
ASE4156 Documentation

Release latest

Nigel Schuster

Dec 15, 2017

Contents

1	Trading	3
2	Authentication	5
3	Stocks	7
4	GraphQL	9
4.1	Trading	9
4.2	Authentication	10
4.3	Stocks	12
	Python Module Index	17

Contents:

CHAPTER 1

Trading

Models here represents any interaction between a user and stocks

```
class trading.models.TradeBucket (*args, **kwargs)
    Bases: django.db.models.base.Model
```

Same as trade but for buckets

```
current_value()
    The value of the trade on the specific date
```

```
class trading.models.TradeStock (*args, **kwargs)
    Bases: django.db.models.base.Model
```

A Trade represents a single exchange of a stock for money

```
current_value()
    Get value calculates the total value of the trade respecting the date
```

```
class trading.models.TradingAccount (*args, **kwargs)
    Bases: django.db.models.base.Model
```

A TradingAccount is owned by a user, we associate stock trades with it.

```
available_buckets (bkt)
    Find the available buckets that have quantity > 0
```

```
available_cash (update=True)
    Returns the available cash for the trading account
```

```
available_stocks (stk)
    Find available stock
```

```
has_enough_bucket (bucket, quantity_bucket)
    Check if you have enough bucket to make a trade
```

```
has_enough_cash (trade_value)
    Check if you have enough cash to make a trade
```

has_enough_stock (*stock, quantity_stock*)

Check if you have enough stock to trade

holding_value ()

Calculates the value of all equity held by the user

total_value ()

Total value of the trading account

trade_bucket (*bucket, quantity*)

Creates a new trade for the bucket and this account

trade_stock (*stock, quantity*)

Trades a stock for the account

trading_balance ()

The stock values from account

CHAPTER 2

Authentication

Models keeps track of all the persistent data around the user profile

```
class authentication.models.Profile(*args, **kwargs)
```

Bases: `django.db.models.base.Model`

Profile is an extension of `django.contrib.auth.models.User`, that allows us to store additional values per User.

```
default_acc()
```

This method retrieves the default account for the profile. If none exists, a new one will be created with the name 'default'.

Returns The default `trading.models.TradingAccount`.

```
class authentication.models.UserBank(*args, **kwargs)
```

Bases: `django.db.models.base.Model`

The UserBank wraps a connection to Plaid. It stores the access token for the User. At the same time it also caches past queries to reduce initial load time.

```
account_name(update=True)
```

Returns the account name

```
current_balance(update=True)
```

Returns the latest balance for the bank account. If update is set, then the balance will be synced with the original bank account.

Parameters `update (bool)` – Whether to sync with the remote bank account.

Returns float of the current balance for the account

```
expenditure(days=30, update=True)
```

Returns the expenditures in the given timespan

```
historical_data(*args, **kwargs)
```

Fetches the historical data (see `authentication.plaid_wrapper.PlaidAPI.historical_data()`)

Returns list of tuples for the historical data.

income (*days=30, update=True*)

Returns the income in the given timespan

plaid()

This method instantiates a new *authentication.plaid_wrapper.PlaidAPI* with the stored access token.

Returns *authentication.plaid_wrapper.PlaidAPI* for the User.

`authentication.models.create_user_profile(instance, created, **_)`

This method will be called every time a `django.contrib.auth.models.User` is saved. It will create a *authentication.models.Profile* to associate with the user.

Parameters

- **instance** (`django.contrib.auth.models.User`) – The User instance that was saved.
- **created** – True if the instance was just created.
- **type** – bool

`authentication.models.save_user_profile(instance, **_)`

This method ensures that the *authentication.models.Profile* is kept in sync with the User (`django.contrib.auth.models.User`)

Parameters **instance** (`django.contrib.auth.models.User`) – The User instance that was saved.

CHAPTER 3

Stocks

Models keeps track of all the persistent data around stocks

```
class stocks.models.DailyStockQuote (*args, **kwargs)
    Bases: django.db.models.base.Model
```

DailyStockQuote is one day in the performance of a stock, for example 2nd July GOOGL value is 281.31\$

```
class stocks.models.InvestmentBucket (*args, **kwargs)
    Bases: django.db.models.base.Model
```

An investment bucket represents a collection of stocks to invest in

```
static accessible_buckets (profile)
    Finds all buckets that the user could view
```

```
add_attribute (text, is_good=True)
    Adds an attribute to an investment bucket
```

```
change_config (new_config)
    Changes the configuration of the investment bucket to new_config
```

```
static create_new_bucket (name, public, owner, available=1000.0)
    Creates a new InvestmentBucket
```

```
get_stock_configs (date=None)
    Get all associated configs
```

```
historical (count=None, skip=None)
    Fetches the historical value of the bucket.
```

```
value_on (date=None)
    The value of the bucket on a specific day
```

```
class stocks.models.InvestmentBucketDescription (*args, **kwargs)
    Bases: django.db.models.base.Model
```

An investment bucket represents a collection of stocks to invest in

change_description (*text*)

Changes the description to the given text

class `stocks.models.InvestmentStockConfiguration` (**args, **kwargs*)

Bases: `django.db.models.base.Model`

Represents the configuration of how much of a stock to invest for a bucket

value_on (*date=None*)

Returns the current value of the stock configuration

class `stocks.models.Stock` (**args, **kwargs*)

Bases: `django.db.models.base.Model`

Stock represents a single stock. For example GOOGL

static create_new_stock (*ticker, name*)

Creates a new stock

static find_stock (*text, first=None*)

Finds the stocks that contain >text<

latest_quote (*date=None*)

Returns the latest quote for the stock

quote_in_range (*start=None, end=None*)

Returns a list of daily stock quotes in the given timerange

trades_for_profile (*profile*)

Returns all trades the user made with this stock

`stocks.models.pre_save_any` (*sender, instance, *_args, **_kwargs*)

Ensures that all constraints are met

4.1 Trading

GraphQL definitions for the Trading App

```
class trading.graphql.AddTrade (*args, **kwargs)
    Bases: graphene.types.mutation.Mutation

    AddTrade creates a new Trade for the user and stock

    class Arguments
        Bases: object

        Arguments to create a trade. Right now it's only ticker and quantity.

    static mutate (_self, info, id_value, quantity, account_name, **_args)
        Creates a Trade and saves it to the DB

class trading.graphql.GInvestmentBucketTrade (*args, **kwargs)
    Bases: graphene_django.types.DjangoObjectType

    Exposing the whole Trade object to GraphQL

    static resolve_value (data, _info, **_args)
        Returns the value of a trade (see the model)

class trading.graphql.GTrade (*args, **kwargs)
    Bases: graphene_django.types.DjangoObjectType

    Exposing the whole Trade object to GraphQL

    static resolve_value (data, _info, **_args)
        Returns the value of a trade (see the model)

class trading.graphql.GTradingAccount (*args, **kwargs)
    Bases: graphene_django.types.DjangoObjectType

    Exposing the whole TradingAccount to GraphQL
```

```
static resolve_available_cash (data, _info, **_args)
    Returns the amount of cash the user has available

static resolve_total_value (data, _info, **_args)
    Returns the total value that the account currently holds

class trading.graphql.InvestBucket (*args, **kwargs)
    Bases: graphene.types.mutation.Mutation

    Invests into the bucket

class Arguments
    Bases: object

    We need quantity, account id and bucket id

static mutate (_self, info, quantity, trading_acc_id, bucket_id, **_args)
    Creates the trade

class trading.graphql.Query
    Bases: object

    We don't want to have any root queries here
```

4.2 Authentication

GraphQL definitions for the Authentication App

```
class authentication.graphql.AddTradingAccount (*args, **kwargs)
    Bases: graphene.types.mutation.Mutation

    AddTradingAccount creates a new trading.models.TradingAccount for the user.

class Arguments
    Bases: object

    Arguments to create a trading.models.TradingAccount. Right now we only need the name.

static mutate (_self, info, name, **_args)
    Creates a new trading.models.TradingAccount for the user and returns it.
```

Parameters

- **info** – Information about the request / user.
- **name** (*str*) – Name of the new trading account.

```
class authentication.graphql.GProfile (*args, **kwargs)
    Bases: graphene_django.types.DjangoObjectType
```

This is the GraphQL representation of a *authentication.models.Profile*. This is more of a publically accessible object. Even though we won't expose everything, this object allows us to add more fields to the user object.

```
static resolve_invest_suggestions (_data, info, **_args)
    Returns a list of buckets that the User can invest in. (see stocks.models.InvestmentBucket.
    available_buckets())
```

Parameters **info** (*Graphene Request Info.*) – Information about the user to check which recommendations are best for the user.

Returns *django.db.models.query.QuerySet* of *stocks.models.InvestmentBucket*

```
static resolve_selected_acc (data, _info, **_args)
```

Returns the current account the user has selected. Right now it just calls the default account of the profile. (see `authentication.models.Profile.default_acc()`)

Returns `django.db.models.query.QuerySet` of `trading.models.TradingAccount`

```
static resolve_stock_find (_self, _info, text, first=None, **_args)
```

Finds a stock given a case insensitive name. (see `stocks.models.Stock.find_stock()`)

Parameters

- **text** – The text the user want to search for.
- **first** – The maximum number of results to return

Returns `django.db.models.query.QuerySet` of `stocks.stocks.Stock`

```
class authentication.graphql.GUser (*args, **kwargs)
```

Bases: `graphene_django.types.DjangoObjectType`

This is the GraphQL representation of a `django.contrib.auth.models.User`. This should *only* be accessible for the user himself.

```
class authentication.graphql.GUserBank (*args, **kwargs)
```

Bases: `graphene_django.types.DjangoObjectType`

GraphQL wrapper around the `authentication.models.UserBank` model. This should *only* be accessible to the user.

```
static resolve_balance (data, _info, **_args)
```

Calls `authentication.models.UserBank.current_balance()` on data.

Parameters **data** (`authentication.models.UserBank`) – The Userbank we want to extract the balance from.

Returns The current balance of that the user has.

```
static resolve_balance_date (_data, _info)
```

Date of the balance

```
static resolve_history (data, _info, start, **_args)
```

This method returns the account history for a user. This is, how much value the bank account historically had. (see `authentication.models.UserBank.historical_data()`)

Parameters

- **data** (`authentication.models.UserBank`) – The bank we want to extract the history from.
- **start** (`str (YYYY-MM-dd)`) – The date with that the history should start. The query will return the history from start until today.

Returns `stocks.graphql.DataPoint` representing the history.

```
static resolve_income (data, _info, **_args)
```

Calls `authentication.models.UserBank.income()` on data.

Parameters **data** (`authentication.models.UserBank`) – The Userbank we want to extract the income from.

Returns The monthly income of the account.

```
static resolve_monthly_end (_data, _info)
```

End date for measuring the monthly income/expenditure

```
static resolve_monthly_start (_data, _info)
```

Start date for measuring the monthly income/expenditure

```
static resolve_name (data, _info, **_args)
```

Calls `authentication.models.UserBank.account_name()` on data.

Parameters `data` (`authentication.models.UserBank`) – The Userbank we want to get the account name.

Returns The account name of the bank.

```
static resolve_outcome (data, _info, **_args)
```

Calls `authentication.models.UserBank.expenditure()` on data.

Parameters `data` (`authentication.models.UserBank`) – The Userbank we want to extract the expenditures from.

Returns The monthly expenditure of the account.

```
class authentication.graphql.Query
```

Bases: `object`

The root of the viewer query. This is the base of building the user object with all of its data.

```
static resolve_viewer (_self, info, **_args)
```

The viewer represents the data for the user making the request.

Parameters `info` – information about the request with context

4.3 Stocks

GraphQL definitions for the Stocks App

```
class stocks.graphql.AddAttributeToInvestment (*args, **kwargs)
```

Bases: `graphene.types.mutation.Mutation`

Adds a description to an Investment Bucket and returns the bucket

```
class Arguments
```

Bases: `object`

We need the description and the bucket as input

```
static mutate (_self, info, desc, bucket_id, is_good, **_args)
```

Executes the mutation to add the attribute

```
class stocks.graphql.AddBucket (*args, **kwargs)
```

Bases: `graphene.types.mutation.Mutation`

Creates a new InvestmentBucket and returns the new bucket

```
class Arguments
```

Bases: `object`

We only need the name of the new bucket to create it

```
static mutate (_self, info, name, investment, public, **_args)
```

Creates a new InvestmentBucket and saves it to the DB

```
class stocks.graphql.AddStock (*args, **kwargs)
```

Bases: `graphene.types.mutation.Mutation`

AddStock creates a new Stock that is tracked

```

class Arguments
    Bases: object

    Arguments to create a stock. We only need the ticker.

    static mutate (_self, _info, ticker, name, **_args)
        Creates a Stock and saves it to the DB

class stocks.graphql.Config (id, quantity)
    Bases: tuple

    id
        Alias for field number 0

    quantity
        Alias for field number 1

class stocks.graphql.DataPoint (date, value)
    Bases: object

    Dummy class to represent a date / value DataPoint

class stocks.graphql.DeleteAttribute (*args, **kwargs)
    Bases: graphene.types.mutation.Mutation

    Deletes an attribute from a bucket

    class Arguments
        Bases: object

        We just need the ID to delete it

    static mutate (_self, info, id_value, **_args)
        Executes the mutation by deleting the attribute

class stocks.graphql.DeleteBucket (*args, **kwargs)
    Bases: graphene.types.mutation.Mutation

    Deletes an attribute from a bucket

    class Arguments
        Bases: object

        We just need the ID to delete it

    static mutate (_self, info, id_value, **_args)
        Executes the mutation by deleting the attribute

class stocks.graphql.EditAttribute (*args, **kwargs)
    Bases: graphene.types.mutation.Mutation

    Allows to edit an attribute description

    class Arguments
        Bases: object

        Description and ID for the mutation

    static mutate (_self, info, id_value, desc, **_args)
        Executes the mutation to change the attribute

class stocks.graphql.EditConfiguration (*args, **kwargs)
    Bases: graphene.types.mutation.Mutation

    Mutation to change the stock configuration of a bucket

```

```
class Arguments
```

```
    Bases: object
```

```
    As input we take the new configuration and the bucket id
```

```
    static mutate(_self, info, id_value, config, **_args)
```

```
        This performs the actual mutation by removing the old configuration and then writing the new one
```

```
class stocks.graphql.GDailyStockQuote(*args, **kwargs)
```

```
    Bases: graphene_django.types.DjangoObjectType
```

```
    GraphQL representation of a DailyStockQuote
```

```
class stocks.graphql.GDataPoint(*args, **kwargs)
```

```
    Bases: graphene.types.objecttype.ObjectType
```

```
    GraphQL definition of the DataPoint above
```

```
class stocks.graphql.GInvestmentBucket(*args, **kwargs)
```

```
    Bases: graphene_django.types.DjangoObjectType
```

```
    GraphQL representation of a InvestmentBucket
```

```
    static resolve_history(data, _info, count=None, skip=None, **_args)
```

```
        Returns the historic data for the bucket
```

```
    static resolve_is_owner(data, info, **_args)
```

```
        Returns whether the user owns the investment bucket
```

```
    static resolve_owned_amount(data, info, **_args)
```

```
        Returns how much of the bucket the user owns
```

```
    static resolve_stocks(data, _info, **_args)
```

```
        Returns the current stocks in the bucket
```

```
    static resolve_value(data, _info, **_args)
```

```
        The current value of the investment bucket
```

```
class stocks.graphql.GInvestmentBucketAttribute(*args, **kwargs)
```

```
    Bases: graphene_django.types.DjangoObjectType
```

```
    GraphQL representation of a InvestmentBucketDescription
```

```
class stocks.graphql.GInvestmentBucketConfigurationUpdate(*args, **kwargs)
```

```
    Bases: graphene.types.inputobjecttype.InputObjectType
```

```
    Represents one choice of stock for a bucket
```

```
class stocks.graphql.GInvestmentStockConfiguration(*args, **kwargs)
```

```
    Bases: graphene_django.types.DjangoObjectType
```

```
    GraphQL representation of a InvestmentStockConfiguration
```

```
class stocks.graphql.GStock(*args, **kwargs)
```

```
    Bases: graphene_django.types.DjangoObjectType
```

```
    GraphQL representation of a Stock
```

```
    static resolve_latest_quote(data, _info, **_args)
```

```
        Returns the most recent stock quote
```

```
    static resolve_quote_in_range(data, _info, start, end, **_args)
```

```
        Finds the stock quotes for the stock within a time range
```

```
    static resolve_trades(stock, info, **_args)
```

```
        We need to apply permission checks to trades
```

```
class stocks.graphql.Query
```

```
    Bases: object
```

We don't want to have any root queries here

```
    static resolve_invest_bucket (_self, info, id_value, *_args)
```

The viewer represents the current logged in user

a

`authentication.graphql`, [10](#)
`authentication.models`, [5](#)

s

`stocks.graphql`, [12](#)
`stocks.models`, [7](#)

t

`trading.graphql`, [9](#)
`trading.models`, [3](#)

A

[accessible_buckets\(\)](#) (stocks.models.InvestmentBucket static method), 7
[account_name\(\)](#) (authentication.models.UserBank method), 5
[add_attribute\(\)](#) (stocks.models.InvestmentBucket method), 7
[AddAttributeToInvestment](#) (class in stocks.graphql), 12
[AddAttributeToInvestment.Arguments](#) (class in stocks.graphql), 12
[AddBucket](#) (class in stocks.graphql), 12
[AddBucket.Arguments](#) (class in stocks.graphql), 12
[AddStock](#) (class in stocks.graphql), 12
[AddStock.Arguments](#) (class in stocks.graphql), 12
[AddTrade](#) (class in trading.graphql), 9
[AddTrade.Arguments](#) (class in trading.graphql), 9
[AddTradingAccount](#) (class in authentication.graphql), 10
[AddTradingAccount.Arguments](#) (class in authentication.graphql), 10
[authentication.graphql](#) (module), 10
[authentication.models](#) (module), 5
[available_buckets\(\)](#) (trading.models.TradingAccount method), 3
[available_cash\(\)](#) (trading.models.TradingAccount method), 3
[available_stocks\(\)](#) (trading.models.TradingAccount method), 3

C

[change_config\(\)](#) (stocks.models.InvestmentBucket method), 7
[change_description\(\)](#) (stocks.models.InvestmentBucketDescription method), 7
[Config](#) (class in stocks.graphql), 13
[create_new_bucket\(\)](#) (stocks.models.InvestmentBucket static method), 7
[create_new_stock\(\)](#) (stocks.models.Stock static method), 8
[create_user_profile\(\)](#) (in module authentication.models),

6

[current_balance\(\)](#) (authentication.models.UserBank method), 5
[current_value\(\)](#) (trading.models.TradeBucket method), 3
[current_value\(\)](#) (trading.models.TradeStock method), 3

D

[DailyStockQuote](#) (class in stocks.models), 7
[DataPoint](#) (class in stocks.graphql), 13
[default_acc\(\)](#) (authentication.models.Profile method), 5
[DeleteAttribute](#) (class in stocks.graphql), 13
[DeleteAttribute.Arguments](#) (class in stocks.graphql), 13
[DeleteBucket](#) (class in stocks.graphql), 13
[DeleteBucket.Arguments](#) (class in stocks.graphql), 13

E

[EditAttribute](#) (class in stocks.graphql), 13
[EditAttribute.Arguments](#) (class in stocks.graphql), 13
[EditConfiguration](#) (class in stocks.graphql), 13
[EditConfiguration.Arguments](#) (class in stocks.graphql), 13
[expenditure\(\)](#) (authentication.models.UserBank method), 5

F

[find_stock\(\)](#) (stocks.models.Stock static method), 8

G

[GDailyStockQuote](#) (class in stocks.graphql), 14
[GDataPoint](#) (class in stocks.graphql), 14
[get_stock_configs\(\)](#) (stocks.models.InvestmentBucket method), 7
[GInvestmentBucket](#) (class in stocks.graphql), 14
[GInvestmentBucketAttribute](#) (class in stocks.graphql), 14
[GInvestmentBucketConfigurationUpdate](#) (class in stocks.graphql), 14
[GInvestmentBucketTrade](#) (class in trading.graphql), 9
[GInvestmentStockConfiguration](#) (class in stocks.graphql), 14

GProfile (class in authentication.graphql), 10
GStock (class in stocks.graphql), 14
GTrade (class in trading.graphql), 9
GTradingAccount (class in trading.graphql), 9
GUser (class in authentication.graphql), 11
GUserBank (class in authentication.graphql), 11

H

has_enough_bucket() (trading.models.TradingAccount method), 3
has_enough_cash() (trading.models.TradingAccount method), 3
has_enough_stock() (trading.models.TradingAccount method), 3
historical() (stocks.models.InvestmentBucket method), 7
historical_data() (authentication.models.UserBank method), 5
holding_value() (trading.models.TradingAccount method), 4

I

id (stocks.graphql.Config attribute), 13
income() (authentication.models.UserBank method), 6
InvestBucket (class in trading.graphql), 10
InvestBucket.Arguments (class in trading.graphql), 10
InvestmentBucket (class in stocks.models), 7
InvestmentBucketDescription (class in stocks.models), 7
InvestmentStockConfiguration (class in stocks.models), 8

L

latest_quote() (stocks.models.Stock method), 8

M

mutate() (authentication.graphql.AddTradingAccount static method), 10
mutate() (stocks.graphql.AddAttributeToInvestment static method), 12
mutate() (stocks.graphql.AddBucket static method), 12
mutate() (stocks.graphql.AddStock static method), 13
mutate() (stocks.graphql.DeleteAttribute static method), 13
mutate() (stocks.graphql.DeleteBucket static method), 13
mutate() (stocks.graphql.EditAttribute static method), 13
mutate() (stocks.graphql.EditConfiguration static method), 14
mutate() (trading.graphql.AddTrade static method), 9
mutate() (trading.graphql.InvestBucket static method), 10

P

plaid() (authentication.models.UserBank method), 6
pre_save_any() (in module stocks.models), 8
Profile (class in authentication.models), 5

Q

quantity (stocks.graphql.Config attribute), 13
Query (class in authentication.graphql), 12
Query (class in stocks.graphql), 14
Query (class in trading.graphql), 10
quote_in_range() (stocks.models.Stock method), 8

R

resolve_available_cash() (trading.graphql.GTradingAccount static method), 9
resolve_balance() (authentication.graphql.GUserBank static method), 11
resolve_balance_date() (authentication.graphql.GUserBank static method), 11
resolve_history() (authentication.graphql.GUserBank static method), 11
resolve_history() (stocks.graphql.GInvestmentBucket static method), 14
resolve_income() (authentication.graphql.GUserBank static method), 11
resolve_invest_bucket() (stocks.graphql.Query static method), 15
resolve_invest_suggestions() (authentication.graphql.GProfile static method), 10
resolve_is_owner() (stocks.graphql.GInvestmentBucket static method), 14
resolve_latest_quote() (stocks.graphql.GStock static method), 14
resolve_monthly_end() (authentication.graphql.GUserBank static method), 11
resolve_monthly_start() (authentication.graphql.GUserBank static method), 11
resolve_name() (authentication.graphql.GUserBank static method), 12
resolve_outcome() (authentication.graphql.GUserBank static method), 12
resolve_owned_amount() (stocks.graphql.GInvestmentBucket static method), 14
resolve_quote_in_range() (stocks.graphql.GStock static method), 14
resolve_selected_acc() (authentication.graphql.GProfile static method), 10
resolve_stock_find() (authentication.graphql.GProfile static method), 11
resolve_stocks() (stocks.graphql.GInvestmentBucket static method), 14
resolve_total_value() (trading.graphql.GTradingAccount static method), 10

resolve_trades() (stocks.graphql.GStock static method),
14
resolve_value() (stocks.graphql.GInvestmentBucket
static method), 14
resolve_value() (trading.graphql.GInvestmentBucketTrade
static method), 9
resolve_value() (trading.graphql.GTrade static method), 9
resolve_viewer() (authentication.graphql.Query static
method), 12

S

save_user_profile() (in module authentication.models), 6
Stock (class in stocks.models), 8
stocks.graphql (module), 12
stocks.models (module), 7

T

total_value() (trading.models.TradingAccount method), 4
trade_bucket() (trading.models.TradingAccount method),
4
trade_stock() (trading.models.TradingAccount method),
4
TradeBucket (class in trading.models), 3
trades_for_profile() (stocks.models.Stock method), 8
TradeStock (class in trading.models), 3
trading.graphql (module), 9
trading.models (module), 3
trading_balance() (trading.models.TradingAccount
method), 4
TradingAccount (class in trading.models), 3

U

UserBank (class in authentication.models), 5

V

value_on() (stocks.models.InvestmentBucket method), 7
value_on() (stocks.models.InvestmentStockConfiguration
method), 8