329846	2021-12-02 10:24:06
5	2021-12-02 10:25:17	0000-00-00 00:00:00	NULL	41042fce-312d-4c90-8db6-5cd9b38d32b2	XRP	0.9815958725761763	2021-12-02 10:23:05
6	2021-12-02 14:04:15	0000-00-00 00:00:00	NULL	1ce1a278-f855-4489-b5c0-0377f2236862	BTC	56637.23695695205	2021-12-02 14:03:05
7	2021-12-02 14:04:15	0000-00-00 00:00:00	NULL	a86a1d06-5a12-4e91-845c-8c3357d97214	ETH	4515.414744115035	2021-12-02 14:02:10
8	2021-12-02 14:04:15	0000-00-00 00:00:00	NULL	0e392ee2-4537-4746-a7f3-e8bdc0dd99e3	ATOM	26.278133844979187	2021-12-02 14:02:40
9	2021-12-02 14:04:15	0000-00-00 00:00:00	NULL	b08e11b2-ba5f-4e6e-8a1c-46f7fdb70665	LTC	203.35133363849616	2021-12-02 14:03:07
10	2021-12-02 14:04:15	0000-00-00 00:00:00	NULL	c9891b6d-ba0e-476b-9a23-b44a56f58f06	XRP	0.9718992031816164	2021-12-02 14:02:05
11	2021-12-02 15:04:14	0000-00-00 00:00:00	NULL	57649068-700d-4114-95c2-ef452436e767	BTC	56835.894591870914	2021-12-02 15:02:06
12	2021-12-02 15:04:14	0000-00-00 00:00:00	NULL	423db4d7-8d45-499f-8ae2-4264a3273266	ETH	4521.09884364809	2021-12-02 15:02:09
13	2021-12-02 15:04:14	0000-00-00 00:00:00	NULL	7aa339b4-51a2-458d-bbea-d86d708d10c8	ATOM	26.355525285878972	2021-12-02 15:02:41
14	2021-12-02 15:04:14	0000-00-00 00:00:00	NULL	8052df5e-264f-4d62-9d48-759942b742d2	LTC	204.15390211344663	2021-12-02 15:02:07
15	2021-12-02 15:04:14	0000-00-00 00:00:00	NULL	658a9430-48da-4e0d-bc52-e8e1c3a4c3e3	XRP	0.9733400681853732	2021-12-02 15:02:12
16	2021-12-02 17:01:32	0000-00-00 00:00:00	NULL	a354241f-6711-4907-ad11-4ff43628bf81	BTC	56869.328725741725	2021-12-02 17:00:02
17	2021-12-02 17:01:32	0000-00-00 00:00:00	NULL	c7e3f4f9-40d1-42eb-80f3-a2158badb3e4	ETH	4565.0590575440465	2021-12-02 17:00:02
18	2021-12-02 17:01:32	0000-00-00 00:00:00	NULL	efa069c3-f4a0-4abf-b2b0-eb7eedfa15be	ATOM	26.426618953643096	2021-12-02 17:00:08
19	2021-12-02 17:01:32	0000-00-00 00:00:00	NULL	7b1cc4c2-fb9e-4c31-aedf-a1a6bffc387f4	LTC	204.1656838912257	2021-12-02 17:00:02
20	2021-12-02 17:01:32	0000-00-00 00:00:00	NULL	b2b3c9db-6fee-427d-a881-a8451f927a61	XRP	0.976047469219234	2021-12-02 17:00:03
21	2021-12-02 18:01:31	0000-00-00 00:00:00	NULL	0758611a-0a71-42f4-b34d-a84d00957d29	BTC	57005.97811600002	2021-12-02 18:00:03
22	2021-12-02 18:01:31	0000-00-00 00:00:00	NULL	97fcaf1-fd0c-4b94-b7a7-21c3c2af22fe	ETH	4590.476587071539	2021-12-02 18:00:02
23	2021-12-02 18:01:31	0000-00-00 00:00:00	NULL	4edaefea-d2af-4c39-b068-798971427bb9	ATOM	26.53924416908296	2021-12-02 18:00:08
24	2021-12-02 18:01:31	0000-00-00 00:00:00	NULL	96eaa6c6-b3d0-4b02-978e-a04f2bd7ee0c	LTC	205.33268418246263	2021-12-02 18:00:02
25	2021-12-02 18:01:31	0000-00-00 00:00:00	NULL	b8fb9b1a-0e4f-4f94-919f-395ed5a3c9a5	XRP	0.9785097183585425	2021-12-02 18:00:03

To this json

```
{
  "resultCode": "0000",
  "resultMsg": "Get CmcListBySymbol Fetched",
  "trID": "20211202214403954357",
  "cmc": [
    {
      "Seq": 21,
      "CreatedAt": "2021-12-02T18:01:31+09:00",
      "UpdatedAt": "0001-01-01T00:00:00Z",
      "DeletedAt": null,
      "cmcID": "0758611a-0a71-42f4-b34d-a84d00957d29",
      "symbol": "BTC",
      "price": 57005.97811600002,
      "last_updated": "2021-12-02T18:00:03+09:00"
    },
    {
      "Seq": 16,
      "CreatedAt": "2021-12-02T17:01:32+09:00",
      "UpdatedAt": "0001-01-01T00:00:00Z",
      "DeletedAt": null,
      "cmcID": "a354241f-6711-4907-ad11-4ff43628bf81",
      "symbol": "BTC",
      "price": 56869.328725741725,
      "last_updated": "2021-12-02T17:00:02+09:00"
    },
    {
      "Seq": 11,
      "CreatedAt": "2021-12-02T15:04:14+09:00",
      "UpdatedAt": "0001-01-01T00:00:00Z",
      "DeletedAt": null,
      "cmcID": "57649068-700d-4114-95c2-ef452436e767",
      "symbol": "BTC",
      "price": 56835.894591870914,
      "last_updated": "2021-12-02T15:02:06+09:00"
    }
  ]
}
```

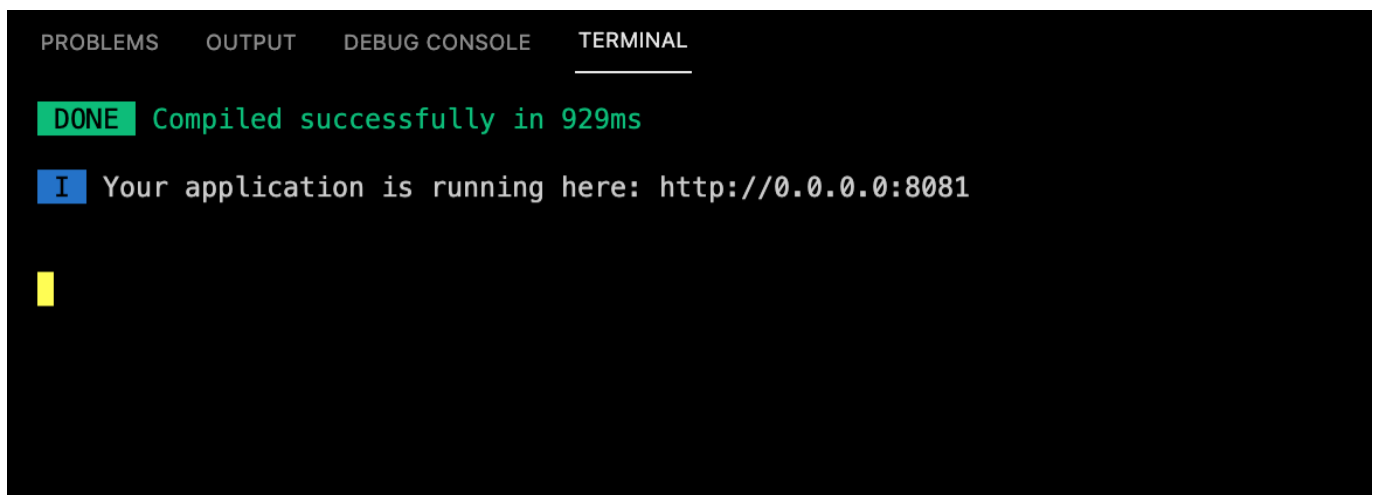
and end up as a front end rendering.

Sequence	symbol	(\$)price	time
25	XRP	0.9785097183585425	2021-12-02, 6:00:03 pm
24	LTC	205.33268418246263	2021-12-02, 6:00:02 pm
23	ATOM	26.53924416908296	2021-12-02, 6:00:08 pm
22	ETH	4590.476587071539	2021-12-02, 6:00:02 pm
21	BTC	57005.97811600002	2021-12-02, 6:00:03 pm
20	XRP	0.976047469219234	2021-12-02, 5:00:03 pm
19	LTC	204.1656838912257	2021-12-02, 5:00:02 pm
18	ATOM	26.426618953643096	2021-12-02, 5:00:08 pm
17	ETH	4565.0590575440465	2021-12-02, 5:00:02 pm
16	BTC	56869.328725741725	2021-12-02, 5:00:02 pm
15	XRP	0.9733400681853732	2021-12-02, 3:02:12 pm
14	LTC	204.15390211344663	2021-12-02, 3:02:07 pm
13	ATOM	26.355525285878972	2021-12-02, 3:02:41 pm

Frontend

cd into Haru directory and run these commands:

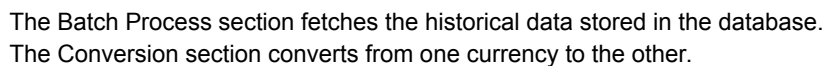
```
npm install  
npm run dev
```



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
  
[DONE] Compiled successfully in 929ms  
  
[I] Your application is running here: http://0.0.0.0:8081  
  
█
```

Once you see this screen, your frontend is running successfully.

You'll end up with a screen like this:



Batch Process

BTC



In this example, I have inputted, BTC, from December 02 to December 03
You may input a specific time but for convenience, I added a date picker.

Start Date

End Date

Q 2021-12-02 10:11:12

2021-12-15 1

2021

Thu, Dec 2

< December 2021 >

S M T W T F S

1 2 3 4

5 6 7 8 9 10 11

12 13 14 15 16 17 18

19 20 21 22 23 24 25

26 27 28 29 30 31

Press the search button and it'll return Bitcoin's historical data collected during the time frame.

Batch Process

Batch Process			
BTC		Start Date Q 2021-12-02 10:11:12	End Date 2021-12-03 18:11:12
Q SEARCH			
Sequence	symbol	(\$)price	time
21	BTC	57005.97811600002	2021-12-02, 6:00:03 pm
16	BTC	56869.328725741725	2021-12-02, 5:00:02 pm
11	BTC	56835.894591870914	2021-12-02, 3:02:06 pm
6	BTC	56637.23695695205	2021-12-02, 2:03:05 pm
1	BTC	57031.85186885341	2021-12-02, 10:24:06 am
< 1 >			

Conversion

From	To	(\$) Result
BTC	XRP	
LTC	USD	
LUNA	ETH	
BTC	USD	

+

CONVERT

To use the conversion calculator, simply press Convert.

Conversion

From	To	(\$) Result
BTC	XRP	57989.83300957722
LTC	USD	
LUNA	ETH	
BTC	USD	

+

Batch Process

BTC

Start Date
2021-12-02 10:11:12

End Date
2021-12-03 18:11:12

SEARCH

Sequence	symbol	(\$) price	time
21	BTC	57005.97811600002	2021-12-02, 6:00:03 pm
16	BTC	56869.328725741725	2021-12-02, 5:00:02 pm
11	BTC	56835.894591870914	2021-12-02, 3:02:06 pm
6		695205	2021-12-02, 2:03:05 pm
1		885341	2021-12-02, 10:24:06 am

Fetching the result ...

25%

The convert button will request the getConversion api. The api will request another api to CoinMarketcap and convert the prices, return the responses, and render back on the screen.

You may also add another conversion calculation by pressing the plus sign on the left hand corner.

For this example, I will convert from XRP to USD.

From

XRP

CANCEL SAVE

input "From" textfield column

To

USD

CANCEL SAVE

input "To" textfield column

XRP

USD



CONVERT

click convert

Conversion

From	To	(\$) Result
BTC	XRP	58084.59478569269
LTC	USD	205.17000915715204
LUNA	ETH	0.014298994870635998
BTC	USD	56632.24676813268
XRP	USD	0.9749959860627667



Batch Process

BTC

Start Date
2021-12-02 10:11:12

End Date
2021-12-03 18:11:12

Sequence	symbol	(\$) price
21	BTC	57005.97811600002
16	BTC	56869.328725741725
11	BTC	56835.894591870914
6		695205
1		885341

Fetching the result ...

20%

Wait for the server to fetch the data.

XRP

USD

0.9749959860627667



CONVERT

The result.