F:\git\java\mar3\filemonitor\target\peatio-master\peatio-master-0.doc

0:F:\git\coin\exchange\peatio-master\app\api\api_v2\auth\authenticator.rb

```
module APIv2
  module Auth
    class Authenticator

      def initialize(request, params)
        @request = request
        @params  = params
      end

      def authenticate!
        check_token!
        check_tonce!
        check_signature!
        token
      end

      def token
        @token ||= APIToken.joins(:member).where(access_key: @params[:access_key]).first
      end

      def check_token!
        raise InvalidAccessKeyError, @params[:access_key] unless token
        raise DisabledAccessKeyError, @params[:access_key] if token.member.api_disabled
        raise ExpiredAccessKeyError, @params[:access_key] if token.expired?
        raise OutOfScopeError unless token.in_scopes?(route_scopes)
      end

      def check_signature!
        if @params[:signature] != Utils.hmac_signature(token.secret_key, payload)
          Rails.logger.warn "APIv2 auth failed: signature doesn't match. token: #{token.access_key}
payload: #{payload}"
          raise IncorrectSignatureError, @params[:signature]
        end
      end

      def check_tonce!
        key = "api_v2:tonce:#{token.access_key}:#{tonce}"
        if Utils.cache.read(key)
          Rails.logger.warn "APIv2 auth failed: used tonce. token: #{token.access_key} payload:
```

```
#{payload} tonce: #{tonce}"
          raise TonceUsedError.new(token.access_key, tonce)
        end
        Utils.cache.write key, tonce, 61 # forget after 61 seconds

        now = Time.now.to_i*1000
        if tonce < now-30000 || tonce > now+30000 # within 30 seconds
          Rails.logger.warn "APIv2 auth failed: invalid tonce. token: #{token.access_key} payload:
#{payload} tonce: #{tonce} current timestamp: #{now}"
          raise InvalidTonceError.new(tonce, now)
        end
      end

      def tonce
        @tonce ||= @params[:tonce].to_i
      end

      def payload
        "#{canonical_verb}|#{APIv2::Mount::PREFIX}#{canonical_uri}|#{canonical_query}"
      end

      def canonical_verb
        @request.request_method
      end

      def canonical_uri
        @request.path_info
      end

      def canonical_query
        hash = @params.select {|k,v| !%w(route_info signature format).include?(k) }
        URI.unescape(hash.to_param)
      end

      def endpoint
        @request.env['api.endpoint']
      end

      def route_scopes
        endpoint.options[:route_options][:scopes]
      end
```

```
    end
  end
end
```

1:F:\git\coin\exchange\peatio-master\app\api\api_v2\auth\middleware.rb

```
module APIv2
  module Auth
    class Middleware < ::Grape::Middleware::Base

      def before
        if provided?
          auth = Authenticator.new(request, params)
          @env['api_v2.token'] = auth.authenticate!
        end
      end

      def provided?
        params[:access_key] && params[:tonce] && params[:signature]
      end

      def request
        @request ||= ::Grape::Request.new(env)
      end

      def params
        @params ||= request.params
      end

    end
  end
end
```

2:F:\git\coin\exchange\peatio-master\app\api\api_v2\auth\utils.rb

```
module APIv2
  module Auth
    module Utils
      class <<self

        def cache
          # Simply use rack-attack cache wrapper
          @cache ||= Rack::Attack::Cache.new
        end
        def urlsafe_string_40
          # 30 is picked so generated string length is 40
          SecureRandom.urlsafe_base64(30).tr('_-', 'xx')
        end

        alias :generate_access_key :urlsafe_string_40
        alias :generate_secret_key :urlsafe_string_40

        def hmac_signature(secret_key, payload)
          OpenSSL::HMAC.hexdigest 'SHA256', secret_key, payload
        end

      end
    end
  end
end
```

3:F:\git\coin\exchange\peatio-master\app\api\api_v2\constraints.rb

```
module APIv2
  module Constraints
    class <<self

      def included(base)
        apply_rules!
        base.use Rack::Attack
      end

      def apply_rules!
        Rack::Attack.blacklist('block api access from other ip if trusted ip set ') do |req|
          req.env['api_v2.token'] && !req.env['api_v2.token'].allow_ip?(req.ip)
        end

        Rack::Attack.throttle('Authorized access', limit: 6000, period: 5.minutes) do |req|
          req.env['api_v2.token'] && req.env['api_v2.token'].access_key
        end
      end

    end
  end
end
```

```
4:F:\git\coin\exchange\peatio-master\app\api\api_v2\deposits.rb
require_relative 'validations'

module APIv2
  class Deposits < Grape::API
    helpers ::APIv2::NamedParams

    before { authenticate! }

    desc 'Get your deposits history.'
    params do
      use :auth
      optional :currency, type: String, values: Currency.all.map(&:code), desc: "Currency value
contains  #{Currency.all.map(&:code).join(',')}"
      optional :limit, type: Integer, range: 1..100, default: 3, desc: "Set result limit."
      optional :state, type: String, values: Deposit::STATES.map(&:to_s)
    end
    get "/deposits" do
      deposits = current_user.deposits.limit(params[:limit]).recent
      deposits = deposits.with_currency(params[:currency]) if params[:currency]
      deposits = deposits.with_aasm_state(params[:state]) if params[:state].present?

      present deposits, with: APIv2::Entities::Deposit
    end

    desc 'Get details of specific deposit.'
    params do
      use :auth
      requires :txid
    end
    get "/deposit" do
      deposit = current_user.deposits.find_by(txid: params[:txid])
      raise DepositByTxidNotFoundError, params[:txid] unless deposit

      present deposit, with: APIv2::Entities::Deposit
    end

    desc 'Where to deposit. The address field could be empty when a new address is generating
(e.g. for bitcoin), you should try again later in that case.'
    params do
      use :auth
      requires :currency, type: String, values: Currency.all.map(&:code), desc: "The account to
```

```
which you want to deposit. Available values: #{Currency.all.map(&:code).join(', ')}"
    end
    get "/deposit_address" do
      current_user.ac(params[:currency]).payment_address.to_json
    end
  end
end
```

5:F:\git\coin\exchange\peatio-master\app\api\api_v2\entities\account.rb

```
module APIv2
  module Entities
    class Account < Base
      expose :currency
      expose :balance, format_with: :decimal
      expose :locked,  format_with: :decimal
    end
  end
end
```

6:F:\git\coin\exchange\peatio-master\app\api\api_v2\entities\base.rb

```
module APIv2
  module Entities
    class Base < Grape::Entity
      format_with(:iso8601) {|t| t.iso8601 if t }
      format_with(:decimal) {|d| d.to_s('F') if d }
    end
  end
end
```

7:F:\git\coin\exchange\peatio-master\app\api\api_v2\entities\deposit.rb

```
module APIv2
  module Entities
    class Deposit < Base
      expose :id, documentation: "Unique deposit id."
      expose :currency
      expose :amount, format_with: :decimal
      expose :fee
      expose :txid
      expose :created_at, format_with: :iso8601
      expose :confirmations
      expose :done_at, format_with: :iso8601
      expose :aasm_state, as: :state
    end
  end
end

8:F:\git\coin\exchange\peatio-master\app\api\api_v2\entities\market.rb
module APIv2
  module Entities
    class Market < Base
      expose :id, documentation: "Unique market id. It's always in the form of xxxyyy, where xxx is the base currency code, yyy is the quote currency code, e.g. 'btccny'. All available markets can be found at /api/v2/markets."

      expose :name
    end
  end
end

9:F:\git\coin\exchange\peatio-master\app\api\api_v2\entities\member.rb
module APIv2
  module Entities
    class Member < Base
      expose :sn
      expose :name
      expose :email
      expose :activated
      expose :accounts, using: ::APIv2::Entities::Account
    end
  end
end

10:F:\git\coin\exchange\peatio-master\app\api\api_v2\entities\order.rb
module APIv2
  module Entities
    class Order < Base
      expose :id, documentation: "Unique order id."

      expose :side, documentation: "Either 'sell' or 'buy'."
      expose :ord_type, documentation: "Type of order, either 'limit' or 'market'."

      expose :price, documentation: "Price for each unit. e.g. If you want to sell/buy 1 btc at 3000 CNY, the price is '3000.0'"
      expose :avg_price, documentation: "Average execution price, average of price in trades."

      expose :state, documentation: "One of 'wait', 'done', or 'cancel'. An order in 'wait' is an active
order, waiting fullfillment; a 'done' order is an order fullfilled; 'cancel' means the order has been
cancelled."

      expose :currency, as: :market, documentation: "The market in which the order is placed, e.g.
'btccny'. All available markets can be found at /api/v2/markets."

      expose :created_at, format_with: :iso8601, documentation: "Order create time in iso8601
format."

      expose :origin_volume, as: :volume, documentation: "The amount user want to sell/buy. An
order could be partially executed, e.g. an order sell 5 btc can be matched with a buy 3 btc order,
left 2 btc to be sold; in this case the order's volume would be '5.0', its remaining_volume would be
'2.0', its executed volume is '3.0'."

      expose :volume, as: :remaining_volume, documentation: "The remaining volume, see
'volume'."

      expose :executed_volume, documentation: "The executed volume, see 'volume'." do |order,
options|
        order.origin_volume - order.volume
      end

      expose :trades_count
      expose :trades, if: {type: :full} do |order, options|
        ::APIv2::Entities::Trade.represent order.trades, side: side
      end

      private

      def side
        @side ||= @object.type[-3, 3] == 'Ask' ? 'sell' : 'buy'
      end

    end
  end
end

11:F:\git\coin\exchange\peatio-master\app\api\api_v2\entities\order_book.rb
module APIv2
```

```
  module Entities
    class OrderBook < Base
      expose :asks, using: Order
      expose :bids, using: Order
    end
  end
end
```

12:F:\git\coin\exchange\peatio-master\app\api\api_v2\entities\trade.rb

```
module APIv2
  module Entities
    class Trade < Base
      expose :id
      expose :price
      expose :volume
      expose :funds
      expose :currency, as: :market
      expose :created_at, format_with: :iso8601

      expose :side do |trade, options|
        options[:side] || trade.side
      end

      expose :order_id, if: ->(trade, options){ options[:current_user] } do |trade, options|
        if trade.ask_member_id == options[:current_user].id
          trade.ask_id
        elsif trade.bid_member_id == options[:current_user].id
          trade.bid_id
        else
          nil
        end
      end

    end
  end
end
```

13:F:\git\coin\exchange\peatio-master\app\api\api_v2\errors.rb

```
module APIv2

  module ExceptionHandlers
  def self.included(base)
    base.instance_eval do
      rescue_from Grape::Exceptions::ValidationErrors do |e|
        Rack::Response.new({
          error: {
            code: 1001,
            message: e.message
          }
        }.to_json, e.status)
      end
    end
  end

end

class Error < Grape::Exceptions::Base
  attr :code, :text

  # code: api error code defined by Peatio, errors originated from
  # subclasses of Error have code start from 2000.
  # text: human readable error message
  # status: http status code
  def initialize(opts={})
    @code    = opts[:code]   || 2000
    @text    = opts[:text]   || ''

    @status  = opts[:status] || 400
    @message = {error: {code: @code, message: @text}}
  end
end

class AuthorizationError < Error
  def initialize
    super code: 2001, text: 'Authorization failed', status: 401
  end
end

class CreateOrderError < Error
  def initialize(e)
    super code: 2002, text: "Failed to create order. Reason: #{e}", status: 400
  end
end
```

```
  class CancelOrderError < Error
    def initialize(e)
      super code: 2003, text: "Failed to cancel order. Reason: #{e}", status: 400
    end
  end

  class OrderNotFoundError < Error
    def initialize(id)
      super code: 2004, text: "Order##{id} doesn't exist.", status: 404
    end
  end

  class IncorrectSignatureError < Error
    def initialize(signature)
      super code: 2005, text: "Signature #{signature} is incorrect.", status: 401
    end
  end

  class TonceUsedError < Error
    def initialize(access_key, tonce)
      super code: 2006, text: "The tonce #{tonce} has already been used by access key
#{access_key}.", status: 401
    end
  end

  class InvalidTonceError < Error
    def initialize(tonce, now)
      super code: 2007, text: "The tonce #{tonce} is invalid, current timestamp is #{now}.", status:
401
    end
  end

  class InvalidAccessKeyError < Error
    def initialize(access_key)
      super code: 2008, text: "The access key #{access_key} does not exist.", status: 401
    end
  end

  class DisabledAccessKeyError < Error
    def initialize(access_key)
      super code: 2009, text: "The access key #{access_key} is disabled.", status: 401
    end
  end

  class ExpiredAccessKeyError < Error
    def initialize(access_key)
      super code: 2010, text: "The access key #{access_key} has expired.", status: 401
    end
  end

  class OutOfScopeError < Error
    def initialize
      super code: 2011, text: "Requested API is out of access key scopes.", status: 401
    end
  end

  class DepositByTxidNotFoundError < Error
    def initialize(txid)
      super code: 2012, text: "Deposit##txid=#{txid} doesn't exist.", status: 404
    end
  end
end
```

14:F:\git\coin\exchange\peatio-master\app\api\api_v2\helpers.rb

```
module APIv2
  module Helpers

    def authenticate!
      current_user or raise AuthorizationError
    end

    def redis
      @r ||= KlineDB.redis
    end

    def current_user
      @current_user ||= current_token.try(:member)
    end

    def current_token
      @current_token ||= env['api_v2.token']
    end
```

```
def current_market
  @current_market ||= Market.find params[:market]
end

def time_to
  params[:timestamp].present? ? Time.at(params[:timestamp]) : nil
end

def build_order(attrs)
  klass = attrs[:side] == 'sell' ? OrderAsk : OrderBid

  order = klass.new(
    source:        'APIv2',
    state:         ::Order::WAIT,
    member_id:     current_user.id,
    ask:           current_market.base_unit,
    bid:           current_market.quote_unit,
    currency:      current_market.id,
    ord_type:      attrs[:ord_type] || 'limit',
    price:         attrs[:price],
    volume:        attrs[:volume],
    origin_volume: attrs[:volume]
  )
end

def create_order(attrs)
  order = build_order attrs
  Ordering.new(order).submit
  order
rescue
  Rails.logger.info "Failed to create order: #{$!}"
  Rails.logger.debug order.inspect
  Rails.logger.debug $!.backtrace.join("\n")
  raise CreateOrderError, $!
end

def create_orders(multi_attrs)
  orders = multi_attrs.map {|attrs| build_order attrs }
  Ordering.new(orders).submit
  orders
rescue
  Rails.logger.info "Failed to create order: #{$!}"
```

```
      Rails.logger.debug $!.backtrace.join("\n")
      raise CreateOrderError, $!
    end

    def order_param
      params[:order_by].downcase == 'asc' ? 'id asc' : 'id desc'
    end

    def format_ticker(ticker)
      { at: ticker[:at],
        ticker: {
          buy: ticker[:buy],
          sell: ticker[:sell],
          low: ticker[:low],
          high: ticker[:high],
          last: ticker[:last],
          vol: ticker[:volume]
        }
      }
    end

    def get_k_json
      key = "peatio:#{params[:market]}:k:#{params[:period]}"

      if params[:timestamp]
        ts = JSON.parse(redis.lindex(key, 0)).first
        offset = (params[:timestamp] - ts) / 60 / params[:period]
        offset = 0 if offset < 0

        JSON.parse('[%s]' % redis.lrange(key, offset, offset + params[:limit] - 1).join(','))
      else
        length = redis.llen(key)
        offset = [length - params[:limit], 0].max
        JSON.parse('[%s]' % redis.lrange(key, offset, -1).join(','))
      end
    end

  end
end
```

15:F:\git\coin\exchange\peatio-master\app\api\api_v2\k.rb

```
module APIv2
```

```
  class K < Grape::API
    helpers ::APIv2::NamedParams

    desc 'Get OHLC(k line) of specific market.'
    params do
      use :market
      optional :limit,     type: Integer, default: 30, values: 1..10000, desc: "Limit the number of returned data points, default to 30."
      optional :period,    type: Integer, default: 1, values: [1, 5, 15, 30, 60, 120, 240, 360, 720, 1440, 4320, 10080], desc: "Time period of K line, default to 1. You can choose between 1, 5, 15, 30, 60, 120, 240, 360, 720, 1440, 4320, 10080"
      optional :timestamp, type: Integer, desc: "An integer represents the seconds elapsed since Unix epoch. If set, only k-line data after that time will be returned."
    end
    get "/k" do
      get_k_json
    end

    desc "Get K data with pending trades, which are the trades not included in K data yet, because there's delay between trade generated and processed by K data generator."
    params do
      use :market
      requires :trade_id,  type: Integer, desc: "The trade id of the first trade you received."
      optional :limit,     type: Integer, default: 30, values: 1..10000, desc: "Limit the number of returned data points, default to 30."
      optional :period,    type: Integer, default: 1, values: [1, 5, 15, 30, 60, 120, 240, 360, 720, 1440, 4320, 10080], desc: "Time period of K line, default to 1. You can choose between 1, 5, 15, 30, 60, 120, 240, 360, 720, 1440, 4320, 10080"
      optional :timestamp, type: Integer, desc: "An integer represents the seconds elapsed since Unix epoch. If set, only k-line data after that time will be returned."
    end
    get "/k_with_pending_trades" do
      k = get_k_json

      if params[:trade_id] > 0
        from = Time.at k.last[0]
        trades = Trade.with_currency(params[:market])
          .where('created_at >= ? AND id < ?', from, params[:trade_id])
          .map(&:for_global)

        {k: k, trades: trades}
      else
```

```
        {k: k, trades: []}
      end
    end

  end
end
```

16:F:\git\coin\exchange\peatio-master\app\api\api_v2\markets.rb

```
module APIv2
  class Markets < Grape::API

    desc 'Get all available markets.'
    get "/markets" do
      present Market.all, with: APIv2::Entities::Market
    end

  end
end
```

17:F:\git\coin\exchange\peatio-master\app\api\api_v2\members.rb

```
module APIv2
  class Members < Grape::API
    helpers ::APIv2::NamedParams

    desc 'Get your profile and accounts info.', scopes: %w(profile)
    params do
      use :auth
    end
    get "/members/me" do
      authenticate!
      present current_user, with: APIv2::Entities::Member
    end

  end
end
```

18:F:\git\coin\exchange\peatio-master\app\api\api_v2\mount.rb

```
require_relative 'errors'
require_relative 'validations'

module APIv2
  class Mount < Grape::API
```

```
    PREFIX = '/api'

    version 'v2', using: :path

    cascade false

    format :json
    default_format :json

    helpers ::APIv2::Helpers

    do_not_route_options!

    use APIv2::Auth::Middleware

    include Constraints
    include ExceptionHandlers

    before do
      header 'Access-Control-Allow-Origin', '*'
    end

    mount Markets
    mount Tickers
    mount Members
    mount Deposits
    mount Orders
    mount OrderBooks
    mount Trades
    mount K
    mount Tools

    base_path = Rails.env.production? ?
"#{ENV['URL_SCHEMA']}://#{ENV['URL_HOST']}/#{PREFIX}" : PREFIX
    add_swagger_documentation base_path: base_path,
      mount_path: '/doc/swagger', api_version: 'v2',
      hide_documentation_path: true
  end
end
```

19:F:\git\coin\exchange\peatio-master\app\api\api_v2\named_params.rb

```
module APIv2
```

```
  module NamedParams
    extend ::Grape::API::Helpers

    params :auth do
      requires :access_key, type: String,  desc: "Access key."
      requires :tonce,      type: Integer, desc: "Tonce is an integer represents the milliseconds
elapsed since Unix epoch."
      requires :signature,  type: String,  desc: "The signature of your request payload, generated
using your secret key."
    end

    params :market do
      requires :market, type: String, values: ::Market.all.map(&:id), desc:
::APIv2::Entities::Market.documentation[:id]
    end

    params :order do
      requires :side,   type: String, values: %w(sell buy), desc:
::APIv2::Entities::Order.documentation[:side]
      requires :volume, type: String, desc: ::APIv2::Entities::Order.documentation[:volume]
      optional :price,  type: String, desc: ::APIv2::Entities::Order.documentation[:price]
      optional :ord_type, type: String, values: %w(limit market), desc:
::APIv2::Entities::Order.documentation[:type]
    end

    params :order_id do
      requires :id, type: Integer, desc: ::APIv2::Entities::Order.documentation[:id]
    end

    params :trade_filters do
      optional :limit,     type: Integer, range: 1..1000, default: 50, desc: 'Limit the number of returned
trades. Default to 50.'
      optional :timestamp, type: Integer, desc: "An integer represents the seconds elapsed since
Unix epoch. If set, only trades executed before the time will be returned."
      optional :from,      type: Integer, desc: "Trade id. If set, only trades created after the trade will
be returned."
      optional :to,        type: Integer, desc: "Trade id. If set, only trades created before the trade will
be returned."
      optional :order_by,     type: String, values: %w(asc desc), default: 'desc', desc: "If set, returned
trades will be sorted in specific order, default to 'desc'."
    end
  end
end

20:F:\git\coin\exchange\peatio-master\app\api\api_v2\orders.rb
module APIv2
  class Orders < Grape::API
    helpers ::APIv2::NamedParams

    before { authenticate! }

    desc 'Get your orders, results is paginated.', scopes: %w(history trade)
    params do
      use :auth, :market
      optional :state, type: String,  default: 'wait', values: Order.state.values, desc: "Filter order by
state, default to 'wait' (active orders)."
      optional :limit, type: Integer, default: 100, range: 1..1000, desc: "Limit the number of returned
orders, default to 100."
      optional :page,  type: Integer, default: 1, desc: "Specify the page of paginated results."
      optional :order_by, type: String, values: %w(asc desc), default: 'asc', desc: "If set, returned
orders will be sorted in specific order, default to 'asc'."
    end
    get "/orders" do
      orders = current_user.orders
        .order(order_param)
        .with_currency(current_market)
        .with_state(params[:state])
        .page(params[:page])
        .per(params[:limit])

      present orders, with: APIv2::Entities::Order
    end

    desc 'Get information of specified order.', scopes: %w(history trade)
    params do
      use :auth, :order_id
    end
    get "/order" do
      order = current_user.orders.where(id: params[:id]).first
      raise OrderNotFoundError, params[:id] unless order
      present order, with: APIv2::Entities::Order, type: :full
    end
    desc 'Create multiple sell/buy orders.', scopes: %w(trade)
    params do
      use :auth, :market
      requires :orders, type: Array do
        use :order
      end
    end
    post "/orders/multi" do
        orders = create_orders params[:orders]
        present orders, with: APIv2::Entities::Order
    end

    desc 'Create a Sell/Buy order.', scopes: %w(trade)
    params do
      use :auth, :market, :order
    end
    post "/orders" do
      order = create_order params
      present order, with: APIv2::Entities::Order
    end

    desc 'Cancel an order.', scopes: %w(trade)
    params do
      use :auth, :order_id
    end
    post "/order/delete" do
      begin
        order = current_user.orders.find(params[:id])
        Ordering.new(order).cancel
        present order, with: APIv2::Entities::Order
      rescue
        raise CancelOrderError, $!
      end
    end

    desc 'Cancel all my orders.', scopes: %w(trade)
    params do
      use :auth
      optional :side, type: String, values: %w(sell buy), desc: "If present, only sell orders (asks) or
buy orders (bids) will be canncelled."
    end
    post "/orders/clear" do
```

```
      begin
        orders = current_user.orders.with_state(:wait)
        if params[:side].present?
          type = params[:side] == 'sell' ? 'OrderAsk' : 'OrderBid'
          orders = orders.where(type: type)
        end
        orders.each {|o| Ordering.new(o).cancel }
        present orders, with: APIv2::Entities::Order
      rescue
        raise CancelOrderError, $!
      end
    end

  end
end
```

21:F:\git\coin\exchange\peatio-master\app\api\api_v2\order_books.rb

```
module APIv2
  class OrderBook < Struct.new(:asks, :bids); end

  class OrderBooks < Grape::API
    helpers ::APIv2::NamedParams

    desc 'Get the order book of specified market.'
    params do
      use :market
      optional :asks_limit, type: Integer, default: 20, range: 1..200, desc: 'Limit the number of
returned sell orders. Default to 20.'
      optional :bids_limit, type: Integer, default: 20, range: 1..200, desc: 'Limit the number of
returned buy orders. Default to 20.'
    end
    get "/order_book" do
      asks =
OrderAsk.active.with_currency(params[:market]).matching_rule.limit(params[:asks_limit])
      bids = OrderBid.active.with_currency(params[:market]).matching_rule.limit(params[:bids_limit])
      book = OrderBook.new asks, bids
      present book, with: APIv2::Entities::OrderBook
    end

    desc 'Get depth or specified market. Both asks and bids are sorted from highest price to lowest.'
    params do
      use :market
      optional :limit, type: Integer, default: 300, range: 1..1000, desc: 'Limit the number of returned
price levels. Default to 300.'
    end
    get "/depth" do
      global = Global[params[:market]]
      asks = global.asks[0,params[:limit]].reverse
      bids = global.bids[0,params[:limit]]
      {timestamp: Time.now.to_i, asks: asks, bids: bids}
    end

  end
end
```

22:F:\git\coin\exchange\peatio-master\app\api\api_v2\tickers.rb

```
module APIv2
  class Tickers < Grape::API
    helpers ::APIv2::NamedParams

    desc 'Get ticker of all markets.'
    get "/tickers" do
      Market.all.inject({}) do |h, m|
        h[m.id] = format_ticker Global[m.id].ticker
        h
      end
    end

    desc 'Get ticker of specific market.'
    params do
      use :market
    end
    get "/tickers/:market" do
      format_ticker Global[params[:market]].ticker
    end

  end
end
```

23:F:\git\coin\exchange\peatio-master\app\api\api_v2\tools.rb

```
module APIv2
  class Tools < Grape::API
    desc 'Get server current time, in seconds since Unix epoch.'
    get "/timestamp" do
```

```
      ::Time.now.to_i
    end
  end
end

24:F:\git\coin\exchange\peatio-master\app\api\api_v2\trades.rb
module APIv2
  class Trades < Grape::API
    helpers ::APIv2::NamedParams

    desc 'Get recent trades on market, each trade is included only once. Trades are sorted in
reverse creation order.'
    params do
      use :market, :trade_filters
    end
    get "/trades" do
      trades = Trade.filter(params[:market], time_to, params[:from], params[:to], params[:limit],
order_param)
      present trades, with: APIv2::Entities::Trade
    end

    desc 'Get your executed trades. Trades are sorted in reverse creation order.', scopes:
%w(history)
    params do
      use :auth, :market, :trade_filters
    end
    get "/trades/my" do
      authenticate!

      trades = Trade.for_member(
        params[:market], current_user,
        limit: params[:limit], time_to: time_to,
        from: params[:from], to: params[:to],
        order: order_param
      )

      present trades, with: APIv2::Entities::Trade, current_user: current_user
    end

  end
end
```

25:F:\git\coin\exchange\peatio-master\app\api\api_v2\validations.rb

```
module APIv2
  module Validations
    class Range < ::Grape::Validations::Validator

      def initialize(attrs, options, required, scope)
        @range    = options
        @required = required
        super
      end

      def validate_param!(attr_name, params)
        if (params[attr_name] || @required) && !@range.cover?(params[attr_name])
          raise Grape::Exceptions::Validation, param: @scope.full_name(attr_name), message: "must
be in range: #{@range}"
        end
      end

    end
  end
end
```

26:F:\git\coin\exchange\peatio-master\app\api\api_v2\websocket_protocol.rb

```
module APIv2
  class WebSocketProtocol

    def initialize(socket, channel, logger)
      @socket = socket
      @channel = channel #FIXME: amqp should not be mixed into this class
      @logger = logger
    end

    def challenge
      @challenge = SecureRandom.urlsafe_base64(40)
      send :challenge, @challenge
    end

    def handle(message)
      @logger.debug message

      message = JSON.parse(message)
      key     = message.keys.first
  data    = message[key]

  case key.downcase
  when 'auth'
    access_key = data['access_key']
    token = APIToken.where(access_key: access_key).includes(:member).first
    result = verify_answer data['answer'], token

    if result
      subscribe_orders
      subscribe_trades token.member
      send :success, {message: "Authenticated."}
    else
      send :error, {message: "Authentication failed."}
    end
  else
  end
rescue
  @logger.error "Error on handling message: #{$!}"
  @logger.error $!.backtrace.join("\n")
end

private

def send(method, data)
  payload = JSON.dump({method => data})
  @logger.debug payload
  @socket.send payload
end

def verify_answer(answer, token)
  str = "#{token.access_key}#{@challenge}"
  answer == OpenSSL::HMAC.hexdigest('SHA256', token.secret_key, str)
end

def subscribe_orders
  x = @channel.send *AMQPConfig.exchange(:orderbook)
  q = @channel.queue '', auto_delete: true
  q.bind(x).subscribe do |metadata, payload|
    begin
      payload = JSON.parse payload
      send :orderbook, payload
    rescue
      @logger.error "Error on receiving orders: #{$!}"
      @logger.error $!.backtrace.join("\n")
    end
  end
end

def subscribe_trades(member)
  x = @channel.send *AMQPConfig.exchange(:trade)
  q = @channel.queue '', auto_delete: true
  q.bind(x, arguments: {'ask_member_id' => member.id, 'bid_member_id' => member.id, 'x-
match' => 'any'})
  q.subscribe(ack: true) do |metadata, payload|
    begin
      payload = JSON.parse payload
      trade   = Trade.find payload['id']

      send :trade, serialize_trade(trade, member, metadata)
    rescue
      @logger.error "Error on receiving trades: #{$!}"
      @logger.error $!.backtrace.join("\n")
    ensure
      metadata.ack
    end
  end
end

def serialize_trade(trade, member, metadata)
  side = trade_side(member, metadata.headers)
  hash = ::APIv2::Entities::Trade.represent(trade, side: side).serializable_hash

  if [:both, :ask].include?(side)
    hash[:ask] = ::APIv2::Entities::Order.represent trade.ask
  end

  if [:both, :bid].include?(side)
    hash[:bid] = ::APIv2::Entities::Order.represent trade.bid
  end

  hash
end
```

```
    def trade_side(member, headers)
      if headers['ask_member_id'] == headers['bid_member_id']
        :both
      elsif headers['ask_member_id'] == member.id
        :ask
      else
        :bid
      end
    end

  end
end
```

27:F:\git\coin\exchange\peatio-master\app\controllers\activations_controller.rb

```
class ActivationsController < ApplicationController
  include Concerns::TokenManagement

  before_action :auth_member!,    only: :new
  before_action :verified?,       only: :new
  before_action :token_required!, only: :edit

  def new
    current_user.send_activation
    redirect_to settings_path
  end

  def edit
    @token.confirm!

    if current_user
      redirect_to settings_path, notice: t('.notice')
    else
      redirect_to signin_path, notice: t('.notice')
    end
  end

  private

  def verified?
    if current_user.activated?
      redirect_to settings_path, notice: t('.verified')
    end
  end

end

28:F:\git\coin\exchange\peatio-master\app\controllers\admin\base_controller.rb
module Admin
  class BaseController < ::ApplicationController
    layout 'admin'

    before_action :auth_admin!
    before_action :auth_member!
    before_action :two_factor_required!

    def current_ability
      @current_ability ||= Admin::Ability.new(current_user)
    end

    def two_factor_required!
      if two_factor_locked?(expired_at: ENV['SESSION_EXPIRE'].to_i.minutes)
        session[:return_to] = request.original_url
        redirect_to two_factors_path
      end
    end
  end
end


29:F:\git\coin\exchange\peatio-master\app\controllers\admin\comments_controller.rb
module Admin
  class CommentsController < BaseController

    def create
      comment = ticket.comments.new(comment_params.merge(author_id: current_user.id))

      if comment.save
        flash[:notice] = I18n.t("private.tickets.comment_succ")
      else
        flash[:alert] = I18n.t("private.tickets.comment_fail")
      end
      redirect_to admin_ticket_path(ticket)
    end
    protected

    def comment_params
      params.required(:comment).permit(:content)
    end

    def ticket
      @ticket ||= Ticket.find(params[:ticket_id])
    end

  end
end
```

30:F:\git\coin\exchange\peatio-master\app\controllers\admin\dashboard_controller.rb

```
module Admin
  class DashboardController < BaseController
    skip_load_and_authorize_resource

    def index
      @daemon_statuses = Global.daemon_statuses
      @currencies_summary = Currency.all.map(&:summary)
      @register_count = Member.count
    end
  end
end
```

31:F:\git\coin\exchange\peatio-master\app\controllers\admin\deposits\banks_controller.rb

```
module Admin
  module Deposits
    class BanksController < ::Admin::Deposits::BaseController
      load_and_authorize_resource :class => '::Deposits::Bank'

      def index
        start_at = DateTime.now.ago(60 * 60 * 24)
        @oneday_banks = @banks.includes(:member).
          where('created_at > ?', start_at).
          order('id DESC')

        @available_banks = @banks.includes(:member).
          with_aasm_state(:submitting, :warning, :submitted).
          order('id DESC')
        @available_banks -= @oneday_banks
      end

      def show
        flash.now[:notice] = t('.notice') if @bank.aasm_state.accepted?
      end

      def update
        if target_params[:txid].blank?
          flash[:alert] = t('.blank_txid')
          redirect_to :back and return
        end

        @bank.charge!(target_params[:txid])

        redirect_to :back
      end

      private
      def target_params
        params.require(:deposits_bank).permit(:sn, :holder, :amount, :created_at, :txid)
      end
    end
  end
end
```

32:F:\git\coin\exchange\peatio-master\app\controllers\admin\deposits\base_controller.rb

```
module Admin
  module Deposits
    class BaseController < ::Admin::BaseController
      def channel
        @channel ||= DepositChannel.find_by_key(self.controller_name.singularize)
      end

      def kls
        channel.kls
      end
    end
  end
end
```

33:F:\git\coin\exchange\peatio-master\app\controllers\admin\deposits\satoshis_controller.rb

```ruby
module Admin
  module Deposits
    class SatoshisController < ::Admin::Deposits::BaseController
      load_and_authorize_resource :class => '::Deposits::Satoshi'

      def index
        start_at = DateTime.now.ago(60 * 60 * 24 * 365)
        @satoshis = @satoshis.includes(:member).
          where('created_at > ?', start_at).
          order('id DESC').page(params[:page]).per(20)
      end

      def update
        @satoshi.accept! if @satoshi.may_accept?
        redirect_to :back, notice: t('.notice')
      end
    end
  end
end
```

34:F:\git\coin\exchange\peatio-master\app\controllers\admin\deposits_controller.rb

```ruby
module Admin
  class DepositsController < BaseController
    def index
      @admin_deposits_grid = Admin::DepositsGrid.new \
        params[:admin_deposits_grid]
      @assets = @admin_deposits_grid.assets.page(params[:page]).per(10)
    end

    def edit
      @deposit = Deposit.find(params[:id])
    end

    def update
      # accpet
      @deposit = Deposit.find(params[:id])

      ActiveRecord::Base.transaction do
        if @deposit.accept! or @deposit.submit!
          redirect_to edit_admin_deposit_path(@deposit), notice: t('.notice')
        else
          redirect_to edit_admin_deposit_path(@deposit), alert: t('.alert')
        end
      end
    end

    def destroy
      # reject
      @deposit = Deposit.find(params[:id])

      ActiveRecord::Base.transaction do
        if @deposit.reject!
          redirect_to admin_deposits_path, notice: t('.notice')
        else
          redirect_to edit_admin_deposit_path(@deposit), alert: t('.alert')
        end
      end
    end
  end
end
```

35:F:\git\coin\exchange\peatio-master\app\controllers\admin\documents_controller.rb

```
module Admin
  class DocumentsController < BaseController
    load_and_authorize_resource find_by: :key

    def index
      @documents_grid = ::DocumentsGrid.new(params[:documents_grid])
      @assets = @documents_grid.assets
    end

    def new
    end

    def create
      if @document.save
        redirect_to admin_documents_path
      else
        render :new
      end
    end
    def show
      render inline: @document.body.html_safe
    end

    def edit
    end

    def update
      if @document.update_attributes(document_params)
        redirect_to admin_documents_path
      else
        render :edit
      end
    end

    def destroy
    end

    private

    def document_params
      params.required(:document).permit(:key, :is_auth, *Document.locale_params)
    end

  end
end
```

36:F:\git\coin\exchange\peatio-master\app\controllers\admin\id_documents_controller.rb

```
module Admin
  class IdDocumentsController < BaseController
    load_and_authorize_resource

    def index
      @id_documents = @id_documents.order(:updated_at).reverse_order.page params[:page]
    end

    def show
    end

    def update
```

```
      @id_document.approve! if params[:approve]
      @id_document.reject!  if params[:reject]

      redirect_to admin_id_document_path(@id_document)
    end
  end
end
```

37:F:\git\coin\exchange\peatio-master\app\controllers\admin\members_controller.rb

```
module Admin
  class MembersController < BaseController
    load_and_authorize_resource

    def index
      @search_field = params[:search_field]
      @search_term = params[:search_term]
      @members = Member.search(field: @search_field, term: @search_term).page params[:page]
    end

    def show
      @account_versions = AccountVersion.where(account_id:
@member.account_ids).order(:id).reverse_order.page params[:page]
    end

    def toggle
      if params[:api]
        @member.api_disabled = !@member.api_disabled?
      else
        @member.disabled = !@member.disabled?
      end
      @member.save
    end

    def active
      @member.update_attribute(:activated, true)
      @member.save
      redirect_to admin_member_path(@member)
    end

  end
end
```

38:F:\git\coin\exchange\peatio-master\app\controllers\admin\proofs_controller.rb
```
module Admin
  class ProofsController < BaseController
    load_and_authorize_resource

    def index
      @grid = ProofsGrid.new(params[:proofs_grid])
      @assets = @grid.assets.page(params[:page])
    end

    def edit
    end

    def update
      if @proof.update_attributes(proof_params)
        redirect_to action: :index
      else
        render :edit
      end
    end

    private

    def proof_params
      params.required(:proof).permit(:balance)
    end

  end
end
```

39:F:\git\coin\exchange\peatio-master\app\controllers\admin\statistic\base_controller.rb
```
module Admin
  module Statistic
    class BaseController < ::Admin::BaseController
    end
  end
end
```

40:F:\git\coin\exchange\peatio-master\app\controllers\admin\statistic\deposits_controller.rb
```
module Admin
  module Statistic
    class DepositsController < BaseController
```

```
      prepend_before_filter :load_grid

      def show
        @groups = {
          :count => @assets.all.size,
          :amount => @assets.sum(:amount)
        }
      end

      private
      def load_grid
        @deposits_grid = ::Statistic::DepositsGrid.new(params[:statistic_deposits_grid])
        @assets = @deposits_grid.assets
      end
    end
  end
end
```

41:F:\git\coin\exchange\peatio-master\app\controllers\admin\statistic\members_controller.rb

```
module Admin
  module Statistic
    class MembersController < BaseController
      def show
        @members_count = Member.count
        @register_group = Member.where('created_at > ?', 30.days.ago).select('date(created_at) as date, count(id) as total, sum(activated IS TRUE) as total_activated').group('date(created_at)')
      end
    end
  end
end
```

42:F:\git\coin\exchange\peatio-master\app\controllers\admin\statistic\orders_controller.rb

```
module Admin
  module Statistic
    class OrdersController < BaseController
      def show
        @orders_grid = ::Statistic::OrdersGrid.new(params[:statistic_orders_grid])
        @assets = @orders_grid.assets

        @groups = {
          :count => @assets.size,
          :sum => @assets.sum(:origin_volume),
          :avg => (@assets.average(:price) || 0.to_d).truncate(2),
          :sum_strike => @assets.all.sum do |o|
            o.origin_volume - o.volume
          end
        }
      end
    end
  end
end
```

43:F:\git\coin\exchange\peatio-master\app\controllers\admin\statistic\trades_controller.rb

```
module Admin
  module Statistic
    class TradesController < BaseController
      def show
        @trades_grid = ::Statistic::TradesGrid.new(params[:statistic_trades_grid])
        @assets = @trades_grid.assets

        @groups = {
          :volume => @assets.sum(:volume),
          :amount => @assets.sum {|t| t.price * t.volume},
          :avg_price => @assets.average(:price),
          :max_price => @assets.maximum(:price),
          :min_price => @assets.minimum(:price)
        }

        @groups.merge!({
          :volume_fee => (@groups[:volume]),
          :amount_fee => (@groups[:amount]),
          :count => @assets.all.size
        })
      end
    end
  end
end
```

44:F:\git\coin\exchange\peatio-master\app\controllers\admin\statistic\withdraws_controller.rb

```
module Admin
  module Statistic
    class WithdrawsController < BaseController
      def show
        @withdraws_grid = ::Statistic::WithdrawsGrid.new(params[:statistic_withdraws_grid])
        @assets = @withdraws_grid.assets

        @groups = {
          :count => @assets.all.size,
          :amount => @assets.sum(:amount),
          :fee => @assets.sum(:fee)
        }
      end
    end
  end
end
```

45:F:\git\coin\exchange\peatio-master\app\controllers\admin\tickets_controller.rb

```
module Admin
  class TicketsController < BaseController

    def index
      @tickets = Ticket.order("created_at DESC")
      @tickets = params[:closed].nil? ? @tickets.open : @tickets.closed
    end

    def show
      @comments = ticket.comments
      @comments.unread_by(current_user).each do |c|
        c.mark_as_read! for: current_user
      end
      @comment = Comment.new
      ticket.mark_as_read!(for: current_user) if ticket.unread?(current_user)
    end

    def close
      flash[:notice] = I18n.t('private.tickets.close_succ') if ticket.close!
      redirect_to admin_tickets_path
    end

    protected

    def ticket
      @ticket ||= Ticket.find(params[:id])
    end

  end
```

```
end

46:F:\git\coin\exchange\peatio-master\app\controllers\admin\two_factors_controller.rb
module Admin
  class TwoFactorsController < BaseController
    load_and_authorize_resource

    def destroy
      @two_factor.deactive!

      redirect_to :back
    end
  end
end

47:F:\git\coin\exchange\peatio-master\app\controllers\admin\withdraws\banks_controller.rb
module Admin
  module Withdraws
    class BanksController < ::Admin::Withdraws::BaseController
      load_and_authorize_resource :class => '::Withdraws::Bank'

      def index
        start_at = DateTime.now.ago(60 * 60 * 24)
        @one_banks = @banks.with_aasm_state(:accepted, :processing).order("id DESC")
        @all_banks = @banks.without_aasm_state(:accepted, :processing).where('created_at > ?',
start_at).order("id DESC")
      end

      def show
      end

      def update
        if @bank.may_process?
          @bank.process!
        elsif @bank.may_succeed?
          @bank.succeed!
        end

        redirect_to :back, notice: t('.notice')
      end
      def destroy
        @bank.reject!
        redirect_to :back, notice: t('.notice')
      end
    end
  end
end
```

48:F:\git\coin\exchange\peatio-master\app\controllers\admin\withdraws\base_controller.rb

```
module Admin
  module Withdraws
    class BaseController < ::Admin::BaseController
      before_action :find_withdraw, only: [:show, :update, :destroy]

      def channel
        @channel ||= WithdrawChannel.find_by_key(self.controller_name.singularize)
      end

      def kls
        channel.kls
      end

      def find_withdraw
        w = channel.kls.find(params[:id])
        self.instance_variable_set("@#{self.controller_name.singularize}", w)
        if w.may_process? and (w.amount > w.account.locked)
          flash[:alert] = 'TECH ERROR !!!!'
          redirect_to action: :index
        end
      end
    end
  end
end
```

49:F:\git\coin\exchange\peatio-master\app\controllers\admin\withdraws\satoshis_controller.rb

```
module Admin
  module Withdraws
    class SatoshisController < ::Admin::Withdraws::BaseController
      load_and_authorize_resource :class => '::Withdraws::Satoshi'

      def index
        start_at = DateTime.now.ago(60 * 60 * 24)
        @one_satoshis = @satoshis.with_aasm_state(:accepted).order("id DESC")
        @all_satoshis = @satoshis.without_aasm_state(:accepted).where('created_at > ?',
start_at).order("id DESC")
      end

      def show
      end

      def update
        @satoshi.process!
        redirect_to :back, notice: t('.notice')
      end

      def destroy
        @satoshi.reject!
        redirect_to :back, notice: t('.notice')
      end
    end
  end
end
```

50:F:\git\coin\exchange\peatio-master\app\controllers\application_controller.rb

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception

  helper_method :current_user, :is_admin?, :current_market, :gon
  before_action :set_timezone, :set_gon
  after_action :allow_iframe
  after_action :set_csrf_cookie_for_ng
  rescue_from CoinRPC::ConnectionRefusedError, with: :coin_rpc_connection_refused

  private

  include SimpleCaptcha::ControllerHelpers
  include TwoFactorHelper

  def currency
    "#{params[:ask]}#{params[:bid]}".to_sym
  end

  def current_market
    @current_market ||= Market.find_by_id(params[:market]) ||
Market.find_by_id(cookies[:market_id]) || Market.first
  end

  def redirect_back_or_settings_page
    if cookies[:redirect_to].present?
      redirect_to cookies[:redirect_to]
      cookies[:redirect_to] = nil
    else
      redirect_to settings_path
    end
  end

  def current_user
    @current_user ||= Member.current = Member.enabled.where(id: session[:member_id]).first
  end

  def auth_member!
    unless current_user
      set_redirect_to
      redirect_to root_path, alert: t('activations.new.login_required')
    end
  end

  def auth_activated!
    redirect_to settings_path, alert: t('private.settings.index.auth-activated') unless
current_user.activated?
  end

  def auth_verified!
    unless current_user and current_user.id_document and current_user.id_document_verified?
      redirect_to settings_path, alert: t('private.settings.index.auth-verified')
    end
  end

  def auth_no_initial!
  end

  def auth_anybody!
    redirect_to root_path if current_user
  end

  def auth_admin!
    redirect_to main_app.root_path unless is_admin?
  end

  def is_admin?
    current_user && current_user.admin?
  end

  def two_factor_activated!
    if not current_user.two_factors.activated?
      redirect_to settings_path, alert: t('two_factors.auth.please_active_two_factor')
    end
  end

  def two_factor_auth_verified?
    return false if not current_user.two_factors.activated?
    return false if two_factor_failed_locked? && !simple_captcha_valid?

    two_factor = current_user.two_factors.by_type(params[:two_factor][:type])
    return false if not two_factor

    two_factor.assign_attributes params.require(:two_factor).permit(:otp)
    if two_factor.verify?
      clear_two_factor_auth_failed
      true
    else
      increase_two_factor_auth_failed
      false
    end
  end

  def two_factor_failed_locked?
    failed_two_factor_auth > 10
  end

  def failed_two_factor_auth
    Rails.cache.read(failed_two_factor_auth_key) || 0
  end

  def failed_two_factor_auth_key
    "peatio:session:#{request.ip}:failed_two_factor_auths"
  end
```

```
def increase_two_factor_auth_failed
  Rails.cache.write(failed_two_factor_auth_key, failed_two_factor_auth+1, expires_in: 1.month)
end

def clear_two_factor_auth_failed
  Rails.cache.delete failed_two_factor_auth_key
end

def set_timezone
  Time.zone = ENV['TIMEZONE'] if ENV['TIMEZONE']
end

def set_gon
  gon.env = Rails.env
  gon.local = I18n.locale
  gon.market = current_market.attributes
  gon.ticker = current_market.ticker
  gon.markets = Market.to_hash

  gon.pusher = {
    key:       ENV['PUSHER_KEY'],
    wsHost:    ENV['PUSHER_HOST']      || 'ws.pusherapp.com',
    wsPort:    ENV['PUSHER_WS_PORT']   || '80',
    wssPort:   ENV['PUSHER_WSS_PORT']  || '443',
    encrypted: ENV['PUSHER_ENCRYPTED'] == 'true'
  }

  gon.clipboard = {
    :click => I18n.t('actions.clipboard.click'),
    :done => I18n.t('actions.clipboard.done')
  }

  gon.i18n = {
    brand: I18n.t('gon.brand'),
    ask: I18n.t('gon.ask'),
    bid: I18n.t('gon.bid'),
    cancel: I18n.t('actions.cancel'),
    latest_trade: I18n.t('private.markets.order_book.latest_trade'),
    switch: {
      notification: I18n.t('private.markets.settings.notification'),
      sound: I18n.t('private.markets.settings.sound')
    },
  notification: {
    title: I18n.t('gon.notification.title'),
    enabled: I18n.t('gon.notification.enabled'),
    new_trade: I18n.t('gon.notification.new_trade')
  },
  time: {
    minute: I18n.t('chart.minute'),
    hour: I18n.t('chart.hour'),
    day: I18n.t('chart.day'),
    week: I18n.t('chart.week'),
    month: I18n.t('chart.month'),
    year: I18n.t('chart.year')
  },
  chart: {
    price: I18n.t('chart.price'),
    volume: I18n.t('chart.volume'),
    open: I18n.t('chart.open'),
    high: I18n.t('chart.high'),
    low: I18n.t('chart.low'),
    close: I18n.t('chart.close'),
    candlestick: I18n.t('chart.candlestick'),
    line: I18n.t('chart.line'),
    zoom: I18n.t('chart.zoom'),
    depth: I18n.t('chart.depth'),
    depth_title: I18n.t('chart.depth_title')
  },
  place_order: {
    confirm_submit: I18n.t('private.markets.show.confirm'),
    confirm_cancel: I18n.t('private.markets.show.cancel_confirm'),
    price: I18n.t('private.markets.place_order.price'),
    volume: I18n.t('private.markets.place_order.amount'),
    sum: I18n.t('private.markets.place_order.total'),
    price_high: I18n.t('private.markets.place_order.price_high'),
    price_low: I18n.t('private.markets.place_order.price_low'),
    full_bid: I18n.t('private.markets.place_order.full_bid'),
    full_ask: I18n.t('private.markets.place_order.full_ask')
  },
  trade_state: {
    new: I18n.t('private.markets.trade_state.new'),
    partial: I18n.t('private.markets.trade_state.partial')
  }
}
```

```
    gon.currencies = Currency.all.inject({}) do |memo, currency|
      memo[currency.code] = {
        code: currency[:code],
        symbol: currency[:symbol],
        isCoin: currency[:coin]
      }
      memo
    end
    gon.fiat_currency = Currency.first.code

    gon.tickers = {}
    Market.all.each do |market|
      gon.tickers[market.id] = market.unit_info.merge(Global[market.id].ticker)
    end

    if current_user
      gon.current_user = { sn: current_user.sn }
      gon.accounts = current_user.accounts.inject({}) do |memo, account|
        memo[account.currency] = {
          currency: account.currency,
          balance: account.balance,
          locked: account.locked
        } if account.currency_obj.try(:visible)
        memo
      end
    end
  end

  def coin_rpc_connection_refused
    render 'errors/connection'
  end

  def save_session_key(member_id, key)
    Rails.cache.write "peatio:sessions:#{member_id}:#{key}", 1, expire_after:
ENV['SESSION_EXPIRE'].to_i.minutes
  end

  def clear_all_sessions(member_id)
    if redis = Rails.cache.instance_variable_get(:@data)
      redis.keys("peatio:sessions:#{member_id}:*").each {|k| Rails.cache.delete k.split(':').last }
    end
    Rails.cache.delete_matched "peatio:sessions:#{member_id}:*"
  end

  def allow_iframe
    response.headers.except! 'X-Frame-Options' if Rails.env.development?
  end

  def set_csrf_cookie_for_ng
    cookies['XSRF-TOKEN'] = form_authenticity_token if protect_against_forgery?
  end

  def verified_request?
    super || form_authenticity_token == request.headers['X-XSRF-TOKEN']
  end

end
```

51:F:\git\coin\exchange\peatio-master\app\controllers\authentications\emails_controller.rb

```
module Authentications
  class EmailsController < ApplicationController
    before_action :auth_member!
    before_action :check_email_present

    def new
      flash.now[:info] = t('.setup_email')
    end

    def create
      if current_user.update_attributes(email: params[:email][:address])
        redirect_to settings_path
      else
        flash.now[:alert] = current_user.errors.full_messages.join(',')
        render :new
      end
    end

    private
    def check_email_present
      redirect_to settings_path if current_user.email.present?
    end
  end
```

```
end

52:F:\git\coin\exchange\peatio-master\app\controllers\authentications\identities_controller.rb
module Authentications
  class IdentitiesController < ApplicationController
    before_action :auth_member!

    def new
      @identity = Identity.new(email: current_user.email)
    end

    def create
      identity = Identity.new(identity_params.merge(email: current_user.email))
      if identity.save && current_user.create_auth_for_identity(identity)
        redirect_to settings_path, notice: t('.success')
      else
        redirect_to new_authentications_identity_path, alert: identity.errors.full_messages.join(',')
      end
    end

    private

    def identity_params
      params.required(:identity).permit(:password, :password_confirmation)
    end

  end
end

53:F:\git\coin\exchange\peatio-
master\app\controllers\authentications\weibo_accounts_controller.rb
module Authentications
  class WeiboAccountsController < ApplicationController
    before_action :auth_member!

    def destroy
      if current_user.authentications.count <= 1
        flash[:alert] = t("authentications.weibo.destroy.last_auth_alert")
      else
        if current_user.remove_auth('weibo')
          flash[:notice] = t("authentications.weibo.destroy.unbind_success")
        end
      end
      redirect_to settings_path
    end
  end
end
```

54:F:\git\coin\exchange\peatio-master\app\controllers\concerns\deposits\ctrl_bankable.rb

```
module Deposits
  module CtrlBankable
    extend ActiveSupport::Concern

    included do
      before_filter :fetch
    end

    def create
      @deposit = model_kls.new(deposit_params)

      if @deposit.save
        render nothing: true
      else
        render text: @deposit.errors.full_messages.join, status: 403
      end
    end

    def destroy
      @deposit = current_user.deposits.find(params[:id])
      @deposit.cancel!
      render nothing: true
    end

    private

    def fetch
      @account = current_user.get_account(channel.currency)
      @model = model_kls
      @fund_sources = current_user.fund_sources.with_currency(channel.currency)
      @assets = model_kls.where(member: current_user).order(:id).reverse_order.limit(10)
    end

    def deposit_params
```

```
      params[:deposit][:currency] = channel.currency
      params[:deposit][:member_id] = current_user.id
      params[:deposit][:account_id] = @account.id
      params.require(:deposit).permit(:fund_source, :amount, :currency, :account_id, :member_id)
    end
  end
end
```

55:F:\git\coin\exchange\peatio-master\app\controllers\concerns\deposits\ctrl_coinable.rb

```
module Deposits
  module CtrlCoinable
    extend ActiveSupport::Concern

    def gen_address
      account = current_user.get_account(channel.currency)
      if !account.payment_address.transactions.empty?
        @address = account.payment_addresses.create currency: account.currency
        @address.gen_address if @address.address.blank?
        render nothing: true
      else
        render text: t('.require_transaction'), status: 403
      end

    end

  end
end
```

56:F:\git\coin\exchange\peatio-master\app\controllers\concerns\order_creation.rb

```
module Concerns
  module OrderCreation
    extend ActiveSupport::Concern

    def order_params(order)
      params[order][:bid] = params[:bid]
      params[order][:ask] = params[:ask]
      params[order][:state] = Order::WAIT
      params[order][:currency] = params[:market]
      params[order][:member_id] = current_user.id
      params[order][:volume] = params[order][:origin_volume]
      params[order][:source] = 'Web'
      params.require(order).permit(
        :bid, :ask, :currency, :price, :source,
        :state, :origin_volume, :volume, :member_id, :ord_type)
    end

    def order_submit
      begin
        Ordering.new(@order).submit
        render status: 200, json: success_result
      rescue
        Rails.logger.warn "Member id=#{current_user.id} failed to submit order: #{$!}"
        Rails.logger.warn params.inspect
        Rails.logger.warn $!.backtrace[0,20].join("\n")
        render status: 500, json: error_result(@order.errors)
      end
    end

    def success_result
      Jbuilder.encode do |json|
        json.result true
        json.message I18n.t("private.markets.show.success")
      end
    end

    def error_result(args)
      Jbuilder.encode do |json|
        json.result false
        json.message I18n.t("private.markets.show.error")
        json.errors args
      end
    end
  end
end
```

57:F:\git\coin\exchange\peatio-master\app\controllers\concerns\token_management.rb

```
module Concerns
  module TokenManagement
    extend ActiveSupport::Concern

    def token_required
      if not @token = Token.available.with_token(params[:token] || params[:id]).first
        redirect_to root_path, :alert => t('.alert')
      end
    end

    alias :'token_required!' :'token_required'
  end
end

58:F:\git\coin\exchange\peatio-master\app\controllers\concerns\withdraws\withdrawable.rb
module Withdraws
  module Withdrawable
    extend ActiveSupport::Concern

    included do
      before_filter :fetch
    end

    def create
      @withdraw = model_kls.new(withdraw_params)

      if two_factor_auth_verified?
        if @withdraw.save
          @withdraw.submit!
          render nothing: true
        else
          render text: @withdraw.errors.full_messages.join(', '), status: 403
        end
      else
        render text: I18n.t('private.withdraws.create.two_factors_error'), status: 403
      end
    end

    def destroy
      Withdraw.transaction do
        @withdraw = current_user.withdraws.find(params[:id]).lock!
        @withdraw.cancel
        @withdraw.save!
      end
      render nothing: true
    end

    private

    def fetch
```

```
      @account = current_user.get_account(channel.currency)
      @model = model_kls
      @fund_sources = current_user.fund_sources.with_currency(channel.currency)
      @assets = model_kls.without_aasm_state(:submitting).where(member:
current_user).order(:id).reverse_order.limit(10)
    end

    def withdraw_params
      params[:withdraw][:currency] = channel.currency
      params[:withdraw][:member_id] = current_user.id
      params.require(:withdraw).permit(:fund_source_id, :member_id, :currency, :sum)
    end

  end
end
```

59:F:\git\coin\exchange\peatio-master\app\controllers\documents_controller.rb

```
class DocumentsController < ApplicationController

  def show
    @doc = Document.find_by_key(params[:id])
    raise ActiveRecord::RecordNotFound unless @doc

    if @doc.is_auth and !current_user
      redirect_to root_path, alert: t('activations.new.login_required')
    end
  end

  def api_v2
    render 'api_v2', layout: 'api_v2'
  end

  def websocket_api
    render 'websocket_api', layout: 'api_v2'
  end

  def oauth
    render 'oauth', layout: 'api_v2'
  end

end
```

```
60:F:\git\coin\exchange\peatio-master\app\controllers\identities_controller.rb
class IdentitiesController < ApplicationController
  before_filter :auth_anybody!, only: :new

  def new
    @identity = env['omniauth.identity'] || Identity.new
  end

  def edit
    @identity = current_user.identity
  end

  def update
    @identity = current_user.identity

    unless @identity.authenticate(params[:identity][:old_password])
      redirect_to edit_identity_path, alert: t('.auth-error') and return
    end

    if @identity.authenticate(params[:identity][:password])
      redirect_to edit_identity_path, alert: t('.auth-same') and return
    end

    if @identity.update_attributes(identity_params)
      current_user.send_password_changed_notification
      clear_all_sessions current_user.id
      reset_session
      redirect_to signin_path, notice: t('.notice')
    else
      render :edit
    end
  end

  private
  def identity_params
    params.required(:identity).permit(:password, :password_confirmation)
  end
end

61:F:\git\coin\exchange\peatio-master\app\controllers\members_controller.rb
class MembersController < ApplicationController
  before_filter :auth_member!
  before_filter :auth_no_initial!

  def edit
    @member = current_user
  end

  def update
    @member = current_user

    if @member.update_attributes(member_params)
      redirect_to forum_path
    else
      render :edit
    end
  end

  private
  def member_params
    params.required(:member).permit(:display_name)
  end
end
```

62:F:\git\coin\exchange\peatio-master\app\controllers\private\account_versions_controller.rb

```
module Private
  class AccountVersionsController < BaseController
    def index
      @account_versions_grid = AccountVersionsGrid.new(params[:account_versions_grid]) do
|scope|
        scope.where(:member_id => current_user.id)
      end
      @assets = @account_versions_grid.assets.page(params[:page]).per(20)
    end
  end
end
```

63:F:\git\coin\exchange\peatio-master\app\controllers\private\api_tokens_controller.rb

```
module Private
  class APITokensController < BaseController
    before_action :auth_activated!
    before_action :auth_verified!
    before_action :two_factor_activated!
```

```
def index
  @tokens = current_user.api_tokens.user_requested
  @oauth_api_tokens = current_user.api_tokens.oauth_requested

  ids = Doorkeeper::AccessToken
    .where(id: @oauth_api_tokens.map(&:oauth_access_token_id))
    .group(:application_id).select('max(id) as id')
  @oauth_access_tokens = Doorkeeper::AccessToken.where(id: ids).includes(:application)
end

def new
  @token = current_user.api_tokens.build
end

def create
  @token = current_user.api_tokens.build api_token_params
  @token.scopes = 'all'

  if !two_factor_auth_verified?
    flash.now[:alert] = t('.alert_two_factor')
    render :new and return
  end

  if @token.save
    flash.now[:notice] = t('.success')
  else
    flash.now[:alert] = t('.failed')
    render :new
  end
end

def edit
  @token = current_user.api_tokens.user_requested.find params[:id]
end

def update
  @token = current_user.api_tokens.user_requested.find params[:id]

  if !two_factor_auth_verified?
    flash.now[:alert] = t('.alert_two_factor')
    render :edit and return
  end
```

```
      if @token.update_attributes(api_token_params)
        flash.now[:notice] = t('.success')
      else
        flash.now[:alert] = t('.failed')
      end

      render :edit
    end

    def destroy
      @token = current_user.api_tokens.user_requested.find params[:id]
      if @token.destroy
        redirect_to url_for(action: :index), notice: t('.success')
      else
        redirect_to url_for(action: :index), notice: t('.failed')
      end
    end

    def unbind
      Doorkeeper::AccessToken.revoke_all_for(params[:id], current_user)
      redirect_to url_for(action: :index), notice: t('.success')
    end

    private

    def api_token_params
      params.require(:api_token).permit(:label, :ip_whitelist)
    end

  end
end
```

64:F:\git\coin\exchange\peatio-master\app\controllers\private\assets_controller.rb

```
module Private
  class AssetsController < BaseController
    skip_before_action :auth_member!, only: [:index]

    def index
      @cny_assets  = Currency.assets('cny')
      @btc_proof   = Proof.current :btc
      @cny_proof   = Proof.current :cny
      if current_user
        @btc_account = current_user.accounts.with_currency(:btc).first
        @cny_account = current_user.accounts.with_currency(:cny).first
      end
    end

    def partial_tree
      account    = current_user.accounts.with_currency(params[:id]).first
      @timestamp = Proof.with_currency(params[:id]).last.timestamp
      @json      = account.partial_tree.to_json.html_safe
      respond_to do |format|
        format.js
      end
    end

  end
end
```

65:F:\git\coin\exchange\peatio-master\app\controllers\private\base_controller.rb

```
module Private
  class BaseController < ::ApplicationController
    before_action :check_email_nil
    before_filter :no_cache, :auth_member!

    private

    def no_cache
      response.headers["Cache-Control"] = "no-cache, no-store, max-age=0, must-revalidate"
      response.headers["Pragma"] = "no-cache"
      response.headers["Expires"] = "Sat, 03 Jan 2009 00:00:00 GMT"
    end

    def check_email_nil
      redirect_to new_authentications_email_path if current_user && current_user.email.nil?
    end

  end
end
```

66:F:\git\coin\exchange\peatio-master\app\controllers\private\comments_controller.rb

```
module Private
```

```
  class CommentsController < BaseController

    def create
      comment = ticket.comments.new(comment_params.merge(author_id: current_user.id))

      if comment.save
        flash[:notice] = I18n.t("private.tickets.comment_succ")
      else
        flash[:alert] = I18n.t("private.tickets.comment_fail")
      end
      redirect_to ticket_path(ticket)
    end

    private

    def comment_params
      params.required(:comment).permit(:content)
    end

    def ticket
      @ticket ||= current_user.tickets.find(params[:ticket_id])
    end

  end
end
```

67:F:\git\coin\exchange\peatio-master\app\controllers\private\deposits\banks_controller.rb

```
module Private
  module Deposits
    class BanksController < ::Private::Deposits::BaseController
      include ::Deposits::CtrlBankable
    end
  end
end
```

68:F:\git\coin\exchange\peatio-master\app\controllers\private\deposits\base_controller.rb

```
module Private
  module Deposits
    class BaseController < ::Private::BaseController
      layout 'app'
      before_action :channel
      before_action :auth_activated!
      before_action :auth_verified!

      def channel
        @channel ||= DepositChannel.find_by_key(self.controller_name.singularize)
      end

      def model_kls
        "deposits/#{self.controller_name.singularize}".camelize.constantize
      end
    end
  end
end
```

69:F:\git\coin\exchange\peatio-master\app\controllers\private\deposits\satoshis_controller.rb

```
module Private
  module Deposits
    class SatoshisController < ::Private::Deposits::BaseController
      include ::Deposits::CtrlCoinable
    end
  end
end
```

70:F:\git\coin\exchange\peatio-master\app\controllers\private\funds_controller.rb

```
module Private
  class FundsController < BaseController
    layout 'funds'

    before_action :auth_activated!
    before_action :auth_verified!
    before_action :two_factor_activated!

    def index
      @deposit_channels = DepositChannel.all
      @withdraw_channels = WithdrawChannel.all
      @currencies = Currency.all.sort
      @deposits = current_user.deposits
      @accounts = current_user.accounts.enabled
      @withdraws = current_user.withdraws
      @fund_sources = current_user.fund_sources
      @banks = Bank.all
      gon.jbuilder
    end

    def gen_address
      current_user.accounts.each do |account|
        next if not account.currency_obj.coin?

        if account.payment_addresses.blank?
          account.payment_addresses.create(currency: account.currency)
        else
          address = account.payment_addresses.last
          address.gen_address if address.address.blank?
        end
      end
      render nothing: true
    end

  end
end
```

71:F:\git\coin\exchange\peatio-master\app\controllers\private\fund_sources_controller.rb

```
module Private
  class FundSourcesController < BaseController

    def create
      new_fund_source = current_user.fund_sources.new fund_source_params

      if new_fund_source.save
        render json: new_fund_source, status: :ok
      else
        head :bad_request
      end
    end

    def update
      account = current_user.accounts.with_currency(fund_source.currency).first
      account.update default_withdraw_fund_source_id: params[:id]

      head :ok
    end
    def destroy
      render json: fund_source.destroy, status: :ok
    end

    private

    def fund_source
      current_user.fund_sources.find(params[:id])
    end

    def fund_source_params
      params.require(:fund_source).permit(:currency, :uid, :extra)
    end
  end
end
```

72:F:\git\coin\exchange\peatio-master\app\controllers\private\history_controller.rb

```
module Private
  class HistoryController < BaseController

    helper_method :tabs

    def account
      @market = current_market

      @deposits = Deposit.where(member: current_user).with_aasm_state(:accepted)
      @withdraws = Withdraw.where(member: current_user).with_aasm_state(:done)

      @transactions = (@deposits + @withdraws).sort_by {|t| -t.created_at.to_i }
      @transactions = Kaminari.paginate_array(@transactions).page(params[:page]).per(20)
    end

    def trades
      @trades = current_user.trades
        .includes(:ask_member).includes(:bid_member)
        .order('id desc').page(params[:page]).per(20)
    end

    def orders
      @orders = current_user.orders.includes(:trades).order("id desc").page(params[:page]).per(20)
    end
    private

    def tabs
      { order: ['header.order_history', order_history_path],
        trade: ['header.trade_history', trade_history_path],
        account: ['header.account_history', account_history_path] }
    end

  end
end
```

73:F:\git\coin\exchange\peatio-master\app\controllers\private\id_documents_controller.rb

```
module Private
  class IdDocumentsController < BaseController

    def edit
      @id_document = current_user.id_document || current_user.create_id_document
    end

    def update
      @id_document = current_user.id_document

      if @id_document.update_attributes id_document_params
        @id_document.submit! if @id_document.unverified?

        redirect_to settings_path, notice: t('.notice')
      else
        render :edit
      end
    end

    private

    def id_document_params
      params.require(:id_document).permit(:name, :birth_date, :address, :city, :country, :zipcode,
                                          :id_document_type, :id_document_number, :id_bill_type,
                                          {id_document_file_attributes: [:id, :file]},
                                          {id_bill_file_attributes: [:id, :file]})
    end
  end
end
```

74:F:\git\coin\exchange\peatio-master\app\controllers\private\markets_controller.rb

```
module Private
  class MarketsController < BaseController
    skip_before_action :auth_member!, only: [:show]
    before_action :visible_market?
    after_action :set_default_market

    layout false

    def show
      @bid = params[:bid]
      @ask = params[:ask]

      @market        = current_market
      @markets       = Market.all.sort
      @market_groups = @markets.map(&:quote_unit).uniq

      @bids   = @market.bids
      @asks   = @market.asks
      @trades = @market.trades

      # default to limit order
      @order_bid = OrderBid.new ord_type: 'limit'
      @order_ask = OrderAsk.new ord_type: 'limit'

      set_member_data if current_user
      gon.jbuilder
    end

    private

    def visible_market?
      redirect_to market_path(Market.first) if not current_market.visible?
    end

    def set_default_market
      cookies[:market_id] = @market.id
    end

    def set_member_data
      @member = current_user
      @orders_wait = @member.orders.with_currency(@market).with_state(:wait)
      @trades_done = Trade.for_member(@market.id, current_user, limit: 100, order: 'id desc')
    end

  end
end

75:F:\git\coin\exchange\peatio-master\app\controllers\private\orders_controller.rb
module Private
  class OrdersController < BaseController

    def destroy
      ActiveRecord::Base.transaction do
        order = current_user.orders.find(params[:id])
        ordering = Ordering.new(order)

        if ordering.cancel
          render status: 200, nothing: true
        else
          render status: 500, nothing: true
        end
      end
    end

    def clear
      @orders = current_user.orders.with_currency(current_market).with_state(:wait)
      Ordering.new(@orders).cancel
      render status: 200, nothing: true
    end

  end
end

76:F:\git\coin\exchange\peatio-master\app\controllers\private\order_asks_controller.rb
module Private
  class OrderAsksController < BaseController
    include Concerns::OrderCreation

    def create
      @order = OrderAsk.new(order_params(:order_ask))
      order_submit
    end
    def clear
      @orders = OrderAsk.where(member_id:
current_user.id).with_state(:wait).with_currency(current_market)
      Ordering.new(@orders).cancel
      render status: 200, nothing: true
    end

  end
end

77:F:\git\coin\exchange\peatio-master\app\controllers\private\order_bids_controller.rb
module Private
  class OrderBidsController < BaseController
    include Concerns::OrderCreation

    def create
      @order = OrderBid.new(order_params(:order_bid))
      order_submit
    end

    def clear
      @orders = OrderBid.where(member_id:
current_user.id).with_state(:wait).with_currency(current_market)
      Ordering.new(@orders).cancel
      render status: 200, nothing: true
    end

  end
end

78:F:\git\coin\exchange\peatio-master\app\controllers\private\payment_addresses_controller.rb
module Private
  class PaymentAddressesController < BaseController
    def update
      account = current_user.get_account(params[:currency])
      payment_address = account.payment_addresses.using
      unless payment_address.transactions.empty?
        account.gen_payment_address
      end
      redirect_to funds_path
    end
  end
```

```
end


79:F:\git\coin\exchange\peatio-master\app\controllers\private\pusher_controller.rb
require "openssl"

module Private
  class PusherController < BaseController
    protect_from_forgery :except => :auth

    def auth
      sn = params[:channel_name].split('-', 2).last
      if current_user && current_user.sn == sn
        response = Pusher[params[:channel_name]].authenticate(params[:socket_id])
        render :json => response
      else
        render :text => "Forbidden", :status => '403'
      end
    end
  end
end

80:F:\git\coin\exchange\peatio-master\app\controllers\private\settings_controller.rb
module Private
  class SettingsController < BaseController
    def index
      unless current_user.activated?
        flash.now[:info] = t('.activated')
      end
    end
  end
end


81:F:\git\coin\exchange\peatio-master\app\controllers\private\tickets_controller.rb
module Private
  class TicketsController < BaseController
    after_filter :mark_ticket_as_read, only: [:create, :show]

    def index
      @tickets = current_user.tickets
      @tickets = params[:closed].nil? ? @tickets.open : @tickets.closed
      redirect_to new_ticket_path if @tickets.empty?
    end

    def new
      @ticket = Ticket.new
    end

    def create
      @ticket = current_user.tickets.create(ticket_params)
      if @ticket.save
        flash[:notice] = I18n.t('private.tickets.ticket_create_succ')
        redirect_to tickets_path
      else
        flash[:alert] = I18n.t('private.tickets.ticket_create_fail')
        render :new
      end
    end

    def show
      @comments = ticket.comments
      @comments.unread_by(current_user).each do |c|
        c.mark_as_read! for: current_user
      end
      @comment = Comment.new
    end

    def close
      flash[:notice] = I18n.t('private.tickets.close_succ') if ticket.close!
      redirect_to tickets_path
    end

    private

    def ticket_params
      params.required(:ticket).permit(:title, :content)
    end

    def ticket
      @ticket ||= current_user.tickets.find(params[:id])
    end

    def mark_ticket_as_read
```

```
      ticket.mark_as_read!(for: current_user) if ticket.unread?(current_user)
    end
  end
end

82:F:\git\coin\exchange\peatio-master\app\controllers\private\trade_states_controller.rb
module Private
  class TradeStatesController < BaseController
    def show
      @member = current_user
      @ask_account = @member.get_account params[:ask]
      @bid_account = @member.get_account params[:bid]
    end
  end
end

83:F:\git\coin\exchange\peatio-master\app\controllers\private\withdraws\banks_controller.rb
module Private::Withdraws
  class BanksController < ::Private::Withdraws::BaseController
    include ::Withdraws::Withdrawable
  end
end

84:F:\git\coin\exchange\peatio-master\app\controllers\private\withdraws\base_controller.rb
module Private
  module Withdraws
    class BaseController < ::Private::BaseController
      before_action :channel
      before_action :auth_activated!
      before_action :auth_verified!
      before_action :two_factor_activated!

      def channel
        @channel ||= WithdrawChannel.find_by_key(self.controller_name.singularize)
      end

      def model_kls
        "withdraws/#{self.controller_name.singularize}".camelize.constantize
      end

    end
  end
```

```
end

85:F:\git\coin\exchange\peatio-master\app\controllers\private\withdraws\satoshis_controller.rb
module Private::Withdraws
  class SatoshisController < ::Private::Withdraws::BaseController
    include ::Withdraws::Withdrawable
  end
end

86:F:\git\coin\exchange\peatio-master\app\controllers\reset_passwords_controller.rb
class ResetPasswordsController < ApplicationController
  include Concerns::TokenManagement

  before_action :auth_anybody!
  before_action :token_required, :only => [:edit, :update]

  def new
    @token = Token::ResetPassword.new
  end

  def create
    @token = Token::ResetPassword.new(reset_password_params)

    if @token.save
      clear_all_sessions @token.member_id
      redirect_to signin_path, notice: t('.success')
    else
      redirect_to url_for(action: :new), alert: @token.errors.full_messages.join(', ')
    end
  end

  def edit
  end

  def update
    if @token.update_attributes(reset_password_update_params)
      @token.confirm!
      redirect_to signin_path, notice: t('.success')
    else
      render :edit
    end
  end
  private
  def reset_password_params
    params.required(:reset_password).permit(:email)
  end

  def reset_password_update_params
    params.required(:reset_password).permit(:password)
  end
end

87:F:\git\coin\exchange\peatio-master\app\controllers\sessions_controller.rb
class SessionsController < ApplicationController

  skip_before_action :verify_authenticity_token, only: [:create]

  before_action :auth_member!, only: :destroy
  before_action :auth_anybody!, only: [:new, :failure]
  before_action :add_auth_for_weibo

  helper_method :require_captcha?

  def new
    @identity = Identity.new
  end

  def create
    if !require_captcha? || simple_captcha_valid?
      @member = Member.from_auth(auth_hash)
    end

    if @member
      if @member.disabled?
        increase_failed_logins
        redirect_to signin_path, alert: t('.disabled')
      else
        clear_failed_logins
        reset_session rescue nil
        session[:member_id] = @member.id
        save_session_key @member.id, cookies['_peatio_session']
        save_signup_history @member.id
        MemberMailer.notify_signin(@member.id).deliver if @member.activated?
        redirect_back_or_settings_page
      end
    else
      increase_failed_logins
      redirect_to signin_path, alert: t('.error')
    end
  end

  def failure
    increase_failed_logins
    redirect_to signin_path, alert: t('.error')
  end

  def destroy
    clear_all_sessions current_user.id
    reset_session
    redirect_to root_path
  end

  private

  def require_captcha?
    failed_logins > 3
  end

  def failed_logins
    Rails.cache.read(failed_login_key) || 0
  end

  def increase_failed_logins
    Rails.cache.write(failed_login_key, failed_logins+1)
  end

  def clear_failed_logins
    Rails.cache.delete failed_login_key
  end

  def failed_login_key
    "peatio:session:#{request.ip}:failed_logins"
  end

  def auth_hash
```

```
    @auth_hash ||= env["omniauth.auth"]
  end

  def add_auth_for_weibo
    if current_user && ENV['WEIBO_AUTH'] == "true" && auth_hash.try(:[], :provider) == 'weibo'
      redirect_to settings_path, notice: t('.weibo_bind_success') if
current_user.add_auth(auth_hash)
    end
  end

  def save_signup_history(member_id)
    SignupHistory.create(
      member_id: member_id,
      ip: request.ip,
      accept_language: request.headers["Accept-Language"],
      ua: request.headers["User-Agent"]
    )
  end

end

88:F:\git\coin\exchange\peatio-master\app\controllers\two_factors_controller.rb
class TwoFactorsController < ApplicationController
  before_action :auth_member!
  before_action :two_factor_required!

  def show
    respond_to do |format|
      if require_send_sms_verify_code?
        send_sms_verify_code
        format.any { render status: :ok, nothing: true }
      elsif two_factor_failed_locked?
        format.any { render status: :locked, inline: "<%= show_simple_captcha %>" }
      else
        format.any { render status: :ok, nothing: true }
      end
    end
  end

  def index
  end
  def update
    if two_factor_auth_verified?
      unlock_two_factor!

      redirect_to session.delete(:return_to) || settings_path
    else
      redirect_to two_factors_path, alert: t('.alert')
    end
  end

  private

  def two_factor_required!
    @two_factor ||= two_factor_by_type || first_available_two_factor

    if @two_factor.nil?
      redirect_to settings_path, alert: t('two_factors.auth.please_active_two_factor')
    end
  end

  def two_factor_by_type
    current_user.two_factors.activated.by_type(params[:id])
  end

  def first_available_two_factor
    current_user.two_factors.activated.first
  end

  def require_send_sms_verify_code?
    @two_factor.is_a?(TwoFactor::Sms) && params[:refresh]
  end

  def send_sms_verify_code
    @two_factor.refresh!
    @two_factor.send_otp
  end
end

89:F:\git\coin\exchange\peatio-master\app\controllers\verify\google_auths_controller.rb
module Verify
  class GoogleAuthsController < ApplicationController
    before_action :auth_member!
    before_action :find_google_auth
    before_action :google_auth_activated?,   only: [:show, :create]
    before_action :google_auth_inactivated?, only: [:edit, :destroy]
    before_action :two_factor_required!,     only: [:show]

    def show
      @google_auth.refresh! if params[:refresh]
    end

    def edit
    end

    def update
      if one_time_password_verified?
        @google_auth.active! and unlock_two_factor!
        redirect_to settings_path, notice: t('.notice')
      else
        redirect_to verify_google_auth_path, alert: t('.alert')
      end
    end

    def destroy
      if two_factor_auth_verified?
        @google_auth.deactive!
        redirect_to settings_path, notice: t('.notice')
      else
        redirect_to edit_verify_google_auth_path, alert: t('.alert')
      end
    end

    private

    def find_google_auth
      @google_auth ||= current_user.app_two_factor
    end

    def google_auth_params
      params.require(:google_auth).permit(:otp)
    end

    def one_time_password_verified?
      @google_auth.assign_attributes(google_auth_params)
```

```
      @google_auth.verify?
    end

    def google_auth_activated?
      redirect_to settings_path, notice: t('.notice.already_activated') if @google_auth.activated?
    end

    def google_auth_inactivated?
      redirect_to settings_path, notice: t('.notice.not_activated_yet') if not @google_auth.activated?
    end

    def two_factor_required!
      return if not current_user.sms_two_factor.activated?

      if two_factor_locked?
        session[:return_to] = request.original_url
        redirect_to two_factors_path
      end
    end

  end
end
```

90:F:\git\coin\exchange\peatio-master\app\controllers\verify\sms_auths_controller.rb

```
module Verify
  class SmsAuthsController < ApplicationController
    before_action :auth_member!
    before_action :find_sms_auth
    before_action :activated?
    before_action :two_factor_required!

    def show
      @phone_number = Phonelib.parse(current_user.phone_number).national
    end

    def update
      if params[:commit] == 'send_code'
        send_code_phase
      else
        verify_code_phase
      end
    end
```

```
private

def activated?
  if @sms_auth.activated?
    redirect_to settings_path, notice: t('.notice.already_activated')
  end
end

def find_sms_auth
  @sms_auth ||= current_user.sms_two_factor
end

def send_code_phase
  @sms_auth.send_code_phase = true
  @sms_auth.assign_attributes token_params

  respond_to do |format|
    if @sms_auth.valid?
      @sms_auth.send_otp

      text = I18n.t('verify.sms_auths.show.notice.send_code_success')
      format.any { render status: :ok, text: {text: text}.to_json }
    else
      text = @sms_auth.errors.full_messages.to_sentence
      format.any { render status: :bad_request, text: {text: text}.to_json }
    end
  end
end

def verify_code_phase
  @sms_auth.assign_attributes token_params

  respond_to do |format|
    if @sms_auth.verify?
      @sms_auth.active! and unlock_two_factor!

      text = I18n.t('verify.sms_auths.show.notice.otp_success')
      flash[:notice] = text
      format.any { render status: :ok, text: {text: text, reload: true}.to_json }
    else
      text = @sms_auth.errors.full_messages.to_sentence
```

```
          format.any { render status: :bad_request, text: {text: text}.to_json }
        end
      end
    end

    def token_params
      params.required(:sms_auth).permit(:country, :phone_number, :otp)
    end

    def two_factor_required!
      return if not current_user.app_two_factor.activated?

      if two_factor_locked?
        session[:return_to] = request.original_url
        redirect_to two_factors_path
      end
    end

  end
end

91:F:\git\coin\exchange\peatio-master\app\controllers\welcome_controller.rb
class WelcomeController < ApplicationController
  layout 'landing'

  def index
  end
end

92:F:\git\coin\exchange\peatio-master\app\grids\account_versions_grid.rb
class AccountVersionsGrid
  include Datagrid
  include Datagrid::Naming
  include Datagrid::ColumnI18n

  scope do |m|
    AccountVersion.order("id DESC")
  end

  filter(:currency, :enum, :select => Deposit.currency.value_options)
  filter(:reason, :enum, :select => AccountVersion.reason.value_options)
  column_localtime :created_at
  column :currency_text, :order => false

  column :modifiable_type, :order => false do |m|
    if m.modifiable_type
      "#{I18n.t("activerecord.models.#{m.modifiable_type.underscore}", default: m.modifiable_type)}
##{m.modifiable_id}"
    else
      'N/A'
    end
  end

  column :reason_text, :order => false
  column :out, :order => false
  column :in, :order => false
  column :amount, :order => false
  column :fee, :order => false do |m|
    if m.fee and not m.fee.zero?
      m.fee
    end
  end
end

93:F:\git\coin\exchange\peatio-master\app\grids\documents_grid.rb
class DocumentsGrid
  include Datagrid
  include Datagrid::Naming
  include Datagrid::ColumnI18n

  scope do |m|
    Document
  end

  column :key
  column :title
  column :is_auth
  column :actions, html: true, header: '' do |o|
    link_to I18n.t('actions.edit'), edit_admin_document_path(o.key)
  end
end

94:F:\git\coin\exchange\peatio-master\app\grids\proofs_grid.rb
```

```
class ProofsGrid

  include Datagrid

  scope do
    Proof.order('id desc')
  end

  filter(:id, :integer)
  filter(:created_at, :date, :range => true)

  column(:id)
  column(:currency)
  column(:balance)
  column(:sum)
  column(:created_at) do |model|
    model.created_at.to_date
  end
  column :actions, html: true, header: '' do |proof|
    link_to I18n.t('actions.edit'), edit_admin_proof_path(proof)
  end
end

95:F:\git\coin\exchange\peatio-master\app\grids\statistic\deposits_grid.rb
module Statistic
  class DepositsGrid
    include Datagrid
    include Datagrid::Naming
    include Datagrid::ColumnI18n

    scope do
      Deposit.includes(:account).order('created_at DESC')
    end

    filter(:currency, :enum, :select => Deposit.currency.value_options, :default => 1)
    filter(:created_at, :datetime, :range => true, :default => proc { [1.day.ago, Time.now]})

    column :member do |model|
      format(model) do
        link_to model.member, member_path(model.member)
      end
    end
    column :currency do
      self.account.currency_text
    end
    column(:amount)
    column(:txid) do |deposit|
      deposit.txid
    end
    column_localtime :created_at
    column(:aasm_state_text)
  end
end
```

96:F:\git\coin\exchange\peatio-master\app\grids\statistic\orders_grid.rb

```
module Statistic
  class OrdersGrid
    include Datagrid
    include Datagrid::Naming
    include Datagrid::ColumnI18n

    scope do
      Order.order('created_at DESC')
    end

    filter(:currency, :enum, :select => Order.currency.value_options, :default => 3, :include_blank =>
false)
    filter(:state, :enum, :select => Order.state.value_options)
    filter(:type, :enum, :select => [[OrderBid.model_name.human, OrderBid.model_name],
[OrderAsk.model_name.human, OrderAsk.model_name]])
    filter(:created_at, :datetime, :range => true, :default => proc { [7.day.ago, Time.now]})

    column(:member_id) do |model|
      format(model) do
        link_to model.member.name, member_path(model.member.id)
      end
    end
    column(:id, :order => nil)
    column(:price)
    column(:volume) do |o|
      if o.volume == o.origin_volume or o.volume.zero?
        o.origin_volume
      else
        "#{o.volume} / #{o.origin_volume}"
      end
    end
    column_localtime :created_at
    column(:state_text)
  end
end
```

97:F:\git\coin\exchange\peatio-master\app\grids\statistic\trades_grid.rb

```
module Statistic
  class TradesGrid
    include Datagrid
    include Datagrid::Naming
    include Datagrid::ColumnI18n

    scope do
      Trade.order('created_at DESC')
    end

    filter(:currency, :enum, :select => Trade.currency.value_options, :default => 3, :include_blank
=> false)
    filter(:created_at, :datetime, :range => true, :default => proc { [1.day.ago, Time.now]})

    column(:id, :order => nil)
    column(:ask_id, :order => nil)
    column(:bid_id, :order => nil)
    column(:price)
    column(:volume)
    column(:strike_amount) { price * volume }
    column_localtime :created_at
  end
end
```

98:F:\git\coin\exchange\peatio-master\app\grids\statistic\withdraws_grid.rb

```
module Statistic
  class WithdrawsGrid
    include Datagrid
    include Datagrid::Naming
    include Datagrid::ColumnI18n

    scope do
      Withdraw.includes(:account).order(id: :desc)
    end
    #filter(:channel, :enum, :select => WithdrawChannel.all, :default => 100, :include_blank =>
false)
    filter(:aasm_state, :enum, :select => Withdraw::STATES, :default => 500)
    filter(:created_at, :datetime, :range => true, :default => proc { [1.day.ago, Time.now]})

    column(:member) do |model|
      format(model) do
        link_to model.account.member.name, member_path(model.member_id)
      end
    end

    column :currency do
      self.account.currency_text
    end

    column(:channel)
    column(:amount)
    column(:address) do
      self.address.mask
    end
    column_localtime :created_at
    column(:aasm_state_text)
  end
end

99:F:\git\coin\exchange\peatio-master\app\helpers\application_helper.rb
module ApplicationHelper
  def document_to(key: nil, title: nil, &block)
    if title
      link_to(title, '', :data => {:remote => "#{main_app.document_path(key)}", :toggle => "modal",
:target => '#document_modal'})
    elsif block
      link_to('', :data => {:remote => "#{main_app.document_path(key)}", :toggle => "modal", :target
=> '#document_modal'}, &block)
    end
  end

  def detail_section_tag(title)
    content_tag('span', title, :class => 'detail-section') + \
    tag('hr')
  end
  def detail_tag(obj, title: 'detail', field: nil, cls: '', clip: nil)
    if field.present?
      field = field.to_s
      val = obj.instance_eval(field)
      display = val || 'N/A'
      content_tag('span', :class => "#{field} detail-item #{val ? nil : 'empty'}" + cls, :data => {:title =>
obj.class.han(field)}) do
        if clip and val
          content_tag('i', display, :class => 'fa fa-copy', :data => {:'clipboard-text' => display})
        else
          content_tag('span', display)
        end
      end
    else
      content_tag('span', obj, :class => 'detail-item ' + cls, :data => {title: title})
    end
  end

  def cs_link
    link_to t('helpers.action.customer_service'), "javascript:void(0);", :onclick =>
"olark('api.box.expand')"
  end

  def check_active(klass)
    if klass.is_a? String
      return 'active' unless (controller.controller_path.exclude?(klass.singularize))
    else
      return 'active' if (klass.model_name.singular == controller.controller_name.singularize)
    end
  end

  def qr_tag(text)
    return if text.blank?
    content_tag :div, '', 'class'       => 'qrcode-container img-thumbnail',
                          'data-width'  => 272,
                          'data-height' => 272,
                          'data-text'   => text
  end

  def rev_category(type)
    type.to_sym == :bid ? :ask : :bid
```

```
end

def orders_json(orders)
  Jbuilder.encode do |json|
    json.array! orders do |order|
      json.id order.id
      json.bid order.bid
      json.ask order.ask
      json.category order.kind
      json.volume order.volume
      json.price order.price
      json.origin_volume order.origin_volume
      json.at order.created_at.to_i
    end
  end
end

def top_nav(link_text, link_path, link_icon, links = nil, controllers: [])
  if links && links.length > 1
    top_dropdown_nav(link_text, link_path, link_icon, links, controllers: controllers)
  else
    top_nav_link(link_text, link_path, link_icon, controllers: controllers)
  end
end

def top_market_link(market, current_market)
  class_name = ((market.id == current_market.id) ? 'active' : nil)

  content_tag(:li, :class => class_name) do
    link_to market_path(market.id)  do
      content_tag(:span, market.name)
    end
  end
end

def top_nav_link(link_text, link_path, link_icon, controllers: [], counter: 0, target: '')
  merged = (controllers & controller_path.split('/'))
  class_name = current_page?(link_path) ? 'active' : nil
  class_name ||= merged.empty? ? nil : 'active'

  content_tag(:li, :class => class_name) do
    link_to link_path, target: target do
```

```
      content_tag(:i, :class => "fa fa-#{link_icon}") do
        content_tag(:span, counter,class: "counter") if counter != 0
      end +
      content_tag(:span, link_text)
    end
  end
end

def top_dropdown_nav(link_text, link_path, link_icon, links, controllers: [])
  class_name = current_page?(link_path) ? 'active' : nil
  class_name ||= (controllers & controller_path.split('/')).empty? ? nil : 'active'

  content_tag(:li, class: "dropdown #{class_name}") do
    link_to(link_path, class: 'dropdown-toggle', 'data-toggle' => 'dropdown') do
      concat content_tag(:i, nil, class: "fa fa-#{link_icon}")
      concat content_tag(:span, link_text)
      concat content_tag(:b, nil, class: 'caret')
    end +
    content_tag(:ul, class: 'dropdown-menu') do
      links.collect do |link|
        concat content_tag(:li, link_to(*link))
      end
    end
  end
end

def history_links
  [ [t('header.order_history'), order_history_path],
    [t('header.trade_history'), trade_history_path],
    [t('header.account_history'), account_history_path] ]
end

def simple_vertical_form_for(record, options={}, &block)
  result = simple_form_for(record, options, &block)
  result = result.gsub(/#{SimpleForm.default_form_class}/, "simple_form").html_safe
  result.gsub(/col-xs-\d/, "").html_safe
end

def panel(name: 'default-panel', key: nil, &block)
  key ||= "guides.#{i18n_controller_path}.#{action_name}.#{name}"

  content_tag(:div, :class => 'panel panel-default') do
```

```
      content_tag(:div, :class => 'panel-heading') do
        content_tag(:h3, :class => 'panel-title') do
          I18n.t(key)
        end
      end +
      content_tag(:div, :class => 'panel-body') do
        capture(&block)
      end
    end
  end

  def locale_name
    I18n.locale.to_s.downcase
  end

  def body_id
    "#{controller_name}-#{action_name}"
  end

  def balance_panel(member: nil)
    member ||= current_user
    panel name: 'balance-pannel', key: 'guides.panels.balance' do
      render partial: 'private/shared/balances', locals: {member: member}
    end
  end

  def guide_panel_title
    @guide_panel_title || t("guides.#{i18n_controller_path}.#{action_name}.panel", default:
t("guides.#{i18n_controller_path}.panel"))
  end

  def guide_title
    @guide_title || t("guides.#{i18n_controller_path}.#{action_name}.title", default:
t("guides.#{i18n_controller_path}.panel"))
  end

  def guide_intro
    @guide_intro || t("guides.#{i18n_controller_path}.#{action_name}.intro", default:
t("guides.#{i18n_controller_path}.intro", default: ''))
  end

  def i18n_controller_path
  @i18n_controller_path ||= controller_path.gsub(/\//, '.')
end

def language_path(lang=nil)
  lang ||= I18n.locale
  asset_path("/languages/#{lang}.png")
end

def i18n_meta(key)
  t("#{i18n_controller_path}.#{action_name}.#{key}", default: :"layouts.meta.#{key}")
end

def description_for(name, &block)
  content_tag :dl, class: "dl-horizontal dl-#{name}" do
    capture(&block)
  end
end

def item_for(model_or_title, name='', value = nil, &block)
  if model_or_title.is_a? String or model_or_title.is_a? Symbol
    title = model_or_title
    capture do
      if block_given?
        content_tag(:dt, title.to_s) +
          content_tag(:dd, capture(&block))
      else
        value = name
        content_tag(:dt, title.to_s) +
          content_tag(:dd, value)
      end
    end
  else
    model = model_or_title
    capture do
      if block_given?
        content_tag(:dt, model.class.human_attribute_name(name)) +
          content_tag(:dd, capture(&block))
      else
        value ||= model.try(name)
        value = value.localtime if value.is_a? DateTime
        value = I18n.t(value) if value.is_a? TrueClass
        content_tag(:dt, model.class.human_attribute_name(name)) +
          content_tag(:dd, value)
      end
    end
  end
end

def yesno(val)
  if val
    content_tag(:span, 'YES', class: 'label label-success')
  else
    content_tag(:span, 'NO', class: 'label label-danger')
  end
end

def format_currency(number, currency, n: nil)
  currency_obj = Currency.find_by_code(currency.to_s)
  digit = n || currency_obj.decimal_digit
  decimal = (number || 0).to_d.round(0, digit)
  decimal = number_with_precision(decimal, precision: digit, delimiter: ',')
  "<span class='decimal'><small>#{currency_obj.symbol}</small>#{decimal}</span>"
end

def partial_phone_number(member)
  number = Phonelib.parse(member.phone_number).national
  mask = number.gsub(/\d/, '*')
  "#{number.first(3)}#{mask[3,number.size-7]}#{number.last(4)}"
end

alias_method :d, :format_currency
end
```

100:F:\git\coin\exchange\peatio-master\app\helpers\mailer_helper.rb

```
module MailerHelper

  def assets_value_change_total(changes)
    total = changes.sum do |(currency, amount, value)|
      currency.code == 'cny' ? 0 : (value[0] || 0)
    end
    pretty_change pretty_currency(total, 'cny'), total
  end
  def trades_change_total(changes)
    total = changes.sum {|(market, change)| change[0] || 0 }
    pretty_change total
  end

  def pretty_currency(amount, currency)
    if amount
      if amount == 0
        '0'
      else
        "%.2f %s" % [amount, currency.upcase]
      end
    else
      '-'
    end
  end

  def pretty_change(change, direction=nil)
    direction ||= change
    if change.nil? || change == '-'
      '-'
    elsif direction > 0
      "#{change} <span style='color:#0F0;'>&#11014;</span>".html_safe
    elsif direction < 0
      "#{change} <span style='color:#F00;'>&#11015;</span>".html_safe
    else
      change
    end
  end

  def pretty_percentage(value)
    if value
      "%.2f%%" % (value*100)
    else
      '-'
    end
  end

end
```

101:F:\git\coin\exchange\peatio-master\app\helpers\private\assets_helper.rb

```
module Private::AssetsHelper
```

```
  def verify_link(proof, partial_tree)
    hashtag = "verify?partial_tree=#{partial_tree.json.to_json}&expected_root=#{proof.root.to_json}"
    uri = "http://syskall.com/proof-of-liabilities/##{URI.encode hashtag}"
    link_to t('.go-verify'), uri, :class => 'btn btn-default', :target => '_blank'
  end

end
```

102:F:\git\coin\exchange\peatio-master\app\helpers\private\history_helper.rb

```
module Private::HistoryHelper

  def trade_side(trade)
    trade.ask_member == current_user ? 'sell' : 'buy'
  end

  def transaction_type(t)
    t(".#{t.class.superclass.name}")
  end

  def transaction_txid_link(t)
    return t.txid unless t.currency_obj.coin?

    txid = t.txid || ''
    link_to txid, t.blockchain_url
  end

end
```

103:F:\git\coin\exchange\peatio-master\app\helpers\private\tickets_helper.rb

```
module Private::TicketsHelper
  def member_tittle(author)
    if current_user == author
      I18n.t('private.tickets.me')
    else
      I18n.t('private.tickets.supporter')
    end
  end

  def close_open_toggle_link
    if params[:closed]
      link_to t('private.tickets.view_open_tickets'), tickets_path
    else
      link_to t('private.tickets.view_closed_tickets'), tickets_path(closed: true)
    end
  end

end

104:F:\git\coin\exchange\peatio-master\app\helpers\tag_helper.rb
module TagHelper
  def member_tag(key)
    raise unless MemberTag.find_by_key(key)
    content_tag('span', I18n.t("tags.#{key}"), :class => "member-tag #{key}")
  end

  def admin_asset_tag(asset)
    return if asset.blank?

    if asset.image?
      link_to image_tag(asset.file.url, style: 'max-width:500px;max-height:500px;'), asset.file.url,
target: '_blank'
    else
      link_to asset['file'], asset.file.url
    end
  end

  def bank_code_to_name(code)
    I18n.t("banks.#{code}")
  end
end

105:F:\git\coin\exchange\peatio-master\app\helpers\two_factor_helper.rb
module TwoFactorHelper

  def two_factor_tag(user)
    locals = {
      app_activated: user.app_two_factor.activated?,
      sms_activated: user.sms_two_factor.activated?
    }
    render partial: 'shared/two_factor_auth', locals: locals
  end

  def unlock_two_factor!
```

```
    session[:two_factor_unlock] = true
    session[:two_factor_unlock_at] = Time.now
  end

  def two_factor_locked?(expired_at: 5.minutes)
    locked  = !session[:two_factor_unlock]
    expired = session[:two_factor_unlock_at].nil? ? true : session[:two_factor_unlock_at] <
expired_at.ago

    if !locked and !expired
      session[:two_factor_unlock_at] = Time.now
    end

    locked or expired
  end

end
```

106:F:\git\coin\exchange\peatio-master\app\inputs\display_input.rb

```
class DisplayInput < SimpleForm::Inputs::Base
  def input
    clip = input_options.delete(:clip)
    value = input_html_options[:value] || object.send(attribute_name)
    template.content_tag(:p, value, class: 'form-control-static') do
      template.concat template.content_tag(:span, value)
      if clip && value
        template.concat template.content_tag('i', '', class: 'fa fa-copy', data: {'clipboard-text' => value})
      end
    end
  end

  def additional_classes
    @additional_classes ||= [input_type].compact # original is `[input_type, required_class,
readonly_class, disabled_class].compact`
  end
end
```

107:F:\git\coin\exchange\peatio-master\app\mailers\base_mailer.rb

```
class BaseMailer < ActionMailer::Base
  include AMQPQueue::Mailer

  layout 'mailers/application'
  add_template_helper MailerHelper

  default from: ENV['SYSTEM_MAIL_FROM'],
          reply_to: ENV['SUPPORT_MAIL']
end

108:F:\git\coin\exchange\peatio-master\app\mailers\comment_mailer.rb
class CommentMailer < BaseMailer

  def user_notification(comment_id)
    comment = Comment.find comment_id
    @ticket_url = ticket_url(comment.ticket)

    mail to: comment.ticket.author.email
  end

  def admin_notification(comment_id)
    comment = Comment.find comment_id
    @ticket_url = admin_ticket_url(comment.ticket)
    @author_email = comment.author.email

    mail to: ENV['SUPPORT_MAIL']
  end

end

109:F:\git\coin\exchange\peatio-master\app\mailers\deposit_mailer.rb
class DepositMailer < BaseMailer

  def accepted(deposit_id)
    @deposit = Deposit.find deposit_id
    mail to: @deposit.member.email
  end

end

110:F:\git\coin\exchange\peatio-master\app\mailers\member_mailer.rb
class MemberMailer < BaseMailer

  def notify_signin(member_id)
    set_mail(member_id)
  end
  def google_auth_activated(member_id)
    set_mail(member_id)
  end

  def google_auth_deactivated(member_id)
    set_mail(member_id)
  end

  def sms_auth_activated(member_id)
    set_mail(member_id)
  end

  def sms_auth_deactivated(member_id)
    set_mail(member_id)
  end

  def reset_password_done(member_id)
    set_mail(member_id)
  end

  def phone_number_verified(member_id)
    set_mail(member_id)
  end

  private

  def set_mail(member_id)
    @member = Member.find member_id
    mail to: @member.email
  end
end
```

111:F:\git\coin\exchange\peatio-master\app\mailers\system_mailer.rb

```
class SystemMailer < BaseMailer

  default from: ENV["SYSTEM_MAIL_FROM"],
          to:   ENV["SYSTEM_MAIL_TO"]

  layout 'mailers/system'

  def balance_warning(amount, balance)
  @amount = amount
  @balance = balance
  mail :subject => "satoshi balance warning"
end

def trade_execute_error(payload, error, backtrace)
  @payload   = payload
  @error     = error
  @backtrace = backtrace
  mail subject: "Trade execute error: #{@error}"
end

def order_processor_error(payload, error, backtrace)
  @payload   = payload
  @error     = error
  @backtrace = backtrace
  mail subject: "Order processor error: #{@error}"
end

def daily_stats(ts, stats, base)
  @stats = stats
  @base  = base

  @changes = {
    assets: Currency.all.map {|c|
      [ c,
        compare(@base['asset_stats'][c.code][1], @stats['asset_stats'][c.code][1]),
        compare(@base['asset_stats'][c.code][0], @stats['asset_stats'][c.code][0])
      ]
    },
    trades: Market.all.map {|m|
      [ m,
        compare(@base['trade_users'][m.id][1], @stats['trade_users'][m.id][1])
      ]
    }
  }

  from   = Time.at(ts)
  to     = Time.at(ts + 1.day - 1)
  mail subject: "Daily Summary (#{from} - #{to})",
       to: ENV['OPERATE_MAIL_TO']
end
```

```
  private

  def compare(before, now)
    if before.nil? || now.nil?
      []
    else
      [ now-before, percentage_compare(before, now) ]
    end
  end

  def percentage_compare(before, now)
    if before == 0
      nil
    else
      (now-before) / before.to_f
    end
  end

end
```

112:F:\git\coin\exchange\peatio-master\app\mailers\ticket_mailer.rb

```
class TicketMailer < BaseMailer

  def author_notification(ticket_id)
    ticket = Ticket.find ticket_id
    @ticket_url = ticket_url(ticket)

    mail to: ticket.author.email
  end

  def admin_notification(ticket_id)
    ticket = Ticket.find ticket_id
    @author_email = ticket.author.email
    @ticket_url = admin_ticket_url(ticket)

    mail to: ENV['SUPPORT_MAIL']
  end

end
```

113:F:\git\coin\exchange\peatio-master\app\mailers\token_mailer.rb

```
class TokenMailer < BaseMailer

  def reset_password(email, token)
    @token_url = edit_reset_password_url(token)
    mail to: email
  end

  def activation(email, token)
    @token_url = edit_activation_url token
    mail to: email
  end

end
```

114:F:\git\coin\exchange\peatio-master\app\mailers\withdraw_mailer.rb

```
class WithdrawMailer < BaseMailer

  def submitted(withdraw_id)
    set_mail(withdraw_id)
  end

  def processing(withdraw_id)
    set_mail(withdraw_id)
  end

  def done(withdraw_id)
    set_mail(withdraw_id)
  end

  def withdraw_state(withdraw_id)
    set_mail(withdraw_id)
  end

  private

  def set_mail(withdraw_id)
    @withdraw = Withdraw.find withdraw_id
    mail to: @withdraw.member.email
  end

end
```

115:F:\git\coin\exchange\peatio-master\app\models\account.rb

```
class Account < ActiveRecord::Base
  include Currencible

  FIX = :fix
  UNKNOWN = :unknown
  STRIKE_ADD = :strike_add
  STRIKE_SUB = :strike_sub
  STRIKE_FEE = :strike_fee
  STRIKE_UNLOCK = :strike_unlock
  ORDER_CANCEL = :order_cancel
  ORDER_SUBMIT = :order_submit
  ORDER_FULLFILLED = :order_fullfilled
  WITHDRAW_LOCK = :withdraw_lock
  WITHDRAW_UNLOCK = :withdraw_unlock
  DEPOSIT = :deposit
  WITHDRAW = :withdraw
  ZERO = 0.to_d

  FUNS = {:unlock_funds => 1, :lock_funds => 2, :plus_funds => 3, :sub_funds => 4,
:unlock_and_sub_funds => 5}

  belongs_to :member
  has_many :payment_addresses
  has_many :versions, class_name: "::AccountVersion"
  has_many :partial_trees

  # Suppose to use has_one here, but I want to store
  # relationship at account side. (Daniel)
  belongs_to :default_withdraw_fund_source, class_name: 'FundSource'

  validates :member_id, uniqueness: { scope: :currency }
  validates_numericality_of :balance, :locked, greater_than_or_equal_to: ZERO

  scope :enabled, -> { where("currency in (?)", Currency.ids) }

  after_commit :trigger, :sync_update

  def payment_address
    payment_addresses.last || payment_addresses.create(currency: self.currency)
  end
  def self.after(*names)
    names.each do |name|
      m = instance_method(name.to_s)
      define_method(name.to_s) do |*args, &block|
        m.bind(self).(*args, &block)
        yield(self, name.to_sym, *args)
        self
      end
    end
  end

  def plus_funds(amount, fee: ZERO, reason: nil, ref: nil)
    (amount <= ZERO or fee > amount) and raise AccountError, "cannot add funds (amount:
#{amount})"
    change_balance_and_locked amount, 0
  end

  def sub_funds(amount, fee: ZERO, reason: nil, ref: nil)
    (amount <= ZERO or amount > self.balance) and raise AccountError, "cannot subtract funds
(amount: #{amount})"
    change_balance_and_locked -amount, 0
  end

  def lock_funds(amount, reason: nil, ref: nil)
    (amount <= ZERO or amount > self.balance) and raise AccountError, "cannot lock funds
(amount: #{amount})"
    change_balance_and_locked -amount, amount
  end

  def unlock_funds(amount, reason: nil, ref: nil)
    (amount <= ZERO or amount > self.locked) and raise AccountError, "cannot unlock funds
(amount: #{amount})"
    change_balance_and_locked amount, -amount
  end

  def unlock_and_sub_funds(amount, locked: ZERO, fee: ZERO, reason: nil, ref: nil)
    raise AccountError, "cannot unlock and subtract funds (amount: #{amount})" if ((amount <= 0) or
(amount > locked))
    raise LockedError, "invalid lock amount" unless locked
    raise LockedError, "invalid lock amount (amount: #{amount}, locked: #{locked}, self.locked:
#{self.locked})" if ((locked <= 0) or (locked > self.locked))
    change_balance_and_locked locked-amount, -locked
  end

  after(*FUNS.keys) do |account, fun, changed, opts|
    begin
      opts ||= {}
      fee = opts[:fee] || ZERO
      reason = opts[:reason] || Account::UNKNOWN

      attributes = { fun: fun,
                     fee: fee,
                     reason: reason,
                     amount: account.amount,
                     currency: account.currency.to_sym,
                     member_id: account.member_id,
                     account_id: account.id }

      if opts[:ref] and opts[:ref].respond_to?(:id)
        ref_klass = opts[:ref].class
        attributes.merge! \
          modifiable_id: opts[:ref].id,
          modifiable_type: ref_klass.respond_to?(:base_class) ? ref_klass.base_class.name :
ref_klass.name
      end

      locked, balance = compute_locked_and_balance(fun, changed, opts)
      attributes.merge! locked: locked, balance: balance

      AccountVersion.optimistically_lock_account_and_create!(account.balance, account.locked,
attributes)
    rescue ActiveRecord::StaleObjectError
      Rails.logger.info "Stale account##{account.id} found when create associated account version,
retry."
      account = Account.find(account.id)
      raise ActiveRecord::RecordInvalid, account unless account.valid?
      retry
    end
  end

  def self.compute_locked_and_balance(fun, amount, opts)
    raise AccountError, "invalid account operation" unless FUNS.keys.include?(fun)

    case fun
    when :sub_funds then [ZERO, ZERO - amount]
    when :plus_funds then [ZERO, amount]
    when :lock_funds then [amount, ZERO - amount]
    when :unlock_funds then [ZERO - amount, amount]
    when :unlock_and_sub_funds
      locked = ZERO - opts[:locked]
      balance = opts[:locked] - amount
      [locked, balance]
    else raise AccountError, "forbidden account operation"
    end
  end

  def amount
    self.balance + self.locked
  end

  def last_version
    versions.last
  end

  def examine
    expected = 0
    versions.find_each(batch_size: 100000) do |v|
      expected += v.amount_change
      return false if expected != v.amount
    end

    expected == self.amount
  end

  def trigger
    return unless member

    json = Jbuilder.encode do |json|
      json.(self, :balance, :locked, :currency)
    end
    member.trigger('account', json)
  end

  def change_balance_and_locked(delta_b, delta_l)
    self.balance += delta_b
    self.locked  += delta_l
```

```
    self.class.connection.execute "update accounts set balance = balance + #{delta_b}, locked =
locked + #{delta_l} where id = #{id}"
    add_to_transaction # so after_commit will be triggered
    self
  end

  scope :locked_sum, -> (currency) { with_currency(currency).sum(:locked) }
  scope :balance_sum, -> (currency) { with_currency(currency).sum(:balance) }

  class AccountError < RuntimeError; end
  class LockedError < AccountError; end
  class BalanceError < AccountError; end

  def as_json(options = {})
    super(options).merge({
      # check if there is a useable address, but don't touch it to create the address now.
      "deposit_address" => payment_addresses.empty? ? "" : payment_address.deposit_address,
      "name_text" => currency_obj.name_text,
      "default_withdraw_fund_source_id" => default_withdraw_fund_source_id
    })
  end

  private

  def sync_update
    ::Pusher["private-#{member.sn}"].trigger_async('accounts', { type: 'update', id: self.id, attributes:
{balance: balance, locked: locked} })
  end

end

116:F:\git\coin\exchange\peatio-master\app\models\account_version.rb
class AccountVersion < ActiveRecord::Base
  include Currencible

  HISTORY = [Account::STRIKE_ADD, Account::STRIKE_SUB, Account::STRIKE_FEE,
Account::DEPOSIT, Account::WITHDRAW, Account::FIX]

  enumerize :fun, in: Account::FUNS

  REASON_CODES = {
    Account::UNKNOWN => 0,
  Account::FIX => 1,
  Account::STRIKE_FEE => 100,
  Account::STRIKE_ADD => 110,
  Account::STRIKE_SUB => 120,
  Account::STRIKE_UNLOCK => 130,
  Account::ORDER_SUBMIT => 600,
  Account::ORDER_CANCEL => 610,
  Account::ORDER_FULLFILLED => 620,
  Account::WITHDRAW_LOCK => 800,
  Account::WITHDRAW_UNLOCK => 810,
  Account::DEPOSIT => 1000,
  Account::WITHDRAW => 2000 }
enumerize :reason, in: REASON_CODES, scope: true

belongs_to :account
belongs_to :modifiable, polymorphic: true

scope :history, -> { with_reason(*HISTORY).reverse_order }

# Use account balance and locked columes as optimistic lock column. If the
# passed in balance and locked doesn't match associated account's data in
# database, exception raise. Otherwise the AccountVersion record will be
# created.
#
# TODO: find a more generic way to construct the sql
def self.optimistically_lock_account_and_create!(balance, locked, attrs)
  attrs = attrs.symbolize_keys

  attrs[:created_at] = Time.now
  attrs[:updated_at] = attrs[:created_at]
  attrs[:fun]        = Account::FUNS[attrs[:fun]]
  attrs[:reason]     = REASON_CODES[attrs[:reason]]
  attrs[:currency]   = Currency.enumerize[attrs[:currency]]

  account_id = attrs[:account_id]
  raise ActiveRecord::ActiveRecordError, "account must be specified" unless account_id.present?

  qmarks       = (['?']*attrs.size).join(',')
  values_array = [qmarks, *attrs.values]
  values       = ActiveRecord::Base.send :sanitize_sql_array, values_array

  select = Account.unscoped.select(values).where(id: account_id, balance: balance, locked:
```

```
locked).to_sql
    stmt   = "INSERT INTO account_versions (#{attrs.keys.join(',')}) #{select}"

    connection.insert(stmt).tap do |id|
      if id == 0
        record = new attrs
        raise ActiveRecord::StaleObjectError.new(record, "create")
      end
    end
  end

  def detail_template
    if self.detail.nil? || self.detail.empty?
      return ["system", {}]
    end

    [self.detail.delete(:tmp) || "default", self.detail || {}]
  end

  def amount_change
    balance + locked
  end

  def in
    amount_change > 0 ? amount_change : nil
  end

   def out
    amount_change < 0 ? amount_change : nil
  end

  alias :template :detail_template
end

117:F:\git\coin\exchange\peatio-master\app\models\active_yaml_base.rb
class ActiveYamlBase < ActiveYaml::Base
  field :sort_order, default: 9999

  if Rails.env == 'test'
    set_root_path "#{Rails.root}/spec/fixtures"
  else
    set_root_path "#{Rails.root}/config"
  end

  private

  def <=>(other)
    self.sort_order <=> other.sort_order
  end
end

118:F:\git\coin\exchange\peatio-master\app\models\admin\ability.rb
module Admin
  class Ability
    include CanCan::Ability

    def initialize(user)
      return unless user.admin?

      can :read, Order
      can :read, Trade
      can :read, Proof
      can :update, Proof
      can :manage, Document
      can :manage, Member
      can :manage, Ticket
      can :manage, IdDocument
      can :manage, TwoFactor

      can :menu, Deposit
      can :manage, ::Deposits::Bank
      can :manage, ::Deposits::Satoshi

      can :menu, Withdraw
      can :manage, ::Withdraws::Bank
      can :manage, ::Withdraws::Satoshi
    end
  end
end

119:F:\git\coin\exchange\peatio-master\app\models\amqp_config.rb
class AMQPConfig
  class <<self
    def data
      @data ||= Hashie::Mash.new YAML.load_file(Rails.root.join('config', 'amqp.yml'))
    end

    def connect
      data[:connect]
    end

    def binding_exchange_id(id)
      data[:binding][id][:exchange]
    end

    def binding_exchange(id)
      eid = binding_exchange_id(id)
      eid && exchange(eid)
    end

    def binding_queue(id)
      queue data[:binding][id][:queue]
    end

    def binding_worker(id)
      ::Worker.const_get(id.to_s.camelize).new
    end

    def routing_key(id)
      binding_queue(id).first
    end

    def topics(id)
      data[:binding][id][:topics].split(',')
    end

    def channel(id)
      (data[:channel] && data[:channel][id]) || {}
    end

    def queue(id)
      name = data[:queue][id][:name]
      settings = { durable: data[:queue][id][:durable] }
      [name, settings]
    end
```

```
    def exchange(id)
      type = data[:exchange][id][:type]
      name = data[:exchange][id][:name]
      [type, name]
    end

  end
end

120:F:\git\coin\exchange\peatio-master\app\models\amqp_queue.rb
class AMQPQueue

  class <<self
    def connection
      @connection ||= Bunny.new(AMQPConfig.connect).tap do |conn|
        conn.start
      end
    end

    def channel
      @channel ||= connection.create_channel
    end

    def exchanges
      @exchanges ||= {default: channel.default_exchange}
    end

    def exchange(id)
      exchanges[id] ||= channel.send *AMQPConfig.exchange(id)
    end

    def publish(eid, payload, attrs={})
      payload = JSON.dump payload
      exchange(eid).publish(payload, attrs)
    end

    # enqueue = publish to direct exchange
    def enqueue(id, payload, attrs={})
      eid = AMQPConfig.binding_exchange_id(id) || :default
      payload.merge!({locale: I18n.locale})
      attrs.merge!({routing_key: AMQPConfig.routing_key(id)})
      publish(eid, payload, attrs)
    end
  end

  module Mailer
    class <<self
      def included(base)
        base.extend(ClassMethods)
      end

      def excluded_environment?(name)
        [:test].include?(name.try(:to_sym))
      end
    end

    module ClassMethods

      def method_missing(method_name, *args)
        if action_methods.include?(method_name.to_s)
          MessageDecoy.new(self, method_name, *args)
        else
          super
        end
      end

      def deliver?
        true
      end
    end

    class MessageDecoy
      delegate :to_s, :to => :actual_message

      def initialize(mailer_class, method_name, *args)
        @mailer_class = mailer_class
        @method_name = method_name
        *@args = *args
        actual_message if environment_excluded?
      end

      def environment_excluded?
        !ActionMailer::Base.perform_deliveries ||
::AMQPQueue::Mailer.excluded_environment?(Rails.env)
```

```
      end

      def actual_message
        @actual_message ||= @mailer_class.send(:new, @method_name, *@args).message
      end

      def deliver
        return deliver! if environment_excluded?

        if @mailer_class.deliver?
          begin
            AMQPQueue.enqueue(:email_notification, mailer_class: @mailer_class.to_s, method:
@method_name, args: @args)
          rescue
            Rails.logger.error "Unable to enqueue :mailer: #{$!}, fallback to synchronous mail delivery"
            deliver!
          end
        end
      end

      def deliver!
        actual_message.deliver
      end

      def method_missing(method_name, *args)
        actual_message.send(method_name, *args)
      end
    end
  end

end

121:F:\git\coin\exchange\peatio-master\app\models\api_token.rb
class APIToken < ActiveRecord::Base
  paranoid

  belongs_to :member
  belongs_to :oauth_access_token, class_name: 'Doorkeeper::AccessToken', dependent: :destroy

  serialize :trusted_ip_list

  validates_presence_of :access_key, :secret_key
```

```
before_validation :generate_keys, on: :create

scope :user_requested,  -> { where('oauth_access_token_id IS NULL') }
scope :oauth_requested, -> { where('oauth_access_token_id IS NOT NULL') }

def self.from_oauth_token(token)
  return nil unless token && token.token.present?
  access_key, secret_key = token.token.split(':')
  find_by_access_key access_key
end

def to_oauth_token
  [access_key, secret_key].join(':')
end

def expired?
  expire_at && expire_at < Time.now
end

def in_scopes?(ary)
  return true if ary.blank?
  return true if self[:scopes] == 'all'
  (ary & scopes).present?
end

def allow_ip?(ip)
  trusted_ip_list.blank? || trusted_ip_list.include?(ip)
end

def ip_whitelist=(list)
  self.trusted_ip_list = list.split(/,\s*/)
end

def ip_whitelist
  trusted_ip_list.try(:join, ',')
end

def scopes
  self[:scopes] ? self[:scopes].split(/\s+/) : []
end
```

```
  private

  def generate_keys
    begin
      self.access_key = APIv2::Auth::Utils.generate_access_key
    end while APIToken.where(access_key: access_key).any?

    begin
      self.secret_key = APIv2::Auth::Utils.generate_secret_key
    end while APIToken.where(secret_key: secret_key).any?
  end

end
```

122:F:\git\coin\exchange\peatio-master\app\models\asset.rb

```
class Asset < ActiveRecord::Base
  belongs_to :attachable, polymorphic: true

  mount_uploader :file, FileUploader

  def image?
    file.content_type.start_with?('image') if file?
  end
end

class Asset::IdDocumentFile < Asset
end

class Asset::IdBillFile < Asset
end
```

123:F:\git\coin\exchange\peatio-master\app\models\audit\audit_log.rb

```
module Audit
  class AuditLog < ActiveRecord::Base
    belongs_to :operator, class_name: 'Member', foreign_key: 'operator_id'
    belongs_to :auditable, polymorphic: true
  end
end
```

124:F:\git\coin\exchange\peatio-master\app\models\audit\transfer_audit_log.rb

```
module Audit
  class TransferAuditLog < AuditLog
```

```
    def self.audit!(transfer, operator = nil)
      create(operator_id: operator.try(:id), auditable: transfer,
             source_state: transfer.aasm_state_was, target_state: transfer.aasm_state)
    end

  end
end

125:F:\git\coin\exchange\peatio-master\app\models\authentication.rb
class Authentication < ActiveRecord::Base
  belongs_to :member

  validates :provider, presence: true, uniqueness: { scope: :member_id }
  validates :uid,      presence: true, uniqueness: { scope: :provider }

  class << self
    def locate(auth)
      uid      = auth['uid'].to_s
      provider = auth['provider']
      find_by_provider_and_uid provider, uid
    end

    def build_auth(auth)
      new \
        uid:      auth['uid'],
        provider: auth['provider'],
        token:    auth['credentials'].try(:[], 'token'),
        secret:   auth['credentials'].try(:[], 'secret'),
        nickname: auth['info'].try(:[], 'nickname')
    end
  end
end

126:F:\git\coin\exchange\peatio-master\app\models\bank.rb
class Bank < ActiveYamlBase
  include HashCurrencible

  def self.with_currency(c)
    find_all_by_currency c.to_s
  end
end
```

```
127:F:\git\coin\exchange\peatio-master\app\models\comment.rb
class Comment < ActiveRecord::Base
  after_commit :send_notification, on: [:create]

  acts_as_readable on: :created_at
  belongs_to :ticket
  belongs_to :author, class_name: 'Member', foreign_key: 'author_id'

  validates :content, presence: true

  private

  def send_notification
    ticket_author = self.ticket.author

    if ticket_author != self.author
      CommentMailer.user_notification(self.id).deliver
    else
      CommentMailer.admin_notification(self.id).deliver
    end
  end
end

128:F:\git\coin\exchange\peatio-master\app\models\concerns\aasm_absolutely.rb
module AasmAbsolutely
  extend ActiveSupport::Concern

  included do
    enumerize :aasm_state, in: self.superclass::STATES, scope: true, i18n_scope:
"#{name.underscore}.aasm_state"
  end
end

129:F:\git\coin\exchange\peatio-master\app\models\concerns\channelable.rb
module Channelable
  extend ActiveSupport::Concern

  included do
    def self.category
      to_s.underscore.split('_').first.pluralize
    end
  end

  def kls
    "#{self.class.category}/#{key}".camelize.constantize
  end

end
```

130:F:\git\coin\exchange\peatio-master\app\models\concerns\currencible.rb

```
module Currencible
  extend ActiveSupport::Concern

  included do
    extend Enumerize
    enumerize :currency, in: Currency.enumerize, scope: true
    belongs_to_active_hash :currency_obj, class_name: 'Currency', foreign_key: 'currency_value'
    delegate :key_text, to: :currency_obj, prefix: true
  end
end
```

131:F:\git\coin\exchange\peatio-master\app\models\concerns\deposits\bankable.rb

```
module Deposits
  module Bankable
    extend ActiveSupport::Concern

    included do
      validates :fund_extra, :fund_uid, :amount, presence: true
      delegate :accounts, to: :channel
    end
  end
end
```

132:F:\git\coin\exchange\peatio-master\app\models\concerns\deposits\coinable.rb

```
module Deposits
  module Coinable
    extend ActiveSupport::Concern

    included do
      validates_presence_of :payment_transaction_id
      validates_uniqueness_of :payment_transaction_id
      belongs_to :payment_transaction
    end
    def channel
      @channel ||= DepositChannel.find_by_key(self.class.name.demodulize.underscore)
    end

    def min_confirm?(confirmations)
      update_confirmations(confirmations)
      confirmations >= channel.min_confirm && confirmations < channel.max_confirm
    end

    def max_confirm?(confirmations)
      update_confirmations(confirmations)
      confirmations >= channel.max_confirm
    end

    def update_confirmations(confirmations)
      if !self.new_record? && self.confirmations.to_s != confirmations.to_s
        self.update_attribute(:confirmations, confirmations.to_s)
      end
    end

    def blockchain_url
      currency_obj.blockchain_url(txid)
    end

    def as_json(options = {})
      super(options).merge({
        txid: txid.blank? ? "" : txid[0..29],
        confirmations: payment_transaction.nil? ? 0 : payment_transaction.confirmations,
        blockchain_url: blockchain_url
      })
    end
  end
end

133:F:\git\coin\exchange\peatio-master\app\models\concerns\fund_sourceable.rb
module FundSourceable
  extend ActiveSupport::Concern

  included do
    attr_accessor :fund_source
    before_validation :set_fund_source_attributes, on: :create
    validates :fund_source, presence: true, on: :create
  end

  def set_fund_source_attributes
    if fs = FundSource.find_by(id: fund_source)
      self.fund_extra = fs.extra
      self.fund_uid = fs.uid.strip
    end
  end
end

134:F:\git\coin\exchange\peatio-master\app\models\concerns\hash_currencible.rb
module HashCurrencible
  extend ActiveSupport::Concern

  included do
    def currency_obj
      Currency.find_by_code(attributes[:currency])
    end
  end
end

135:F:\git\coin\exchange\peatio-master\app\models\concerns\international.rb
module International
  extend ActiveSupport::Concern

  included do
    def method_missing(name, *args)
      if name =~ /(.*)_text$/
        attr = $1
        I18n.t(i18n_text_key(attr), attr)
      else
        super(name, *args)
      end
    end

    def i18n_text_key(key)
      "#{self.class.model_name.i18n_key}.#{self.key}.#{key}"
    end
  end
end
```

136:F:\git\coin\exchange\peatio-master\app\models\concerns\withdraws\bankable.rb
```
module Withdraws
  module Bankable
    extend ActiveSupport::Concern

    included do
      validates_presence_of :fund_extra

      delegate :name, to: :member, prefix: true

      alias_attribute :remark, :id
    end

  end
end
```

137:F:\git\coin\exchange\peatio-master\app\models\concerns\withdraws\coinable.rb
```
module Withdraws
  module Coinable
    extend ActiveSupport::Concern

    def set_fee
      self.fee = "0.0001".to_d
    end

    def blockchain_url
      currency_obj.blockchain_url(txid)
    end

    def audit!
      result = CoinRPC[currency].validateaddress(fund_uid)

      if result.nil? || (result[:isvalid] == false)
        Rails.logger.info "#{self.class.name}##{id} uses invalid address: #{fund_uid.inspect}"
        reject
        save!
      elsif (result[:ismine] == true) || PaymentAddress.find_by_address(fund_uid)
        Rails.logger.info "#{self.class.name}##{id} uses hot wallet address: #{fund_uid.inspect}"
        reject
        save!
      else
        super
      end
    end

    def as_json(options={})
      super(options).merge({
        blockchain_url: blockchain_url
      })
    end

  end
end
```

138:F:\git\coin\exchange\peatio-master\app\models\currency.rb

```
class Currency < ActiveYamlBase
  include International
  include ActiveHash::Associations

  field :visible, default: true

  self.singleton_class.send :alias_method, :all_with_invisible, :all
  def self.all
    all_with_invisible.select &:visible
  end

  def self.enumerize
    all_with_invisible.inject({}) {|memo, i| memo[i.code.to_sym] = i.id; memo}
  end

  def self.codes
    @keys ||= all.map &:code
  end

  def self.ids
    @ids ||= all.map &:id
  end

  def self.assets(code)
    find_by_code(code)[:assets]
  end

  def precision
    self[:precision]
  end

  def api
    raise unless coin?
    CoinRPC[code]
  end

  def fiat?
    not coin?
  end

  def balance_cache_key
    "peatio:hotwallet:#{code}:balance"
  end

  def balance
    Rails.cache.read(balance_cache_key) || 0
  end

  def decimal_digit
    self.try(:default_decimal_digit) || (fiat? ? 2 : 4)
  end

  def refresh_balance
    Rails.cache.write(balance_cache_key, api.safe_getbalance) if coin?
  end

  def blockchain_url(txid)
    raise unless coin?
    blockchain.gsub('#{txid}', txid.to_s)
  end

  def address_url(address)
    raise unless coin?
    self[:address_url].try :gsub, '#{address}', address
  end

  def quick_withdraw_max
    @quick_withdraw_max ||= BigDecimal.new self[:quick_withdraw_max].to_s
  end
```

```
  def as_json(options = {})
    {
      key: key,
      code: code,
      coin: coin,
      blockchain: blockchain
    }
  end

  def summary
    locked = Account.locked_sum(code)
    balance = Account.balance_sum(code)
    sum = locked + balance

    coinable = self.coin?
    hot = coinable ? self.balance : nil

    {
      name: self.code.upcase,
      sum: sum,
      balance: balance,
      locked: locked,
      coinable: coinable,
      hot: hot
    }
  end
end
```

139:F:\git\coin\exchange\peatio-master\app\models\deposit.rb

```
class Deposit < ActiveRecord::Base
  STATES = [:submitting, :cancelled, :submitted, :rejected, :accepted, :checked, :warning]

  extend Enumerize

  include AASM
  include AASM::Locking
  include Currencible

  has_paper_trail on: [:update, :destroy]

  enumerize :aasm_state, in: STATES, scope: true
```

```
alias_attribute :sn, :id

delegate :name, to: :member, prefix: true
delegate :id, to: :channel, prefix: true
delegate :coin?, :fiat?, to: :currency_obj

belongs_to :member
belongs_to :account

validates_presence_of \
  :amount, :account, \
  :member, :currency
validates_numericality_of :amount, greater_than: 0

scope :recent, -> { order('id DESC')}

after_update :sync_update
after_create :sync_create
after_destroy :sync_destroy

aasm :whiny_transitions => false do
  state :submitting, initial: true, before_enter: :set_fee
  state :cancelled
  state :submitted
  state :rejected
  state :accepted, after_commit: [:do, :send_mail, :send_sms]
  state :checked
  state :warning

  event :submit do
    transitions from: :submitting, to: :submitted
  end

  event :cancel do
    transitions from: :submitting, to: :cancelled
  end

  event :reject do
    transitions from: :submitted, to: :rejected
  end

  event :accept do
```

```
    transitions from: :submitted, to: :accepted
  end

  event :check do
    transitions from: :accepted, to: :checked
  end

  event :warn do
    transitions from: :accepted, to: :warning
  end
end

def txid_desc
  txid
end

class << self
  def channel
    DepositChannel.find_by_key(name.demodulize.underscore)
  end

  def resource_name
    name.demodulize.underscore.pluralize
  end

  def params_name
    name.underscore.gsub('/', '_')
  end

  def new_path
    "new_#{params_name}_path"
  end
end

def channel
  self.class.channel
end

def update_confirmations(data)
  update_column(:confirmations, data)
end
```

```
  def txid_text
    txid && txid.truncate(40)
  end

  private
  def do
    account.lock!.plus_funds amount, reason: Account::DEPOSIT, ref: self
  end

  def send_mail
    DepositMailer.accepted(self.id).deliver if self.accepted?
  end

  def send_sms
    return true if not member.sms_two_factor.activated?

    sms_message = I18n.t('sms.deposit_done', email: member.email,
                                              currency: currency_text,
                                              time: I18n.l(Time.now),
                                              amount: amount,
                                              balance: account.balance)

    AMQPQueue.enqueue(:sms_notification, phone: member.phone_number, message:
sms_message)
  end

  def set_fee
    amount, fee = calc_fee
    self.amount = amount
    self.fee = fee
  end

  def calc_fee
    [amount, 0]
  end

  def sync_update
    ::Pusher["private-#{member.sn}"].trigger_async('deposits', { type: 'update', id: self.id, attributes:
self.changes_attributes_as_json })
  end

  def sync_create
```

```
    ::Pusher["private-#{member.sn}"].trigger_async('deposits', { type: 'create', attributes:
self.as_json })
  end

  def sync_destroy
    ::Pusher["private-#{member.sn}"].trigger_async('deposits', { type: 'destroy', id: self.id })
  end
end

140:F:\git\coin\exchange\peatio-master\app\models\deposits\bank.rb
module Deposits
  class Bank < ::Deposit
    include ::AasmAbsolutely
    include ::Deposits::Bankable
    include ::FundSourceable

    def charge!(txid)
      with_lock do
        submit!
        accept!
        touch(:done_at)
        update_attribute(:txid, txid)
      end
    end

  end
end

141:F:\git\coin\exchange\peatio-master\app\models\deposits\satoshi.rb
module Deposits
  class Satoshi < ::Deposit
    include ::AasmAbsolutely
    include ::Deposits::Coinable

    validates_uniqueness_of :txout, scope: :txid
  end
end

142:F:\git\coin\exchange\peatio-master\app\models\deposit_channel.rb
class DepositChannel < ActiveYamlBase
  include Channelable
  include HashCurrencible
  include International

  def accounts
    bank_accounts.map {|i| OpenStruct.new(i) }
  end

  def as_json(options = {})
    super(options)['attributes'].merge({resource_name: key.pluralize})
  end
end

143:F:\git\coin\exchange\peatio-master\app\models\document.rb
class Document < ActiveRecord::Base
  TRANSLATABLE_ATTR = [:title, :desc, :keywords, :body]
  translates *TRANSLATABLE_ATTR

  def to_param
    self.key
  end

  TRANSLATABLE_ATTR.each do |attr|
    Rails.configuration.i18n.available_locales.each do |locale|
      locale = locale.to_s
      define_method "#{locale.underscore}_#{attr}=" do |value|
        with_locale locale do
          self.send("#{attr}=", value)
        end
      end

      define_method "#{locale.underscore}_#{attr}" do
        with_locale locale do
          self.send("#{attr}")
        end
      end
    end
  end

  def self.locale_params
    params = []
    TRANSLATABLE_ATTR.each do |attr|
      Rails.configuration.i18n.available_locales.each do |locale|
        locale = locale.to_s
        params << "#{locale.underscore}_#{attr}".to_sym
      end
    end
    params
  end

  private

  def with_locale locale
    original_locale = I18n.locale
    I18n.locale = locale
    value = yield if block_given?
    I18n.locale = original_locale
    value
  end
end
```

144:F:\git\coin\exchange\peatio-master\app\models\fund_source.rb

```
class FundSource < ActiveRecord::Base
  include Currencible

  attr_accessor :name

  paranoid

  belongs_to :member

  validates_presence_of :uid, :extra, :member

  def label
    if currency_obj.try :coin?
      "#{uid} (#{extra})"
    else
      [I18n.t("banks.#{extra}"), "****#{uid[-4..-1]}"].join('#')
    end
  end

  def as_json(options = {})
    super(options).merge({label: label})
  end
end
```

145:F:\git\coin\exchange\peatio-master\app\models\global.rb

```
class Global
  ZERO = '0.0'.to_d
  NOTHING_ARRAY = YAML::dump([])
  LIMIT = 80

  class << self
    def channel
      "market-global"
    end

    def trigger(event, data)
      Pusher.trigger_async(channel, event, data)
    end

    def daemon_statuses
      Rails.cache.fetch('peatio:daemons:statuses', expires_in: 3.minute) do
        Daemons::Rails::Monitoring.statuses
      end
    end
  end

  def initialize(currency)
    @currency = currency
  end

  def channel
    "market-#{@currency}-global"
  end

  attr_accessor :currency

  def self.[](market)
    if market.is_a? Market
      self.new(market.id)
    else
      self.new(market)
    end
  end

  def key(key, interval=5)
    seconds  = Time.now.to_i
    time_key = seconds - (seconds % interval)
    "peatio:#{@currency}:#{key}:#{time_key}"
  end

  def asks
    Rails.cache.read("peatio:#{currency}:depth:asks") || []
  end

  def bids
    Rails.cache.read("peatio:#{currency}:depth:bids") || []
  end

  def default_ticker
    {low: ZERO, high: ZERO, last: ZERO, volume: ZERO}
  end

  def ticker
    ticker           = Rails.cache.read("peatio:#{currency}:ticker") || default_ticker
    open = Rails.cache.read("peatio:#{currency}:ticker:open") || ticker[:last]
    best_buy_price   = bids.first && bids.first[0] || ZERO
    best_sell_price  = asks.first && asks.first[0] || ZERO

    ticker.merge({
      open: open,
      volume: h24_volume,
      sell: best_sell_price,
      buy: best_buy_price,
      at: at
    })
  end

  def h24_volume
    Rails.cache.fetch key('h24_volume', 5), expires_in: 24.hours do
      Trade.with_currency(currency).h24.sum(:volume) || ZERO
    end
  end

  def trades
    Rails.cache.read("peatio:#{currency}:trades") || []
  end

  def trigger_orderbook
```

```
    data = {asks: asks, bids: bids}
    Pusher.trigger_async(channel, "update", data)
  end

  def trigger_trades(trades)
    Pusher.trigger_async(channel, "trades", trades: trades)
  end

  def at
    @at ||= DateTime.now.to_i
  end
end

146:F:\git\coin\exchange\peatio-master\app\models\identity.rb
class Identity < OmniAuth::Identity::Models::ActiveRecord
  auth_key :email
  attr_accessor :old_password

  MAX_LOGIN_ATTEMPTS = 5

  validates :email, presence: true, uniqueness: true, email: true
  validates :password, presence: true, length: { minimum: 6, maximum: 64 }
  validates :password_confirmation, presence: true, length: { minimum: 6, maximum: 64 }

  before_validation :sanitize

  def increment_retry_count
    self.retry_count = (retry_count || 0) + 1
  end

  def too_many_failed_login_attempts
    retry_count.present? && retry_count >= MAX_LOGIN_ATTEMPTS
  end

  private

  def sanitize
    self.email.try(:downcase!)
  end

end
```

147:F:\git\coin\exchange\peatio-master\app\models\id_document.rb

```
class IdDocument < ActiveRecord::Base
  extend Enumerize
  include AASM
  include AASM::Locking

  has_one :id_document_file, class_name: 'Asset::IdDocumentFile', as: :attachable
  accepts_nested_attributes_for :id_document_file

  has_one :id_bill_file, class_name: 'Asset::IdBillFile', as: :attachable
  accepts_nested_attributes_for :id_bill_file

  belongs_to :member

  validates_presence_of :name, :id_document_type, :id_document_number, :id_bill_type,
allow_nil: true
  validates_uniqueness_of :member

  enumerize :id_document_type, in: {id_card: 0, passport: 1, driver_license: 2}
  enumerize :id_bill_type,     in: {bank_statement: 0, tax_bill: 1}

  alias_attribute :full_name, :name

  aasm do
    state :unverified, initial: true
    state :verifying
    state :verified

    event :submit do
      transitions from: :unverified, to: :verifying
    end

    event :approve do
      transitions from: [:unverified, :verifying],  to: :verified
    end

    event :reject do
      transitions from: [:verifying, :verified],  to: :unverified
    end
  end
end
```

148:F:\git\coin\exchange\peatio-master\app\models\market.rb

```
# People exchange commodities in markets. Each market focuses on certain
# commodity pair `{A, B}`. By convention, we call people exchange A for B
# *sellers* who submit *ask* orders, and people exchange B for A *buyers*
# who submit *bid* orders.
#
# ID of market is always in the form "#{B}#{A}". For example, in 'btccny'
# market, the commodity pair is `{btc, cny}`. Sellers sell out _btc_ for
# _cny_, buyers buy in _btc_ with _cny_. _btc_ is the `base_unit`, while
# _cny_ is the `quote_unit`.

class Market < ActiveYamlBase
  field :visible, default: true

  attr :name

  self.singleton_class.send :alias_method, :all_with_invisible, :all
  def self.all
    all_with_invisible.select &:visible
  end

  def self.enumerize
    all_with_invisible.inject({}) {|hash, i| hash[i.id.to_sym] = i.code; hash }
  end

  def self.to_hash
    return @markets_hash if @markets_hash

    @markets_hash = {}
    all.each {|m| @markets_hash[m.id.to_sym] = m.unit_info }
    @markets_hash
  end

  def initialize(*args)
    super

    raise "missing base_unit or quote_unit: #{args}" unless base_unit.present? &&
quote_unit.present?
    @name = self[:name] || "#{base_unit}/#{quote_unit}".upcase
  end

  def latest_price
    Trade.latest_price(id.to_sym)
  end

  # type is :ask or :bid
  def fix_number_precision(type, d)
    digits = send(type)['fixed']
    d.round digits, 2
  end

  # shortcut of global access
  def bids;   global.bids   end
  def asks;   global.asks   end
  def trades; global.trades end
  def ticker; global.ticker end

  def to_s
    id
  end

  def ask_currency
    Currency.find_by_code(ask["currency"])
  end

  def bid_currency
    Currency.find_by_code(bid["currency"])
  end

  def scope?(account_or_currency)
    code = if account_or_currency.is_a? Account
             account_or_currency.currency
           elsif account_or_currency.is_a? Currency
             account_or_currency.code
           else
             account_or_currency
           end

    base_unit == code || quote_unit == code
  end

  def unit_info
    {name: name, base_unit: base_unit, quote_unit: quote_unit}
  end
```

```
  private

  def global
    @global || Global[self.id]
  end

end
```

149:F:\git\coin\exchange\peatio-master\app\models\matching\constants.rb

```
module Matching

  ZERO = 0.to_d unless defined?(ZERO)

  class DoubleSubmitError   < StandardError; end
  class InvalidOrderError   < StandardError; end
  class NotEnoughVolume     < StandardError; end
  class ExceedSumLimit      < StandardError; end
  class TradeExecutionError < StandardError; end

end
```

150:F:\git\coin\exchange\peatio-master\app\models\matching\engine.rb

```
module Matching
  class Engine

    attr :orderbook, :mode, :queue
    delegate :ask_orders, :bid_orders, to: :orderbook

    def initialize(market, options={})
      @market    = market
      @orderbook = OrderBookManager.new(market.id)

      # Engine is able to run in different mode:
      # dryrun: do the match, do not publish the trades
      # run:    do the match, publish the trades (default)
      shift_gears(options[:mode] || :run)
    end

    def submit(order)
      book, counter_book = orderbook.get_books order.type
      match order, counter_book
  add_or_cancel order, book
rescue
  Rails.logger.fatal "Failed to submit order #{order.label}: #{$!}"
  Rails.logger.fatal $!.backtrace.join("\n")
end

def cancel(order)
  book, counter_book = orderbook.get_books order.type
  if removed_order = book.remove(order)
    publish_cancel removed_order, "cancelled by user"
  else
    Rails.logger.warn "Cannot find order##{order.id} to cancel, skip."
  end
rescue
  Rails.logger.fatal "Failed to cancel order #{order.label}: #{$!}"
  Rails.logger.fatal $!.backtrace.join("\n")
end

def limit_orders
  { ask: ask_orders.limit_orders,
    bid: bid_orders.limit_orders }
end

def market_orders
  { ask: ask_orders.market_orders,
    bid: bid_orders.market_orders }
end

def shift_gears(mode)
  case mode
  when :dryrun
    @queue = []
    class <<@queue
      def enqueue(*args)
        push args
      end
    end
  when :run
    @queue = AMQPQueue
  else
    raise "Unrecognized mode: #{mode}"
  end
```

```
    @mode = mode
  end

  private

  def match(order, counter_book)
    return if order.filled?

    counter_order = counter_book.top
    return unless counter_order

    if trade = order.trade_with(counter_order, counter_book)
      counter_book.fill_top *trade
      order.fill *trade

      publish order, counter_order, trade

      match order, counter_book
    end
  end

  def add_or_cancel(order, book)
    return if order.filled?
    order.is_a?(LimitOrder) ?
      book.add(order) : publish_cancel(order, "fill or kill market order")
  end

  def publish(order, counter_order, trade)
    ask, bid = order.type == :ask ? [order, counter_order] : [counter_order, order]

    price  = @market.fix_number_precision :bid, trade[0]
    volume = @market.fix_number_precision :ask, trade[1]
    funds  = trade[2]

    Rails.logger.info "[#{@market.id}] new trade - ask: #{ask.label} bid: #{bid.label} price: #{price}
volume: #{volume} funds: #{funds}"

    @queue.enqueue(
      :trade_executor,
      {market_id: @market.id, ask_id: ask.id, bid_id: bid.id, strike_price: price, volume: volume,
funds: funds},
      {persistent: false}
    )
  end

  def publish_cancel(order, reason)
    Rails.logger.info "[#{@market.id}] cancel order ##{order.id} - reason: #{reason}"
    @queue.enqueue(
      :order_processor,
      {action: 'cancel', order: order.attributes},
      {persistent: false}
    )
  end

 end
end

151:F:\git\coin\exchange\peatio-master\app\models\matching\executor.rb
require_relative 'constants'

module Matching
 class Executor

  def initialize(payload)
    @payload = payload
    @market  = Market.find payload[:market_id]
    @price   = BigDecimal.new payload[:strike_price]
    @volume  = BigDecimal.new payload[:volume]
    @funds   = BigDecimal.new payload[:funds]
  end

  def execute!
    retry_on_error(5) { create_trade_and_strike_orders }
    publish_trade
    @trade
  end

  private

  def valid?
    return false if @ask.ord_type == 'limit' && @ask.price > @price
    return false if @bid.ord_type == 'limit' && @bid.price < @price
    @funds > ZERO && [@ask.volume, @bid.volume].min >= @volume
  end

  def trend
    @price >= @market.latest_price ? 'up' : 'down'
  end

  # in worst condition, the method will run 1+retry_count times then fail
  def retry_on_error(retry_count, &block)
    block.call
  rescue ActiveRecord::StatementInvalid
    # cope with "Mysql2::Error: Deadlock found ..." exception
    if retry_count > 0
      sleep 0.2
      retry_count -= 1
      puts "Retry trade execution (#{retry_count} retry left) .."
      retry
    else
      puts "Failed to execute trade: #{@payload.inspect}"
      raise $!
    end
  end

  def create_trade_and_strike_orders
    ActiveRecord::Base.transaction do
      @ask = OrderAsk.lock(true).find(@payload[:ask_id])
      @bid = OrderBid.lock(true).find(@payload[:bid_id])

      raise TradeExecutionError.new({ask: @ask, bid: @bid, price: @price, volume: @volume,
funds: @funds}) unless valid?

      @trade = Trade.create!(ask_id: @ask.id, ask_member_id: @ask.member_id,
                             bid_id: @bid.id, bid_member_id: @bid.member_id,
                             price: @price, volume: @volume, funds: @funds,
                             currency: @market.id.to_sym, trend: trend)

      @bid.strike @trade
      @ask.strike @trade
    end

    # TODO: temporary fix, can be removed after pusher -> polling refactoring
    if @trade.ask_member_id == @trade.bid_member_id
      @ask.hold_account.reload.trigger
      @bid.hold_account.reload.trigger
    end
  end

  def publish_trade
    AMQPQueue.publish(
      :trade,
      @trade.as_json,
      { headers: {
          market: @market.id,
          ask_member_id: @ask.member_id,
          bid_member_id: @bid.member_id
        }
      }
    )
  end

 end
end
```

152:F:\git\coin\exchange\peatio-master\app\models\matching\limit_order.rb

```
require_relative 'constants'

module Matching
  class LimitOrder
    attr :id, :timestamp, :type, :price, :market
    attr_accessor :volume

    def initialize(attrs)
      @id         = attrs[:id]
      @timestamp  = attrs[:timestamp]
      @type       = attrs[:type].to_sym
      @volume     = attrs[:volume].to_d
      @price      = attrs[:price].to_d
      @market     = Market.find attrs[:market]

      raise InvalidOrderError.new(attrs) unless valid?(attrs)
    end

    def trade_with(counter_order, counter_book)
      if counter_order.is_a?(LimitOrder)
        if crossed?(counter_order.price)
          trade_price  = counter_order.price
          trade_volume = [volume, counter_order.volume].min
          trade_funds  = trade_price*trade_volume
          [trade_price, trade_volume, trade_funds]
        end
      else
        trade_volume = [volume, counter_order.volume, counter_order.volume_limit(price)].min
        trade_funds  = price*trade_volume
        [price, trade_volume, trade_funds]
      end
    end

    def fill(trade_price, trade_volume, trade_funds)
      raise NotEnoughVolume if trade_volume > @volume
      @volume -= trade_volume
    end

    def filled?
      volume <= ZERO
    end

    def crossed?(price)
      if type == :ask
        price >= @price # if people offer price higher or equal than ask limit
      else
        price <= @price # if people offer price lower or equal than bid limit
      end
    end

    def label
      "%d/$%s/%s" % [id, price.to_s('F'), volume.to_s('F')]
    end

    def valid?(attrs)
      return false unless [:ask, :bid].include?(type)
      id && timestamp && market && price > ZERO
    end

    def attributes
      { id: @id,
        timestamp: @timestamp,
        type: @type,
        volume: @volume,
        price: @price,
        market: @market.id,
        ord_type: 'limit' }
    end

  end
end
```

153:F:\git\coin\exchange\peatio-master\app\models\matching\market_order.rb

```
require_relative 'constants'

module Matching
  class MarketOrder
    attr :id, :timestamp, :type, :locked, :market
    attr_accessor :volume

    def initialize(attrs)
      @id         = attrs[:id]
      @timestamp  = attrs[:timestamp]
      @type       = attrs[:type].to_sym
      @locked     = attrs[:locked].to_d
      @volume     = attrs[:volume].to_d
      @market     = Market.find attrs[:market]

      raise ::Matching::InvalidOrderError.new(attrs) unless valid?(attrs)
    end

    def trade_with(counter_order, counter_book)
      if counter_order.is_a?(LimitOrder)
        trade_price  = counter_order.price
        trade_volume = [volume, volume_limit(trade_price), counter_order.volume].min
        trade_funds  = trade_price*trade_volume
        [trade_price, trade_volume, trade_funds]
      elsif price = counter_book.best_limit_price
        trade_price  = price
        trade_volume = [volume, volume_limit(trade_price), counter_order.volume,
counter_order.volume_limit(trade_price)].min
        trade_funds  = trade_price*trade_volume
        [trade_price, trade_volume, trade_funds]
      end
    end
    def volume_limit(trade_price)
      type == :ask ? locked : locked/trade_price
    end

    def fill(trade_price, trade_volume, trade_funds)
      raise NotEnoughVolume if trade_volume > @volume
      @volume -= trade_volume

      funds = type == :ask ? trade_volume : trade_funds
      raise ExceedSumLimit if funds > @locked
      @locked -= funds
    end

    def filled?
      volume <= ZERO || locked <= ZERO
    end

    def label
      "%d/%s" % [id, volume.to_s('F')]
    end

    def valid?(attrs)
      return false unless [:ask, :bid].include?(type)
      return false if attrs[:price].present? # should have no limit price
      id && timestamp && market && locked > ZERO
    end

    def attributes
      { id: @id,
        timestamp: @timestamp,
        type: @type,
        locked: @locked,
        volume: @volume,
        market: @market.id,
        ord_type: 'market' }
    end

  end
end
```

154:F:\git\coin\exchange\peatio-master\app\models\matching\order_book.rb

```
require_relative 'constants'

module Matching
  class OrderBook

    attr :side

    def initialize(market, side, options={})
      @market = market
      @side   = side.to_sym
      @limit_orders = RBTree.new
      @market_orders = RBTree.new

      @broadcast = options.has_key?(:broadcast) ? options[:broadcast] : true
      broadcast(action: 'new', market: @market, side: @side)

      singleton = class<<self;self;end
      singleton.send :define_method, :limit_top, self.class.instance_method("#{@side}_limit_top")
    end

    def best_limit_price
      limit_top.try(:price)
    end

    def top
      @market_orders.empty? ? limit_top : @market_orders.first[1]
    end

    def fill_top(trade_price, trade_volume, trade_funds)
      order = top
      raise "No top order in empty book." unless order

      order.fill trade_price, trade_volume, trade_funds
      if order.filled?
        remove order
      else
        broadcast(action: 'update', order: order.attributes)
      end
    end

    def find(order)
      case order
      when LimitOrder
        @limit_orders[order.price].find(order.id)
      when MarketOrder
        @market_orders[order.id]
      end
    end

    def add(order)
      raise InvalidOrderError, "volume is zero" if order.volume <= ZERO

      case order
      when LimitOrder
        @limit_orders[order.price] ||= PriceLevel.new(order.price)
        @limit_orders[order.price].add order
      when MarketOrder
        @market_orders[order.id] = order
      else
        raise ArgumentError, "Unknown order type"
      end

      broadcast(action: 'add', order: order.attributes)
    end

    def remove(order)
      case order
      when LimitOrder
        remove_limit_order(order)
      when MarketOrder
        remove_market_order(order)
      else
        raise ArgumentError, "Unknown order type"
      end
    end

    def limit_orders
      orders = {}
      @limit_orders.keys.each {|k| orders[k] = @limit_orders[k].orders }
      orders
    end

    def market_orders
      @market_orders.values
```

```
end

private

def remove_limit_order(order)
  price_level = @limit_orders[order.price]
  return unless price_level

  order = price_level.find order.id # so we can return fresh order
  return unless order

  price_level.remove order
  @limit_orders.delete(order.price) if price_level.empty?

  broadcast(action: 'remove', order: order.attributes)
  order
end

def remove_market_order(order)
  if order = @market_orders[order.id]
    @market_orders.delete order.id
    broadcast(action: 'remove', order: order.attributes)
    order
  end
end

def ask_limit_top # lowest price wins
  return if @limit_orders.empty?
  price, level = @limit_orders.first
  level.top
end

def bid_limit_top # highest price wins
  return if @limit_orders.empty?
  price, level = @limit_orders.last
  level.top
end

def broadcast(data)
  return unless @broadcast
  Rails.logger.debug "orderbook broadcast: #{data.inspect}"
  AMQPQueue.enqueue(:slave_book, data, {persistent: false})
```

```
    end

  end
end
```

155:F:\git\coin\exchange\peatio-master\app\models\matching\order_book_manager.rb

```
module Matching
  class OrderBookManager

    attr :ask_orders, :bid_orders

    def self.build_order(attrs)
      attrs.symbolize_keys!

      raise ArgumentError, "Missing ord_type: #{attrs.inspect}" unless attrs[:ord_type].present?

      klass = ::Matching.const_get "#{attrs[:ord_type]}_order".camelize
      klass.new attrs
    end

    def initialize(market, options={})
      @market     = market
      @ask_orders = OrderBook.new(market, :ask, options)
      @bid_orders = OrderBook.new(market, :bid, options)
    end

    def get_books(type)
      case type
      when :ask
        [@ask_orders, @bid_orders]
      when :bid
        [@bid_orders, @ask_orders]
      end
    end

  end
end
```

156:F:\git\coin\exchange\peatio-master\app\models\matching\price_level.rb

```
module Matching
  class PriceLevel
```

```
    attr :price, :orders

    def initialize(price)
      @price  = price
      @orders = []
    end

    def top
      @orders.first
    end

    def empty?
      @orders.empty?
    end

    def add(order)
      @orders << order
    end

    def remove(order)
      @orders.delete_if {|o| o.id == order.id }
    end

    def find(id)
      @orders.find {|o| o.id == id }
    end

  end
end
```

157:F:\git\coin\exchange\peatio-master\app\models\member.rb

```
class Member < ActiveRecord::Base
  acts_as_taggable
  acts_as_reader

  has_many :orders
  has_many :accounts
  has_many :payment_addresses, through: :accounts
  has_many :withdraws
  has_many :fund_sources
  has_many :deposits
  has_many :api_tokens
  has_many :two_factors
  has_many :tickets, foreign_key: 'author_id'
  has_many :comments, foreign_key: 'author_id'
  has_many :signup_histories

  has_one :id_document

  has_many :authentications, dependent: :destroy

  scope :enabled, -> { where(disabled: false) }

  delegate :activated?, to: :two_factors, prefix: true, allow_nil: true
  delegate :name,       to: :id_document, allow_nil: true
  delegate :full_name,  to: :id_document, allow_nil: true
  delegate :verified?,  to: :id_document, prefix: true, allow_nil: true

  before_validation :sanitize, :generate_sn

  validates :sn, presence: true
  validates :display_name, uniqueness: true, allow_blank: true
  validates :email, email: true, uniqueness: true, allow_nil: true

  before_create :build_default_id_document
  after_create  :touch_accounts
  after_update :resend_activation
  after_update :sync_update

  class << self
    def from_auth(auth_hash)
      locate_auth(auth_hash) || locate_email(auth_hash) || create_from_auth(auth_hash)
    end

    def current
      Thread.current[:user]
    end

    def current=(user)
      Thread.current[:user] = user
    end

    def admins
      Figaro.env.admin.split(',')
```

```
end

def search(field: nil, term: nil)
  result = case field
           when 'email'
             where('members.email LIKE ?', "%#{term}%")
           when 'phone_number'
             where('members.phone_number LIKE ?', "%#{term}%")
           when 'name'
             joins(:id_document).where('id_documents.name LIKE ?', "%#{term}%")
           when 'wallet_address'
             members = joins(:fund_sources).where('fund_sources.uid' => term)
             if members.empty?
              members = joins(:payment_addresses).where('payment_addresses.address' => term)
             end
             members
           else
             all
           end

  result.order(:id).reverse_order
end

private

def locate_auth(auth_hash)
  Authentication.locate(auth_hash).try(:member)
end

def locate_email(auth_hash)
  return nil if auth_hash['info']['email'].blank?
  member = find_by_email(auth_hash['info']['email'])
  return nil unless member
  member.add_auth(auth_hash)
  member
end

def create_from_auth(auth_hash)
  member = create(email: auth_hash['info']['email'], nickname: auth_hash['info']['nickname'],
                  activated: false)
  member.add_auth(auth_hash)
  member.send_activation if auth_hash['provider'] == 'identity'
```

```
    member
  end
end


def create_auth_for_identity(identity)
  self.authentications.create(provider: 'identity', uid: identity.id)
end

def trades
  Trade.where('bid_member_id = ? OR ask_member_id = ?', id, id)
end

def active!
  update activated: true
end

def update_password(password)
  identity.update password: password, password_confirmation: password
  send_password_changed_notification
end

def admin?
  @is_admin ||= self.class.admins.include?(self.email)
end

def add_auth(auth_hash)
  authentications.build_auth(auth_hash).save
end

def trigger(event, data)
  AMQPQueue.enqueue(:pusher_member, {member_id: id, event: event, data: data})
end

def notify(event, data)
  ::Pusher["private-#{sn}"].trigger_async event, data
end

def to_s
  "#{name || email} - #{sn}"
end
```

```
def gravatar
  "//gravatar.com/avatar/" + Digest::MD5.hexdigest(email.strip.downcase) + "?d=retro"
end

def initial?
  name? and !name.empty?
end

def get_account(currency)
  account = accounts.with_currency(currency.to_sym).first

  if account.nil?
    touch_accounts
    account = accounts.with_currency(currency.to_sym).first
  end

  account
end
alias :ac :get_account

def touch_accounts
  less = Currency.codes - self.accounts.map(&:currency).map(&:to_sym)
  less.each do |code|
    self.accounts.create(currency: code, balance: 0, locked: 0)
  end
end

def identity
  authentication = authentications.find_by(provider: 'identity')
  authentication ? Identity.find(authentication.uid) : nil
end

def auth(name)
  authentications.where(provider: name).first
end

def auth_with?(name)
  auth(name).present?
end

def remove_auth(name)
  identity.destroy if name == 'identity'
  auth(name).destroy
end

def send_activation
  Token::Activation.create(member: self)
end

def send_password_changed_notification
  MemberMailer.reset_password_done(self.id).deliver

  if sms_two_factor.activated?
    sms_message = I18n.t('sms.password_changed', email: self.email)
    AMQPQueue.enqueue(:sms_notification, phone: phone_number, message: sms_message)
  end
end

def unread_comments
  ticket_ids = self.tickets.open.collect(&:id)
  if ticket_ids.any?
    Comment.where(ticket_id: [ticket_ids]).where("author_id <> ?", self.id).unread_by(self).to_a
  else
    []
  end
end

def app_two_factor
  two_factors.by_type(:app)
end

def sms_two_factor
  two_factors.by_type(:sms)
end

def as_json(options = {})
  super(options).merge({
    "name" => self.name,
    "app_activated" => self.app_two_factor.activated?,
    "sms_activated" => self.sms_two_factor.activated?,
    "memo" => self.id
  })
end
```

```
  private

  def sanitize
    self.email.try(:downcase!)
  end

  def generate_sn
    self.sn and return
    begin
      self.sn = "PEA#{ROTP::Base32.random_base32(8).upcase}TIO"
    end while Member.where(:sn => self.sn).any?
  end

  def build_default_id_document
    build_id_document
    true
  end

  def resend_activation
    self.send_activation if self.email_changed?
  end

  def sync_update
    ::Pusher["private-#{sn}"].trigger_async('members', { type: 'update', id: self.id, attributes:
self.changes_attributes_as_json })
  end
end

158:F:\git\coin\exchange\peatio-master\app\models\member_tag.rb
class MemberTag < ActiveYamlBase
end

159:F:\git\coin\exchange\peatio-master\app\models\order.rb
class Order < ActiveRecord::Base
  extend Enumerize

  enumerize :bid, in: Currency.enumerize
  enumerize :ask, in: Currency.enumerize
  enumerize :currency, in: Market.enumerize, scope: true
  enumerize :state, in: {:wait => 100, :done => 200, :cancel => 0}, scope: true

  ORD_TYPES = %w(market limit)
  enumerize :ord_type, in: ORD_TYPES, scope: true

  SOURCES = %w(Web APIv2 debug)
  enumerize :source, in: SOURCES, scope: true

  after_commit :trigger
  before_validation :fix_number_precision, on: :create

  validates_presence_of :ord_type, :volume, :origin_volume, :locked, :origin_locked
  validates_numericality_of :origin_volume, :greater_than => 0

  validates_numericality_of :price, greater_than: 0, allow_nil: false,
    if: "ord_type == 'limit'"
  validate :market_order_validations, if: "ord_type == 'market'"

  WAIT = 'wait'
  DONE = 'done'
  CANCEL = 'cancel'

  ATTRIBUTES = %w(id at market kind price state state_text volume origin_volume)

  belongs_to :member
  attr_accessor :total

  scope :done, -> { with_state(:done) }
  scope :active, -> { with_state(:wait) }
  scope :position, -> { group("price").pluck(:price, 'sum(volume)') }
  scope :best_price, ->(currency) { where(ord_type:
'limit').active.with_currency(currency).matching_rule.position }

  def funds_used
    origin_locked - locked
  end

  def fee
    config[kind.to_sym]["fee"]
  end

  def config
    @config ||= Market.find(currency)
  end
  def trigger
    return unless member

    json = Jbuilder.encode do |json|
      json.(self, *ATTRIBUTES)
    end
    member.trigger('order', json)
  end

  def strike(trade)
    raise "Cannot strike on cancelled or done order. id: #{id}, state: #{state}" unless state ==
Order::WAIT

    real_sub, add = get_account_changes trade
    real_fee      = add * fee
    real_add      = add - real_fee

    hold_account.unlock_and_sub_funds \
      real_sub, locked: real_sub,
      reason: Account::STRIKE_SUB, ref: trade

    expect_account.plus_funds \
      real_add, fee: real_fee,
      reason: Account::STRIKE_ADD, ref: trade

    self.volume         -= trade.volume
    self.locked         -= real_sub
    self.funds_received += add
    self.trades_count   += 1

    if volume.zero?
      self.state = Order::DONE

      # unlock not used funds
      hold_account.unlock_funds locked,
        reason: Account::ORDER_FULLFILLED, ref: trade unless locked.zero?
    elsif ord_type == 'market' && locked.zero?
      # partially filled market order has run out its locked fund
      self.state = Order::CANCEL
    end

    self.save!
  end

  def kind
    type.underscore[-3, 3]
  end

  def self.head(currency)
    active.with_currency(currency.downcase).matching_rule.first
  end

  def at
    created_at.to_i
  end

  def market
    currency
  end

  def to_matching_attributes
    { id: id,
      market: market,
      type: type[-3, 3].downcase.to_sym,
      ord_type: ord_type,
      volume: volume,
      price: price,
      locked: locked,
      timestamp: created_at.to_i }
  end

  def fix_number_precision
    self.price = config.fix_number_precision(:bid, price.to_d) if price

    if volume
      self.volume = config.fix_number_precision(:ask, volume.to_d)
      self.origin_volume = origin_volume.present? ? config.fix_number_precision(:ask,
origin_volume.to_d) : volume
    end
  end

  private

  def market_order_validations
```

```
    errors.add(:price, 'must not be present') if price.present?
  end

  FUSE = '0.9'.to_d
  def estimate_required_funds(price_levels)
    required_funds = Account::ZERO
    expected_volume = volume

    start_from, _ = price_levels.first
    filled_at     = start_from

    until expected_volume.zero? || price_levels.empty?
      level_price, level_volume = price_levels.shift
      filled_at = level_price

      v = [expected_volume, level_volume].min
      required_funds += yield level_price, v
      expected_volume -= v
    end

    raise "Market is not deep enough" unless expected_volume.zero?
    raise "Volume too large" if (filled_at-start_from).abs/start_from > FUSE

    required_funds
  end

end

160:F:\git\coin\exchange\peatio-master\app\models\order_ask.rb
class OrderAsk < Order

  has_many :trades, foreign_key: 'ask_id'

  scope :matching_rule, -> { order('price ASC, created_at ASC') }

  def get_account_changes(trade)
    [trade.volume, trade.funds]
  end

  def hold_account
    member.get_account(ask)
  end
  def expect_account
    member.get_account(bid)
  end

  def avg_price
    return ::Trade::ZERO if funds_used.zero?
    config.fix_number_precision(:bid, funds_received / funds_used)
  end

  def compute_locked
    case ord_type
    when 'limit'
      volume
    when 'market'
      estimate_required_funds(Global[currency].bids) {|p, v| v}
    end
  end

end
```

161:F:\git\coin\exchange\peatio-master\app\models\order_bid.rb

```
class OrderBid < Order

  has_many :trades, foreign_key: 'bid_id'

  scope :matching_rule, -> { order('price DESC, created_at ASC') }

  def get_account_changes(trade)
    [trade.funds, trade.volume]
  end

  def hold_account
    member.get_account(bid)
  end

  def expect_account
    member.get_account(ask)
  end

  def avg_price
    return ::Trade::ZERO if funds_received.zero?
    config.fix_number_precision(:bid, funds_used / funds_received)
  end

  LOCKING_BUFFER_FACTOR = '1.1'.to_d
  def compute_locked
    case ord_type
    when 'limit'
      price*volume
    when 'market'
      funds = estimate_required_funds(Global[currency].asks) {|p, v| p*v }
      funds*LOCKING_BUFFER_FACTOR
    end
  end

end

162:F:\git\coin\exchange\peatio-master\app\models\partial_tree.rb
class PartialTree < ActiveRecord::Base

  belongs_to :account
  belongs_to :proof

  serialize :json, JSON
  validates_presence_of :proof_id, :account_id, :json

end

163:F:\git\coin\exchange\peatio-master\app\models\payment_address.rb
class PaymentAddress < ActiveRecord::Base
  include Currencible
  belongs_to :account

  after_commit :gen_address, on: :create

  has_many :transactions, class_name: 'PaymentTransaction', foreign_key: 'address',
primary_key: 'address'

  validates_uniqueness_of :address, allow_nil: true

  def gen_address
    payload = { payment_address_id: id, currency: currency }
    attrs   = { persistent: true }
    AMQPQueue.enqueue(:deposit_coin_address, payload, attrs)
  end

  def memo
    address && address.split('|', 2).last
  end

  def deposit_address
    currency_obj[:deposit_account] || address
  end

  def as_json(options = {})
    {
      account_id: account_id,
      deposit_address: deposit_address
    }.merge(options)
  end

  def trigger_deposit_address
    ::Pusher["private-#{account.member.sn}"].trigger_async('deposit_address', {type: 'create',
attributes: as_json})
  end

  def self.construct_memo(obj)
    member = obj.is_a?(Account) ? obj.member : obj
    checksum = member.created_at.to_i.to_s[-3..-1]
    "#{member.id}#{checksum}"
  end

  def self.destruct_memo(memo)
    member_id = memo[0...-3]
    checksum  = memo[-3..-1]

    member = Member.find_by_id member_id
    return nil unless member
    return nil unless member.created_at.to_i.to_s[-3..-1] == checksum
    member
  end

  def to_json
    {address: deposit_address}
  end
```

```
end

164:F:\git\coin\exchange\peatio-master\app\models\payment_transaction\normal.rb
class PaymentTransaction::Normal < PaymentTransaction
  # Default payment transaction captures all bitcoin-like transactions.

  validates_presence_of :txout
  validates_uniqueness_of :txout, scope: :txid

end

165:F:\git\coin\exchange\peatio-master\app\models\payment_transaction.rb
class PaymentTransaction < ActiveRecord::Base
  extend Enumerize

  include AASM
  include AASM::Locking
  include Currencible

  STATE = [:unconfirm, :confirming, :confirmed]
  enumerize :aasm_state, in: STATE, scope: true

  validates_presence_of :txid

  has_one :deposit
  belongs_to :payment_address, foreign_key: 'address', primary_key: 'address'
  has_one :account, through: :payment_address
  has_one :member, through: :account

  after_update :sync_update

  aasm :whiny_transitions => false do
    state :unconfirm, initial: true
    state :confirming, after_commit: :deposit_accept
    state :confirmed, after_commit: :deposit_accept

    event :check do |e|
      before :refresh_confirmations

      transitions :from => [:unconfirm, :confirming], :to => :confirming, :guard => :min_confirm?
      transitions :from => [:unconfirm, :confirming, :confirmed], :to => :confirmed, :guard =>
```

```
:max_confirm?
    end
  end

  def min_confirm?
    deposit.min_confirm?(confirmations)
  end

  def max_confirm?
    deposit.max_confirm?(confirmations)
  end

  def refresh_confirmations
    raw = CoinRPC[deposit.currency].gettransaction(txid)
    self.confirmations = raw[:confirmations]
    save!
  end

  def deposit_accept
    if deposit.may_accept?
      deposit.accept!
    end
  end

  private

  def sync_update
    if self.confirmations_changed?
      ::Pusher["private-#{deposit.member.sn}"].trigger_async('deposits', { type: 'update', id:
self.deposit.id, attributes: {confirmations: self.confirmations}})
    end
  end
end

166:F:\git\coin\exchange\peatio-master\app\models\proof.rb
class Proof < ActiveRecord::Base
  include Currencible

  has_many :partial_trees

  serialize :root, JSON
  serialize :addresses, JSON
  validates_presence_of :root, :currency
  validates_numericality_of :balance, allow_nil: true, greater_than_or_equal_to: 0

  delegate :coin?, to: :currency_obj

  def self.current(code)
    proofs = with_currency(code)
    proofs.where('created_at <= ?', 1.day.ago).last || proofs.last
  end

  def ready!
    self.ready = true
    save!
  end

  def timestamp
    Time.at(root['timestamp']/1000) || updated_at
  end

  def partial_tree_of(account)
    partial_trees.where(account: account).first
  end

  def asset_sum
    addresses.reduce 0 do |memo, address|
      memo + address["balance"]
    end
  end

  def address_url(address)
    currency_obj.address_url(address)
  end

end

167:F:\git\coin\exchange\peatio-master\app\models\running_account.rb
class RunningAccount < ActiveRecord::Base
  include Currencible

  CATEGORY = {
    withdraw_fee:         0,
    trading_fee:          1,
    register_reward:      2,
    referral_code_reward: 3,
    deposit_reward:       4
  }

  enumerize :category, in: CATEGORY

  belongs_to :member
  belongs_to :source, polymorphic: true

end

168:F:\git\coin\exchange\peatio-master\app\models\signup_history.rb
class SignupHistory < ActiveRecord::Base

end

169:F:\git\coin\exchange\peatio-master\app\models\ticket.rb
class Ticket < ActiveRecord::Base
  include AASM
  include AASM::Locking
  acts_as_readable on: :created_at

  after_commit :send_notification, on: [:create]

  validates_with TicketValidator

  has_many :comments
  belongs_to :author, class_name: 'Member', foreign_key: 'author_id'

  scope :open, -> { where(aasm_state: :open) }
  scope :close, -> { where(aasm_state: :closed) }

  aasm whiny_transitions: false do
    state :open
    state :closed

    event :close do
      transitions from: :open, to: :closed
    end
    event :reopen do
      transitions from: :closed, to: :open
    end
  end

  def title_for_display(n = 60)
    title.blank? ? content.truncate(n) : title.truncate(n)
  end

  private

  def send_notification
    TicketMailer.author_notification(self.id).deliver
    TicketMailer.admin_notification(self.id).deliver
  end

end
```

170:F:\git\coin\exchange\peatio-master\app\models\token\activation.rb

```
class Token::Activation < ::Token
  after_create :send_token

  def confirm!
    super
    member.active!
  end

  private

  def send_token
    TokenMailer.activation(member.email, token).deliver
  end
end
```

171:F:\git\coin\exchange\peatio-master\app\models\token\reset_password.rb

```
class Token::ResetPassword < ::Token
  attr_accessor :email
  attr_accessor :password

  before_validation :set_member, on: :create

  validates_presence_of :email, on: :create
  validates :password, presence: true,
                       on: :update,
                       length: { minimum: 6, maximum: 64 }

  after_create :send_token

  def confirm!
    super
    member.update_password password
  end

  private

  def set_member
    if member = Member.find_by_email(self.email)
      self.member = member
    end
  end

  def send_token
    TokenMailer.reset_password(member.email, token).deliver
  end
end
```

172:F:\git\coin\exchange\peatio-master\app\models\token.rb

```
class Token < ActiveRecord::Base
  belongs_to :member

  before_validation :generate_token, on: :create

  validates_presence_of :member, :token
  validate :check_latest_send, on: :create

  scope :with_member, -> (id) { where(member_id: id) }
  scope :with_token, -> (token) { where(token: token) }
  scope :available, -> { where("expire_at > ? and is_used = ?", DateTime.now, false) }

  class << self
    def verify(token)
      with_token(token).available.any?
    end
    def for_member(member)
      token = find_or_create_by(member_id: member.id, is_used: false)

      if token.expired?
        token = create(member_id: member.id)
      end

      token
    end
  end

  def to_param
    self.token
  end

  def expired?
    expire_at <= Time.now
  end

  def confirm!
    self.update is_used: true
  end

  private

  def check_latest_send
    latest = self.class.available.with_member(self.member_id)
      .order(:created_at).reverse_order.first

    if latest && latest.created_at > 30.minutes.ago
      self.errors.add(:base, :too_soon)
    end
  end

  def generate_token
    self.token = SecureRandom.hex(16)
    self.expire_at = 30.minutes.from_now
  end
end
```

173:F:\git\coin\exchange\peatio-master\app\models\trade.rb

```
class Trade < ActiveRecord::Base
```

```
  extend ActiveHash::Associations::ActiveRecordExtensions
  ZERO = '0.0'.to_d

  extend Enumerize
  enumerize :trend, in: {:up => 1, :down => 0}
  enumerize :currency, in: Market.enumerize, scope: true

  belongs_to :market, class_name: 'Market', foreign_key: 'currency'
  belongs_to :ask, class_name: 'OrderAsk', foreign_key: 'ask_id'
  belongs_to :bid, class_name: 'OrderBid', foreign_key: 'bid_id'

  belongs_to :ask_member, class_name: 'Member', foreign_key: 'ask_member_id'
  belongs_to :bid_member, class_name: 'Member', foreign_key: 'bid_member_id'

  validates_presence_of :price, :volume, :funds

  scope :h24, -> { where("created_at > ?", 24.hours.ago) }

  attr_accessor :side

  alias_method :sn, :id

  class << self
    def latest_price(currency)
      with_currency(currency).order(:id).reverse_order
        .limit(1).first.try(:price) || "0.0".to_d
    end

    def filter(market, timestamp, from, to, limit, order)
      trades = with_currency(market).order(order)
      trades = trades.limit(limit) if limit.present?
      trades = trades.where('created_at <= ?', timestamp) if timestamp.present?
      trades = trades.where('id > ?', from) if from.present?
      trades = trades.where('id < ?', to) if to.present?
      trades
    end

    def for_member(currency, member, options={})
      trades = filter(currency, options[:time_to], options[:from], options[:to], options[:limit],
options[:order]).where("ask_member_id = ? or bid_member_id = ?", member.id, member.id)
      trades.each do |trade|
        trade.side = trade.ask_member_id == member.id ? 'ask' : 'bid'
      end
    end
  end

  def trigger_notify
    ask.member.notify 'trade', for_notify('ask')
    bid.member.notify 'trade', for_notify('bid')
  end

  def for_notify(kind=nil)
    {
      id:     id,
      kind:   kind || side,
      at:     created_at.to_i,
      price:  price.to_s  || ZERO,
      volume: volume.to_s || ZERO,
      market: currency
    }
  end

  def for_global
    {
      tid:    id,
      type:   trend == 'down' ? 'sell' : 'buy',
      date:   created_at.to_i,
      price:  price.to_s || ZERO,
      amount: volume.to_s || ZERO
    }
  end
end
```

174:F:\git\coin\exchange\peatio-master\app\models\two_factor\app.rb

```
class TwoFactor::App < ::TwoFactor

  def verify?
    return false if otp_secret.blank?

    rotp = ROTP::TOTP.new(otp_secret)

    if rotp.verify(otp)
      touch(:last_verify_at)
      true
    else
      errors.add :otp, :invalid
      false
    end
  end

  def uri
    totp = ROTP::TOTP.new(otp_secret)
    totp.provisioning_uri(member.email) + "&issuer=#{ENV['URL_HOST']}"
  end

  def now
    ROTP::TOTP.new(otp_secret).now
  end

  def refresh!
    return if activated?
    super
  end

  private

  def gen_code
    self.otp_secret = ROTP::Base32.random_base32
    self.refreshed_at = Time.new
  end

  def send_notification
    return if not self.activated_changed?

    if self.activated
      MemberMailer.google_auth_activated(member.id).deliver
    else
      MemberMailer.google_auth_deactivated(member.id).deliver
    end
  end

end
```

175:F:\git\coin\exchange\peatio-master\app\models\two_factor\email.rb

```
class TwoFactor::Email < ::TwoFactor
```

```
end

176:F:\git\coin\exchange\peatio-master\app\models\two_factor\sms.rb
class TwoFactor::Sms < ::TwoFactor
  attr_accessor :send_code_phase
  attr_accessor :country, :phone_number

  validates_presence_of :phone_number, if: :send_code_phase
  validate :valid_phone_number_for_country

  def verify?
    if !expired? && otp_secret == otp
      touch(:last_verify_at)
      refresh!
      true
    else
      if otp.blank?
        errors.add :otp, :blank
      else
        errors.add :otp, :invalid
      end
      false
    end
  end

  def sms_message
    I18n.t('sms.verification_code', code: otp_secret)
  end

  def send_otp
    refresh! if expired?
    update_phone_number_to_member if send_code_phase
    AMQPQueue.enqueue(:sms_notification, phone: member.phone_number, message:
sms_message)
  end

  private

  def valid_phone_number_for_country
    return if not send_code_phase

    if Phonelib.invalid_for_country?(phone_number, country)
```

```
      errors.add :phone_number, :invalid
    end
  end

  def country_code
    ISO3166::Country[country].try :country_code
  end

  def update_phone_number_to_member
    phone = Phonelib.parse([country_code, phone_number].join)
    member.update phone_number: phone.sanitized.to_s
  end

  def gen_code
    self.otp_secret = '%06d' % SecureRandom.random_number(1000000)
    self.refreshed_at = Time.now
  end

  def send_notification
    return if not self.activated_changed?

    if self.activated
      MemberMailer.sms_auth_activated(member.id).deliver
    else
      MemberMailer.sms_auth_deactivated(member.id).deliver
    end
  end
end
```

177:F:\git\coin\exchange\peatio-master\app\models\two_factor\wechat.rb

```
class TwoFactor::Wechat < ::TwoFactor

end
```

178:F:\git\coin\exchange\peatio-master\app\models\two_factor.rb

```
class TwoFactor < ActiveRecord::Base
  belongs_to :member

  before_validation :gen_code, on: :create
  after_update :send_notification

  validates_presence_of :member, :otp_secret, :refreshed_at
```

```ruby
attr_accessor :otp

SUBCLASS = ['app', 'sms', 'email', 'wechat']

validates_uniqueness_of :type, scope: :member_id

scope :activated, -> { where(activated: true) }

class << self
  def by_type(type)
    return if not SUBCLASS.include?(type.to_s)

    klass = "two_factor/#{type}".camelize.constantize
    klass.find_or_create_by(type: klass.name)
  end

  def activated?
    activated.any?
  end
end

def verify?
  msg = "#{self.class.name}#verify? is not implemented."
  raise NotImplementedError.new(msg)
end

def expired?
  Time.now >= 30.minutes.since(refreshed_at)
end

def refresh!
  gen_code
  save
end

def active!
  update activated: true, last_verify_at: Time.now
end

def deactive!
  update activated: false
  end

  private

  def gen_code
    msg = "#{self.class.name}#gen_code is not implemented."
    raise NotImplementedError.new(msg)
  end

  def send_notification
    msg = "#{self.class.name}#send_notification is not implemented."
    raise NotImplementedError.new(msg)
  end

end
```

179:F:\git\coin\exchange\peatio-master\app\models\withdraw.rb

```
class Withdraw < ActiveRecord::Base
  STATES = [:submitting, :submitted, :rejected, :accepted, :suspect, :processing,
            :done, :canceled, :almost_done, :failed]
  COMPLETED_STATES = [:done, :rejected, :canceled, :almost_done, :failed]

  extend Enumerize

  include AASM
  include AASM::Locking
  include Currencible

  has_paper_trail on: [:update, :destroy]

  enumerize :aasm_state, in: STATES, scope: true

  belongs_to :member
  belongs_to :account
  has_many :account_versions, as: :modifiable

  delegate :balance, to: :account, prefix: true
  delegate :key_text, to: :channel, prefix: true
  delegate :id, to: :channel, prefix: true
  delegate :name, to: :member, prefix: true
  delegate :coin?, :fiat?, to: :currency_obj
```

```
before_validation :fix_precision
before_validation :calc_fee
before_validation :set_account
after_create :generate_sn

after_update :sync_update
after_create :sync_create
after_destroy :sync_destroy

validates_with WithdrawBlacklistValidator

validates :fund_uid, :amount, :fee, :account, :currency, :member, presence: true

validates :fee, numericality: {greater_than_or_equal_to: 0}
validates :amount, numericality: {greater_than: 0}

validates :sum, presence: true, numericality: {greater_than: 0}, on: :create
validates :txid, uniqueness: true, allow_nil: true, on: :update

validate :ensure_account_balance, on: :create

scope :completed, -> { where aasm_state: COMPLETED_STATES }
scope :not_completed, -> { where.not aasm_state: COMPLETED_STATES }

def self.channel
  WithdrawChannel.find_by_key(name.demodulize.underscore)
end

def channel
  self.class.channel
end

def channel_name
  channel.key
end

alias_attribute :withdraw_id, :sn
alias_attribute :full_name, :member_name

def generate_sn
  id_part = sprintf '%04d', id
  date_part = created_at.localtime.strftime('%y%m%d%H%M')
  self.sn = "#{date_part}#{id_part}"
  update_column(:sn, sn)
end

aasm :whiny_transitions => false do
  state :submitting,  initial: true
  state :submitted,   after_commit: :send_email
  state :canceled,    after_commit: [:send_email]
  state :accepted
  state :suspect,     after_commit: :send_email
  state :rejected,    after_commit: :send_email
  state :processing,  after_commit: [:send_coins!, :send_email]
  state :almost_done
  state :done,        after_commit: [:send_email, :send_sms]
  state :failed,      after_commit: :send_email

  event :submit do
    transitions from: :submitting, to: :submitted
    after do
      lock_funds
    end
  end

  event :cancel do
    transitions from: [:submitting, :submitted, :accepted], to: :canceled
    after do
      after_cancel
    end
  end

  event :mark_suspect do
    transitions from: :submitted, to: :suspect
  end

  event :accept do
    transitions from: :submitted, to: :accepted
  end

  event :reject do
    transitions from: [:submitted, :accepted, :processing], to: :rejected
    after :unlock_funds
  end
  event :process do
    transitions from: :accepted, to: :processing
  end

  event :call_rpc do
    transitions from: :processing, to: :almost_done
  end

  event :succeed do
    transitions from: [:processing, :almost_done], to: :done

    before [:set_txid, :unlock_and_sub_funds]
  end

  event :fail do
    transitions from: :processing, to: :failed
  end
end

def cancelable?
  submitting? or submitted? or accepted?
end

def quick?
  sum <= currency_obj.quick_withdraw_max
end

def audit!
  with_lock do
    if account.examine
      accept
      process if quick?
    else
      mark_suspect
    end

    save!
  end
end

private
```

```
  def after_cancel
    unlock_funds unless aasm.from_state == :submitting
  end

  def lock_funds
    account.lock!
    account.lock_funds sum, reason: Account::WITHDRAW_LOCK, ref: self
  end

  def unlock_funds
    account.lock!
    account.unlock_funds sum, reason: Account::WITHDRAW_UNLOCK, ref: self
  end

  def unlock_and_sub_funds
    account.lock!
    account.unlock_and_sub_funds sum, locked: sum, fee: fee, reason: Account::WITHDRAW, ref:
self
  end

  def set_txid
    self.txid = @sn unless coin?
  end

  def send_email
    case aasm_state
    when 'submitted'
      WithdrawMailer.submitted(self.id).deliver
    when 'processing'
      WithdrawMailer.processing(self.id).deliver
    when 'done'
      WithdrawMailer.done(self.id).deliver
    else
      WithdrawMailer.withdraw_state(self.id).deliver
    end
  end

  def send_sms
    return true if not member.sms_two_factor.activated?

    sms_message = I18n.t('sms.withdraw_done', email: member.email,
                                         currency: currency_text,
                                         time: I18n.l(Time.now),
                                         amount: amount,
                                         balance: account.balance)

    AMQPQueue.enqueue(:sms_notification, phone: member.phone_number, message:
sms_message)
  end

  def send_coins!
    AMQPQueue.enqueue(:withdraw_coin, id: id) if coin?
  end

  def ensure_account_balance
    if sum.nil? or sum > account.balance
      errors.add :base, -> { I18n.t('activerecord.errors.models.withdraw.account_balance_is_poor') }
    end
  end

  def fix_precision
    if sum && currency_obj.precision
      self.sum = sum.round(currency_obj.precision, BigDecimal::ROUND_DOWN)
    end
  end

  def calc_fee
    if respond_to?(:set_fee)
      set_fee
    end

    self.sum ||= 0.0
    self.fee ||= 0.0
    self.amount = sum - fee
  end

  def set_account
    self.account = member.get_account(currency)
  end

  def self.resource_name
    name.demodulize.underscore.pluralize
  end
  def sync_update
    ::Pusher["private-#{member.sn}"].trigger_async('withdraws', { type: 'update', id: self.id,
attributes: self.changes_attributes_as_json })
  end

  def sync_create
    ::Pusher["private-#{member.sn}"].trigger_async('withdraws', { type: 'create', attributes:
self.as_json })
  end

  def sync_destroy
    ::Pusher["private-#{member.sn}"].trigger_async('withdraws', { type: 'destroy', id: self.id })
  end


end
```

180:F:\git\coin\exchange\peatio-master\app\models\withdraws\bank.rb

```
module Withdraws
  class Bank < ::Withdraw
    include ::AasmAbsolutely
    include ::Withdraws::Bankable
    include ::FundSourceable
  end
end
```

181:F:\git\coin\exchange\peatio-master\app\models\withdraws\satoshi.rb

```
module Withdraws
  class Satoshi < ::Withdraw
    include ::AasmAbsolutely
    include ::Withdraws::Coinable
    include ::FundSourceable
  end
end
```

182:F:\git\coin\exchange\peatio-master\app\models\withdraw_channel.rb

```
class WithdrawChannel < ActiveYamlBase
  include Channelable
  include HashCurrencible
  include International
  def blacklist
    self[:blacklist]
  end

  def as_json(options = {})
    super(options)['attributes'].merge({resource_name: key.pluralize})
  end

end

183:F:\git\coin\exchange\peatio-master\app\models\worker\deposit_coin.rb
module Worker
  class DepositCoin

    def process(payload, metadata, delivery_info)
      payload.symbolize_keys!

      sleep 0.5 # nothing result without sleep by query gettransaction api

      channel_key = payload[:channel_key]
      txid = payload[:txid]

      channel = DepositChannel.find_by_key(channel_key)
      raw     = get_raw channel, txid

      raw[:details].each_with_index do |detail, i|
        detail.symbolize_keys!
        deposit!(channel, txid, i, raw, detail)
      end
    end

    def deposit!(channel, txid, txout, raw, detail)
      return if detail[:account] != "payment" || detail[:category] != "receive"

      ActiveRecord::Base.transaction do
        unless PaymentAddress.where(currency: channel.currency_obj.id, address:
detail[:address]).first
          Rails.logger.info "Deposit address not found, skip. txid: #{txid}, txout: #{txout}, address:
#{detail[:address]}, amount: #{detail[:amount]}"
          return
        end
        return if PaymentTransaction::Normal.where(txid: txid, txout: txout).first

        tx = PaymentTransaction::Normal.create! \
          txid: txid,
          txout: txout,
          address: detail[:address],
          amount: detail[:amount].to_s.to_d,
          confirmations: raw[:confirmations],
          receive_at: Time.at(raw[:timereceived]).to_datetime,
          currency: channel.currency

        deposit = channel.kls.create! \
          payment_transaction_id: tx.id,
          txid: tx.txid,
          txout: tx.txout,
          amount: tx.amount,
          member: tx.member,
          account: tx.account,
          currency: tx.currency,
          confirmations: tx.confirmations

        deposit.submit!
      end
    rescue
      Rails.logger.error "Failed to deposit: #{$!}"
      Rails.logger.error "txid: #{txid}, txout: #{txout}, detail: #{detail.inspect}"
      Rails.logger.error $!.backtrace.join("\n")
    end

    def get_raw(channel, txid)
      channel.currency_obj.api.gettransaction(txid)
    end

  end
end

184:F:\git\coin\exchange\peatio-master\app\models\worker\deposit_coin_address.rb
module Worker
  class DepositCoinAddress

    def process(payload, metadata, delivery_info)
      payload.symbolize_keys!
      payment_address = PaymentAddress.find payload[:payment_address_id]
      return if payment_address.address.present?

      currency = payload[:currency]
      address  = CoinRPC[currency].getnewaddress("payment")

      if payment_address.update address: address
        ::Pusher["private-#{payment_address.account.member.sn}"].trigger_async('deposit_address',
{ type: 'create', attributes: payment_address.as_json})
      end
    end

  end
end
```

185:F:\git\coin\exchange\peatio-master\app\models\worker\email_notification.rb

```
module Worker
  class EmailNotification

    def process(payload, metadata, delivery_info)
      payload.symbolize_keys!
      set_locale(payload)

      mailer = payload[:mailer_class].constantize
      action = payload[:method]
      args   = payload[:args]

      message = mailer.send(:new, action, *args).message
      message.deliver
    end

    private

    def set_locale(payload)
      locale = payload[:locale]
      I18n.locale = locale if locale
    end

  end
end
```

186:F:\git\coin\exchange\peatio-master\app\models\worker\fund_stats.rb

```
module Worker
  class FundStats < Stats

    def initialize(currency)
      super()
      @currency = currency
    end

    def to_s
      "#{self.class.name} (#{@currency.code})"
    end

    def key_for(period)
      "peatio:stats:funds:#{@currency.code}:#{period}"
    end

    def point_1(from)
      to = from + 1.minute
      deposits = Deposit.with_aasm_state(:accepted).where(currency: @currency.id, created_at: 
from...to).pluck(:amount)
      withdraws = Withdraw.with_aasm_state(:done).where(currency: @currency.id, created_at: 
from...to).pluck(:amount)
      [from.to_i, deposits.size, deposits.sum.to_f, withdraws.size, withdraws.sum.to_f]
    end

    def point_n(from ,period)
      arr = point_1_set from, period
      deposits_count = arr.sum {|point| point[1] }
      deposits_amount = arr.sum {|point| point[2] }
      withdraws_count = arr.sum {|point| point[3] }
      withdraws_amount = arr.sum {|point| point[4] }
      [from.to_i, deposits_count, deposits_amount, withdraws_count, withdraws_amount]
    end

  end
end
```

187:F:\git\coin\exchange\peatio-master\app\models\worker\market_ticker.rb

```
module Worker
  class MarketTicker
```

```
FRESH_TRADES = 80

def initialize
  @tickers = {}
  @trades  = {}

  Market.all.each do |market|
    initialize_market_data market
  end
end

def process(payload, metadata, delivery_info)
  trade = Trade.new payload
  update_ticker trade
  update_latest_trades trade
end

def update_ticker(trade)
  ticker        = @tickers[trade.market.id]
  ticker[:low]  = get_market_low trade.market.id, trade
  ticker[:high] = get_market_high trade.market.id, trade
  ticker[:last] = trade.price
  Rails.logger.info ticker.inspect
  Rails.cache.write "peatio:#{trade.market.id}:ticker", ticker
end

def update_latest_trades(trade)
  trades = @trades[trade.market.id]
  trades.unshift(trade.for_global)
  trades.pop if trades.size > FRESH_TRADES

  Rails.cache.write "peatio:#{trade.market.id}:trades", trades
end

def initialize_market_data(market)
  trades = Trade.with_currency(market)

  @trades[market.id] = trades.order('id desc').limit(FRESH_TRADES).map(&:for_global)
  Rails.cache.write "peatio:#{market.id}:trades", @trades[market.id]

  low_trade = initialize_market_low(market.id)
  high_trade = initialize_market_high(market.id)
  @tickers[market.id] = {
    low:  low_trade.try(:price)   || ::Trade::ZERO,
    high: high_trade.try(:price)  || ::Trade::ZERO,
    last: trades.last.try(:price) || ::Trade::ZERO
  }
  Rails.cache.write "peatio:#{market.id}:ticker", @tickers[market.id]
end

private

def get_market_low(market, trade)
  low_key = "peatio:#{market}:h24:low"
  low = Rails.cache.read(low_key)

  if low.nil?
    trade = initialize_market_low(market)
    low = trade.price
  elsif trade.price < low
    low = trade.price
    write_h24_key low_key, low
  end

  low
end

def get_market_high(market, trade)
  high_key = "peatio:#{market}:h24:high"
  high = Rails.cache.read(high_key)

  if high.nil?
    trade = initialize_market_high(market)
    high = trade.price
  elsif trade.price > high
    high = trade.price
    write_h24_key high_key, high
  end

  high
end

def initialize_market_low(market)
```

```
      if low_trade = Trade.with_currency(market).h24.order('price asc').first
        ttl = low_trade.created_at.to_i + 24.hours - Time.now.to_i
        write_h24_key "peatio:#{market}:h24:low", low_trade.price, ttl
        low_trade
      end
    end

    def initialize_market_high(market)
      if high_trade = Trade.with_currency(market).h24.order('price desc').first
        ttl = high_trade.created_at.to_i + 24.hours - Time.now.to_i
        write_h24_key "peatio:#{market}:h24:high", high_trade.price, ttl
        high_trade
      end
    end

    def write_h24_key(key, value, ttl=24.hours)
      Rails.cache.write key, value, expires_in: ttl
    end

  end
end
```

188:F:\git\coin\exchange\peatio-master\app\models\worker\matching.rb

```
module Worker
  class Matching

    class DryrunError < StandardError
      attr :engine

      def initialize(engine)
        @engine = engine
      end
    end

    def initialize(options={})
      @options = options
      reload 'all'
    end

    def process(payload, metadata, delivery_info)
      payload.symbolize_keys!
  case payload[:action]
  when 'submit'
    submit build_order(payload[:order])
  when 'cancel'
    cancel build_order(payload[:order])
  when 'reload'
    reload payload[:market]
  else
    Rails.logger.fatal "Unknown action: #{payload[:action]}"
  end
end

def submit(order)
  engines[order.market.id].submit(order)
end

def cancel(order)
  engines[order.market.id].cancel(order)
end

def reload(market)
  if market == 'all'
    Market.all.each {|market| initialize_engine market }
    Rails.logger.info "All engines reloaded."
  else
    initialize_engine Market.find(market)
    Rails.logger.info "#{market} engine reloaded."
  end
rescue DryrunError => e
  # stop started engines
  engines.each {|id, engine| engine.shift_gears(:dryrun) unless engine == e.engine }

  Rails.logger.fatal "#{market} engine failed to start. Matched during dryrun:"
  e.engine.queue.each do |trade|
    Rails.logger.info trade[1].inspect
  end
end

def build_order(attrs)
  ::Matching::OrderBookManager.build_order attrs
end
```

```
def initialize_engine(market)
  create_engine market
  load_orders   market
  start_engine  market
end

def create_engine(market)
  engines[market.id] = ::Matching::Engine.new(market, @options)
end

def load_orders(market)
  ::Order.active.with_currency(market.id).order('id asc').each do |order|
    submit build_order(order.to_matching_attributes)
  end
end

def start_engine(market)
  engine = engines[market.id]
  if engine.mode == :dryrun
    if engine.queue.empty?
      engine.shift_gears :run
    else
      accept = ENV['ACCEPT_MINUTES'] ? ENV['ACCEPT_MINUTES'].to_i : 30
      order_ids = engine.queue
        .map {|args| [args[1][:ask_id], args[1][:bid_id]] }
        .flatten.uniq

      orders = Order.where('created_at < ?', accept.minutes.ago).where(id: order_ids)
      if orders.exists?
        # there're very old orders matched, need human intervention
        raise DryrunError, engine
      else
        # only buffered orders matched, just publish trades and continue
        engine.queue.each {|args| AMQPQueue.enqueue(*args) }
        engine.shift_gears :run
      end
    end
  else
    Rails.logger.info "#{market.id} engine already started. mode=#{engine.mode}"
  end
end
```

```
def engines
  @engines ||= {}
end

# dump limit orderbook
def on_usr1
  engines.each do |id, eng|
    dump_file = File.join('/', 'tmp', "limit_orderbook_#{id}")
    limit_orders = eng.limit_orders

    File.open(dump_file, 'w') do |f|
      f.puts "ASK"
      limit_orders[:ask].keys.reverse.each do |k|
        f.puts k.to_s('F')
        limit_orders[:ask][k].each {|o| f.puts "\t#{o.label}" }
      end
      f.puts "-"*40
      limit_orders[:bid].keys.reverse.each do |k|
        f.puts k.to_s('F')
        limit_orders[:bid][k].each {|o| f.puts "\t#{o.label}" }
      end
      f.puts "BID"
    end

    puts "#{id} limit orderbook dumped to #{dump_file}."
  end
end

# dump market orderbook
def on_usr2
  engines.each do |id, eng|
    dump_file = File.join('/', 'tmp', "market_orderbook_#{id}")
    market_orders = eng.market_orders

    File.open(dump_file, 'w') do |f|
      f.puts "ASK"
      market_orders[:ask].each {|o| f.puts "\t#{o.label}" }
      f.puts "-"*40
      market_orders[:bid].each {|o| f.puts "\t#{o.label}" }
      f.puts "BID"
    end
```

```
      puts "#{id} market orderbook dumped to #{dump_file}."
    end
  end

  end
end
```

189:F:\git\coin\exchange\peatio-master\app\models\worker\member_stats.rb

```
module Worker
  class MemberStats < Stats

    def key_for(period)
      "peatio:stats:member:#{period}"
    end

    def to_s
      self.class.name
    end

    def point_1(from)
      to = from + 1.minute
      signup_count = Member.where(created_at: from...to).count
      activate_count = Member.where(activated: true, created_at: from...to).count
      [from.to_i, signup_count, activate_count]
    end

    def point_n(from, period)
      arr = point_1_set from, period
      signup_count = arr.sum {|point| point[1] }
      activate_count = arr.sum(&:last)
      [from.to_i, signup_count, activate_count]
    end

  end
end
```

190:F:\git\coin\exchange\peatio-master\app\models\worker\order_processor.rb

```
module Worker
  class OrderProcessor

    def initialize
      @cancel_queue = []
  create_cancel_thread
end

def process(payload, metadata, delivery_info)
  case payload['action']
  when 'cancel'
    unless check_and_cancel(payload['order'])
      @cancel_queue << payload['order']
    end
  else
    raise ArgumentError, "Unrecogonized action: #{payload['action']}"
  end
rescue
  SystemMailer.order_processor_error(payload, $!.message, $!.backtrace.join("\n")).deliver
  raise $!
end

def check_and_cancel(attrs)
  retry_count = 5
  begin
    order = Order.find attrs['id']
    if order.volume == attrs['volume'].to_d # all trades has been processed
      Ordering.new(order).cancel!
      puts "Order##{order.id} cancelled."
      true
    end
  rescue ActiveRecord::StatementInvalid
    # in case: Mysql2::Error: Lock wait timeout exceeded
    if retry_count > 0
      sleep 0.5
      retry_count -= 1
      puts $!
      puts "Retry order.cancel! (#{retry_count} retry left) .."
      retry
    else
      puts "Failed to cancel order##{order.id}"
      raise $!
    end
  end
rescue Ordering::CancelOrderError
  puts "Skipped: #{$!}"
  true
```

```
    end

    def process_cancel_jobs
      queue = @cancel_queue
      @cancel_queue = []

      queue.each do |attrs|
        unless check_and_cancel(attrs)
          @cancel_queue << attrs
        end
      end

      Rails.logger.info "Cancel queue size: #{@cancel_queue.size}"
    rescue
      Rails.logger.debug "Failed to process cancel job: #{$!}"
      Rails.logger.debug $!.backtrace.join("\n")
    end

    def create_cancel_thread
      Thread.new do
        loop do
          sleep 5
          process_cancel_jobs
        end
      end
    end

  end
end
```

191:F:\git\coin\exchange\peatio-master\app\models\worker\pusher_market.rb

```
module Worker
  class PusherMarket

    def process(payload, metadata, delivery_info)
      trade = Trade.new payload
      trade.trigger_notify
      Global[trade.market].trigger_trades [trade.for_global]
    end

  end
end
```

192:F:\git\coin\exchange\peatio-master\app\models\worker\pusher_member.rb
```
module Worker
  class PusherMember

    def process(payload, metadata, delivery_info)
      member = Member.find payload['member_id']
      event  = payload['event']
      data   = JSON.parse payload['data']
      member.notify event, data
    end

  end
end
```

193:F:\git\coin\exchange\peatio-master\app\models\worker\slave_book.rb
```
module Worker
  class SlaveBook

    def initialize(run_cache_thread=true)
      @managers = {}

      if run_cache_thread
        cache_thread = Thread.new do
          loop do
            sleep 3
            cache_book
          end
        end
      end
    end

    def process(payload, metadata, delivery_info)
      @payload = Hashie::Mash.new payload

      case @payload.action
      when 'new'
        @managers.delete(@payload.market)
        initialize_orderbook_manager(@payload.market)
      when 'add'
        book.add order
      when 'update'
        book.find(order).volume = order.volume # only volume would change
      when 'remove'
        book.remove order
      else
        raise ArgumentError, "Unknown action: #{@payload.action}"
      end
    rescue
      Rails.logger.error "Failed to process payload: #{$!}"
      Rails.logger.error $!.backtrace.join("\n")
    end

    def cache_book
      @managers.keys.each do |id|
        market = Market.find id
        Rails.cache.write "peatio:#{market}:depth:asks", get_depth(market, :ask)
        Rails.cache.write "peatio:#{market}:depth:bids", get_depth(market, :bid)
        Rails.logger.debug "SlaveBook (#{market}) updated"
      end
    rescue
      Rails.logger.error "Failed to cache book: #{$!}"
      Rails.logger.error $!.backtrace.join("\n")
    end

    def order
      ::Matching::OrderBookManager.build_order @payload.order.to_h
    end

    def book
      manager.get_books(@payload.order.type.to_sym).first
    end

    def manager
      market = @payload.order.market
      @managers[market] || initialize_orderbook_manager(market)
    end

    def initialize_orderbook_manager(market)
      @managers[market] = ::Matching::OrderBookManager.new(market, broadcast: false)
    end

    def get_depth(market, side)
      depth = Hash.new {|h, k| h[k] = 0 }
```

```
      price_group_fixed = market[:price_group_fixed]
      mode  = side == :ask ? BigDecimal::ROUND_UP : BigDecimal::ROUND_DOWN
      @managers[market.id].send("#{side}_orders").limit_orders.each do |price, orders|
        price = price.round(price_group_fixed, mode) if price_group_fixed
        depth[price] += orders.map(&:volume).sum
      end

      depth = depth.to_a
      depth.reverse! if side == :bid
      depth
    end

  end
end
```

194:F:\git\coin\exchange\peatio-master\app\models\worker\sms_notification.rb

```
module Worker
  class SmsNotification

    def process(payload, metadata, delivery_info)
      payload.symbolize_keys!

      raise "TWILIO_NUMBER not set" if ENV['TWILIO_NUMBER'].blank?

      twilio_client.account.sms.messages.create(
        from: ENV["TWILIO_NUMBER"],
        to:   Phonelib.parse(payload[:phone]).international,
        body: payload[:message]
      )
    end

    def twilio_client
      Twilio::REST::Client.new ENV["TWILIO_SID"], ENV["TWILIO_TOKEN"], ssl_verify_peer: false
    end

  end
end
```

195:F:\git\coin\exchange\peatio-master\app\models\worker\stats.rb

```
module Worker
  class Stats
  def initialize
    @redis  = Redis.new url: ENV["REDIS_URL"], db: 1
  end

  def run
    [1, 60, 1440, 10080].each do |period|
      collect period
    end
    Rails.logger.info "#{self.to_s} collected."
  end

  def to_s
    self.class.name
  end

  def key_for(period)
    raise "abstract method"
  end

  def point_1(from)
    raise "abstract method"
  end

  def point_n(from)
    raise "abstract method"
  end

  def collect(period)
    key = key_for period
    loop do
      ts = next_point key, period
      break if (ts + period.minutes) > (Time.now + 30.second) # 30 seconds should be enough to
allow data propagate from master to slave

      point = period == 1 ? point_1(ts) : point_n(ts, period)
      @redis.rpush key, point.to_json
    end
  end

  def next_point(key, period=1)
    last = @redis.lindex key, -1
    if last
```

```
        ts = Time.at JSON.parse(last)[0]
        ts += period.minutes
      else
        ts = 7.days.ago(Time.now.beginning_of_day)
      end
    end

    def point_1_set(from, period)
      key1 = key_for 1
      ts = JSON.parse(@redis.lindex(key1, 0)).first

      offset = [(from.to_i - ts)/60, 0].max
      to = offset + period - 1

      to < offset ? [] : @redis.lrange(key1, offset, to).map {|str| JSON.parse(str) }
    end

    def get_point(ts, period)
      key = key_for period
      first_ts = JSON.parse(@redis.lindex(key, 0)).first

      offset = (ts-first_ts) / (60*period)
      return if offset < 0

      JSON.parse @redis.lindex(key, offset)
    end

  end
end
```

196:F:\git\coin\exchange\peatio-master\app\models\worker\top_stats.rb

```
module Worker
  class TopStats < Stats

    def initialize(market)
      super()
      @market = market
    end

    def run
      [60, 1440, 10080].each do |period|
        collect period
      end
      Rails.logger.info "#{self.to_s} collected."
    end

    def to_s
      "#{self.class.name} (#{@market.id})"
    end

    def key_for(period)
      "peatio:stats:top:#{@market.id}:#{period}"
    end

    def point_n(from, period)
      if (from+period.minutes) < (Time.now-period.minutes)
        [from.to_i, [], []]
      else
        to = from + period.minutes
        trades = Trade.with_currency(@market.id).where(created_at:
from..to).pluck(:ask_member_id, :bid_member_id, :volume)

        user_trades = Hash.new {|h, k| h[k] = 0 }
        user_volume = Hash.new {|h, k| h[k] = 0 }
        trades.each do |t|
          if t[0] == t[1] # ask_member_id == bid_member_id
            user_trades[t[0]] += 1
            user_volume[t[0]] += t[2]
          else
            user_trades[t[0]] += 1
            user_trades[t[1]] += 1
            user_volume[t[0]] += t[2]
            user_volume[t[1]] += t[2]
          end
        end

        top_trades_users = user_trades.to_a.sort_by {|ut| -ut.last }[0, 50]
        top_volume_users = user_volume.to_a.sort_by {|uv| -uv.last }[0, 50]
        [from.to_i, top_trades_users, top_volume_users]
      end
    end

  end
end
```

```
197:F:\git\coin\exchange\peatio-master\app\models\worker\trade_executor.rb
module Worker
  class TradeExecutor

    def process(payload, metadata, delivery_info)
      payload.symbolize_keys!
      ::Matching::Executor.new(payload).execute!
    rescue
      SystemMailer.trade_execute_error(payload, $!.message, $!.backtrace.join("\n")).deliver
      raise $!
    end

  end
end

198:F:\git\coin\exchange\peatio-master\app\models\worker\trade_stats.rb
module Worker
  class TradeStats < Stats

    def initialize(market)
      super()
      @market = market
    end

    def to_s
      "#{self.class.name} (#{@market.id})"
    end

    def key_for(period)
      "peatio:stats:trades:#{@market.id}:#{period}"
    end

    def point_1(from)
      to = from + 1.minute
      trades = Trade.with_currency(@market.id).where(created_at:
from...to).pluck(:ask_member_id, :bid_member_id)
      trade_users = trades.flatten.uniq
      [from.to_i, trades.size, trade_users.size]
    end

    def point_n(from, period)
```

```
      arr = point_1_set from, period
      trades_count = arr.sum {|point| point[1]}
      trade_users_count = arr.sum(&:last)
      [from.to_i, trades_count, trade_users_count]
    end

  end
end

199:F:\git\coin\exchange\peatio-master\app\models\worker\wallet_stats.rb
module Worker
  class WalletStats < Stats

    def initialize(currency)
      super()
      @currency = currency
    end

    def run
      [60, 1440, 10080].each do |period|
        collect period
      end
      Rails.logger.info "#{self.to_s} collected."
    end

    def to_s
      "#{self.class.name} (#{@currency.code})"
    end

    def key_for(period)
      "peatio:stats:wallet:#{@currency.code}:#{period}"
    end

    def point_n(from, period)
      if (from+period.minutes) < (Time.now-period.minutes)
        [from.to_i, 0, 0, 0]
      else
        balance = Account.balance_sum(@currency.code)
        locked  = Account.locked_sum(@currency.code)
        [from.to_i, balance.to_f, locked.to_f, (balance+locked).to_f]
      end
    end
  end
end

200:F:\git\coin\exchange\peatio-master\app\models\worker\withdraw_coin.rb
module Worker
  class WithdrawCoin

    def process(payload, metadata, delivery_info)
      payload.symbolize_keys!

      Withdraw.transaction do
        withdraw = Withdraw.lock.find payload[:id]

        return unless withdraw.processing?

        withdraw.whodunnit('Worker::WithdrawCoin') do
          withdraw.call_rpc
          withdraw.save!
        end
      end

      Withdraw.transaction do
        withdraw = Withdraw.lock.find payload[:id]

        return unless withdraw.almost_done?

        balance = CoinRPC[withdraw.currency].getbalance.to_d
        raise Account::BalanceError, 'Insufficient coins' if balance < withdraw.sum

        fee = [withdraw.fee.to_f || withdraw.channel.try(:fee) || 0.0005, 0.1].min

        CoinRPC[withdraw.currency].settxfee fee
        txid = CoinRPC[withdraw.currency].sendtoaddress withdraw.fund_uid, withdraw.amount.to_f

        withdraw.whodunnit('Worker::WithdrawCoin') do
          withdraw.update_column :txid, txid

          # withdraw.succeed! will start another transaction, cause
          # Account after_commit callbacks not to fire
          withdraw.succeed
          withdraw.save!
        end
      end
    end

  end
end
```

201:F:\git\coin\exchange\peatio-master\app\observers\audit_observer.rb

```
class AuditObserver < ActiveRecord::Observer
  def current_user
    Member.current
  end
end
```

202:F:\git\coin\exchange\peatio-master\app\observers\transfer_observer.rb

```
class TransferObserver < AuditObserver
  observe :deposit, :withdraw

  def after_update(record)
    if record.aasm_state_changed?
      Audit::TransferAuditLog.audit!(record, current_user)
    end
  end

end
```

203:F:\git\coin\exchange\peatio-master\app\services\coin_rpc.rb

```
require 'net/http'
require 'uri'
require 'json'

class CoinRPC

  class JSONRPCError < RuntimeError; end
  class ConnectionRefusedError < StandardError; end

  def initialize(uri)
    @uri = URI.parse(uri)
  end

  def self.[](currency)
    c = Currency.find_by_code(currency.to_s)
    if c && c.rpc
      name = c[:handler] || 'BTC'
      "::CoinRPC::#{name}".constantize.new(c.rpc)
    end
  end

  def method_missing(name, *args)
    handle name, *args
  end

  def handle
    raise "Not implemented"
  end

  class BTC < self
    def handle(name, *args)
      post_body = { 'method' => name, 'params' => args, 'id' => 'jsonrpc' }.to_json
      resp = JSON.parse( http_post_request(post_body) )
      raise JSONRPCError, resp['error'] if resp['error']
      result = resp['result']
      result.symbolize_keys! if result.is_a? Hash
      result
    end

    def http_post_request(post_body)
      http    = Net::HTTP.new(@uri.host, @uri.port)
      request = Net::HTTP::Post.new(@uri.request_uri)
      request.basic_auth @uri.user, @uri.password
      request.content_type = 'application/json'
      request.body = post_body
      http.request(request).body
    rescue Errno::ECONNREFUSED => e
      raise ConnectionRefusedError
    end

    def safe_getbalance
      begin
        getbalance
      rescue
        'N/A'
      end
    end
  end

end

204:F:\git\coin\exchange\peatio-master\app\services\ordering.rb
class Ordering

  class CancelOrderError < StandardError; end

  def initialize(order_or_orders)
    @orders = Array(order_or_orders)
  end

  def submit
    ActiveRecord::Base.transaction do
      @orders.each {|order| do_submit order }
    end

    @orders.each do |order|
      AMQPQueue.enqueue(:matching, action: 'submit', order: order.to_matching_attributes)
    end

    true
  end

  def cancel
    @orders.each {|order| do_cancel order }
  end

  def cancel!
    ActiveRecord::Base.transaction do
      @orders.each {|order| do_cancel! order }
    end
  end

  private

  def do_submit(order)
    order.fix_number_precision # number must be fixed before computing locked
    order.locked = order.origin_locked = order.compute_locked
    order.save!
    account = order.hold_account
    account.lock_funds(order.locked, reason: Account::ORDER_SUBMIT, ref: order)
  end

  def do_cancel(order)
    AMQPQueue.enqueue(:matching, action: 'cancel', order: order.to_matching_attributes)
  end

  def do_cancel!(order)
    account = order.hold_account
    order   = Order.find(order.id).lock!

    if order.state == Order::WAIT
      order.state = Order::CANCEL
      account.unlock_funds(order.locked, reason: Account::ORDER_CANCEL, ref: order)
      order.save!
    else
      raise CancelOrderError, "Only active order can be cancelled. id: #{order.id}, state:
#{order.state}"
    end
  end

end

205:F:\git\coin\exchange\peatio-master\app\uploaders\file_uploader.rb
class FileUploader < CarrierWave::Uploader::Base
  def store_dir
    "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}"
  end

  def filename
    "#{secure_token}.#{file.extension}" if original_filename.present?
  end

  def extension_white_list
    %w(jpg jpeg gif png pdf)
  end

  protected

  def secure_token
    var = :"@#{mounted_as}_secure_token"
    model.instance_variable_get(var) or model.instance_variable_set(var, SecureRandom.uuid)
  end
end

206:F:\git\coin\exchange\peatio-master\app\validators\currency_validator.rb
class CurrencyValidator < ActiveModel::EachValidator
  def validate_each(record, attribute, value)
    currency = eval Figaro.env.currency
    key = "#{record.bid}_#{record.ask}"

    precision = currency[key]['precision'][attribute.to_s]

    unless BigDecimal.new(value) % BigDecimal.new(precision.to_s) == 0
      record.errors[attribute] << (options[:message] ||
I18n.t('activemodel.errors.messages.orders.precision', p: precision))
    end

    range = currency[key]['range'][attribute.to_s]
    range = Range.new(*range)

    unless range.cover? value.to_f
      record.errors[attribute] << (options[:message] ||
I18n.t('activemodel.errors.messages.orders.price', l: range.min, h: range.max))
    end

    range = currency[key]['range']['sum']
    range = Range.new(*range)
    sum = BigDecimal.new(record.price) * BigDecimal.new(record.volume)

    unless range.cover? sum.to_f
      record.errors[attribute] << (options[:message] ||
I18n.t('activemodel.errors.messages.orders.sum', l: range.min, h: range.max))
    end
  end
end


207:F:\git\coin\exchange\peatio-master\app\validators\email_validator.rb
class EmailValidator < ActiveModel::EachValidator
  def validate_each(record, attribute, value)
    return if value.nil?
    unless value =~ /\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\z/i
```

```
      record.errors[attribute] << (options[:message] ||
I18n.t("activerecord.errors.messages.invalid_email"))
    end
  end
end

208:F:\git\coin\exchange\peatio-master\app\validators\strength_validator.rb
class StrengthValidator < ActiveModel::EachValidator
  def validate_each(record, attribute, value)
    min = options[:min]
    min ||= 6
    unless value =~ /(?=^.{#{min},}$)((?=.*\d)|(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*\z/
      record.errors[attribute] << (options[:message] ||
I18n.t("activemodel.errors.messages.strength"))
    end
  end
end


209:F:\git\coin\exchange\peatio-master\app\validators\ticket_validator.rb
class TicketValidator < ActiveModel::Validator
  def validate(record)
    if record.title.blank? && record.content.blank?
      record.errors[:title] << I18n.t('private.tickets.title_content_both_blank')
    end
  end
end

210:F:\git\coin\exchange\peatio-master\app\validators\withdraw_blacklist_validator.rb
class WithdrawBlacklistValidator < ActiveModel::Validator

  def validate(record)
    if record.channel.blacklist && record.channel.blacklist.include?(record.fund_uid)
      record.errors[:fund_uid] << I18n.t('withdraws.invalid_address')
    end
  end

end

211:F:\git\coin\exchange\peatio-master\bin\concurrent_create_order_benchmark.rb
#!/usr/bin/env ruby
```

```
ENV['RAILS_ENV'] = 'test'
require_relative '../config/environment'
require_relative 'matching_benchmark'

class ConcurrentCreateOrderBenchmark < MatchingBenchmark

  def initialize(label, num, round, process_num)
    super(label, num, round)
    @process_num = process_num
  end

  def collect_time
    time = Dir[Rails.root.join('tmp', 'concurrent_create_order_*')].map do |f|
      File.open(f, 'r') {|ff| ff.read.to_f }
    end.max
    puts "elapsed: #{time}"
    Benchmark::Tms.new(0, 0, 0, 0, time)
  end

  def create_orders
    members = Member.all
    members.in_groups(@process_num, false).each_with_index do |users, i|
      unless Process.fork
        ActiveRecord::Base.connection.reconnect!
        puts "Process #{i+1} started."

        t1 = Time.now
        users.each {|m| SweatFactory.make_ask_order(m, 10, 4000) }
        elapsed = Time.now - t1
        File.open(Rails.root.join('tmp', "concurrent_create_order_#{i+1}"), 'w') {|f| f.write(elapsed.to_f)
}

        puts "Process #{i+1} finished, stop."
        exit 0
      end
    end

    pid_and_status = Process.waitall
    ActiveRecord::Base.connection.reconnect!

    collect_time
  end
  def run_prepare_orders
    (1..@round).map do |i|
      puts "\n>> Round #{i}"
      Benchmark.benchmark(Benchmark::CAPTION, 20, Benchmark::FORMAT) do |x|
        @times[:create_members] << x.report("create members") { create_members }
        @times[:lock_funds]     << x.report("lock funds") { lock_funds }
        nil
      end
    end
  end

  def run
    run_prepare_orders

    Benchmark.benchmark(Benchmark::CAPTION, 20, Benchmark::FORMAT) do |x|
      @times[:create_orders] = [ create_orders ]
      puts "#{Order.count} orders created by #{@process_num} processes."
    end

    save
  end

end


if $0 == __FILE__
  raise "Must run in test environment!" unless Rails.env.test?

  process_num = ARGV[0] ? ARGV[0].to_i : 8
  num = ARGV[1] ? ARGV[1].to_i : 250
  round = ARGV[2] ? ARGV[2].to_i : 4
  label = ARGV[3] || Time.now.to_i

  puts "\n>> Setup environment"
  system("rake db:reset")
  Dir[Rails.root.join('tmp', 'concurrent_create_order_*')].each {|f| FileUtils.rm(f) }

  ConcurrentCreateOrderBenchmark.new(label, num, round, process_num).run
end
```

212:F:\git\coin\exchange\peatio-master\config\application.rb

```
require File.expand_path('../boot', __FILE__)

# Pick the frameworks you want:
require "active_record/railtie"
require "action_controller/railtie"
require "action_mailer/railtie"
require "sprockets/railtie"
# require "rails/test_unit/railtie"

# Require the gems listed in Gemfile, including any gems
# you've limited to :test, :development, or :production.
Bundler.require(:default, Rails.env)

module Peatio
  class Application < Rails::Application
    # Settings in config/environments/* take precedence over those specified here.
    # Application configuration should go into files in config/initializers
    # -- all .rb files in that directory are automatically loaded.

    # Set Time.zone default to the specified zone and make Active Record auto-convert to this
zone.
    # Run "rake -D time" for a list of tasks for finding time zone names. Default is UTC.
    # config.time_zone = 'Central Time (US & Canada)'

    config.i18n.enforce_available_locales = false

    # The default locale is :en and all translations from config/locales/*.rb,yml are auto loaded.
    config.i18n.load_path += Dir[Rails.root.join('config', 'locales', 'custom', '*.{yml}')]
    config.i18n.available_locales = ['en', 'zh-CN', 'ko']

    config.autoload_paths += %W(#{config.root}/lib #{config.root}/lib/extras)

    #config.assets.precompile += ['bootstrap-datetimepicker.css']
    config.assets.initialize_on_precompile = true

    # Precompile all available locales
    Dir.glob("#{config.root}/app/assets/javascripts/locales/*.js.erb").each do |file|
      config.assets.precompile << "locales/#{file.match(/([a-z\-A-Z]+\.js)\.erb$/)[1]}"
    end

    config.generators do |g|
      g.orm             :active_record
      g.template_engine :erb
      g.stylesheets     false
    end

    # Observer configuration
    config.active_record.observers = :transfer_observer
  end
end
```

213:F:\git\coin\exchange\peatio-master\config\boot.rb

```
require 'rubygems'

# Set up gems listed in the Gemfile.
ENV['BUNDLE_GEMFILE'] ||= File.expand_path('../../Gemfile', __FILE__)

require 'bundler/setup' if File.exists?(ENV['BUNDLE_GEMFILE'])
```

214:F:\git\coin\exchange\peatio-master\config\deploy.rb

```
require 'mina/bundler'
require 'mina/rails'
require 'mina/git'
require 'mina/rbenv'
require 'mina/slack/tasks'

set :repository, 'https://github.com/peatio/peatio.git'
set :user, 'deploy'
set :deploy_to, '/home/deploy/peatio'
set :branch, 'master'
set :domain, 'demo.peatio.com'

set :shared_paths, [
  'config/database.yml',
  'config/application.yml',
  'config/currencies.yml',
  'config/markets.yml',
  'config/amqp.yml',
  'config/banks.yml',
  'config/deposit_channels.yml',
  'config/withdraw_channels.yml',
  'public/uploads',
  'tmp',
  'log'
]

task :environment do
  invoke :'rbenv:load'
end

task :setup => :environment do
  queue! %[mkdir -p "#{deploy_to}/shared/log"]
  queue! %[chmod g+rx,u+rwx "#{deploy_to}/shared/log"]

  queue! %[mkdir -p "#{deploy_to}/shared/config"]
  queue! %[chmod g+rx,u+rwx "#{deploy_to}/shared/config"]

  queue! %[mkdir -p "#{deploy_to}/shared/tmp"]
  queue! %[chmod g+rx,u+rwx "#{deploy_to}/shared/tmp"]

  queue! %[mkdir -p "#{deploy_to}/shared/public/uploads"]
  queue! %[chmod g+rx,u+rwx "#{deploy_to}/shared/public/uploads"]

  queue! %[touch "#{deploy_to}/shared/config/database.yml"]
  queue! %[touch "#{deploy_to}/shared/config/currencies.yml"]
  queue! %[touch "#{deploy_to}/shared/config/application.yml"]
  queue! %[touch "#{deploy_to}/shared/config/markets.yml"]
  queue! %[touch "#{deploy_to}/shared/config/amqp.yml"]
  queue! %[touch "#{deploy_to}/shared/config/banks.yml"]
  queue! %[touch "#{deploy_to}/shared/config/deposit_channels.yml"]
  queue! %[touch "#{deploy_to}/shared/config/withdraw_channels.yml"]
end

desc "Deploys the current version to the server."
task deploy: :environment do
  deploy do
    invoke :'git:clone'
    invoke :'deploy:link_shared_paths'
    invoke :'bundle:install'
    invoke :'rails:db_migrate'
    invoke :'rails:touch_client_i18n_assets'
    invoke :'rails:assets_precompile'

    to :launch do
      invoke :'passenger:restart'
    end
  end
end

namespace :passenger do
  desc "Restart Passenger"
  task :restart do
    queue %{
      echo "-----> Restarting passenger"
      cd #{deploy_to}/current
      #{echo_cmd %[mkdir -p tmp]}
      #{echo_cmd %[touch tmp/restart.txt]}
    }
  end
end

namespace :rails do
  task :touch_client_i18n_assets do
    queue %[
      echo "-----> Touching clint i18n assets"
      #{echo_cmd %[RAILS_ENV=production bundle exec rake deploy:touch_client_i18n_assets]}
    ]
  end
end

namespace :daemons do
  desc "Start Daemons"
  task start: :environment do
    queue %{
      cd #{deploy_to}/current
      RAILS_ENV=production bundle exec ./bin/rake daemons:start
      echo Daemons START DONE!!!
    }
  end

  desc "Stop Daemons"
  task stop: :environment do
    queue %{
      cd #{deploy_to}/current
      RAILS_ENV=production bundle exec ./bin/rake daemons:stop
      echo Daemons STOP DONE!!!
    }
  end
  desc "Query Daemons"
  task status: :environment do
    queue %{
      cd #{deploy_to}/current
      RAILS_ENV=production bundle exec ./bin/rake daemons:status
    }
  end
end

desc "Generate liability proof"
task 'solvency:liability_proof' do
  queue "cd #{deploy_to}/current && RAILS_ENV=production bundle exec rake
solvency:liability_proof"
end
```

215:F:\git\coin\exchange\peatio-master\config\environment.rb

```
# Load the rails application
require File.expand_path('../application', __FILE__)

# Initialize the rails application
Peatio::Application.initialize!
```

216:F:\git\coin\exchange\peatio-master\config\environments\development.rb

```
Peatio::Application.configure do
  # Settings specified here will take precedence over those in config/application.rb

  # In the development environment your application's code is reloaded on
  # every request. This slows down response time but is perfect for development
  # since you don't have to restart the web server when you make code changes.
  config.cache_classes = false

  # Do not eager load code on boot.
  config.eager_load = false

  # Show full error reports and disable caching.
  config.consider_all_requests_local       = true
  config.action_controller.perform_caching = true

  # Use a different cache store in production.
  # config.cache_store = :file_store, "tmp"
  config.cache_store = :redis_store, ENV['REDIS_URL']
  config.session_store :redis_store, :key => '_peatio_session', :expire_after =>
ENV['SESSION_EXPIRE'].to_i.minutes

  # Don't care if the mailer can't send.
  config.action_mailer.raise_delivery_errors = false

  config.action_mailer.delivery_method = :file
  config.action_mailer.file_settings = { location: 'tmp/mails' }

  config.action_mailer.default_url_options = { :host => ENV["URL_HOST"] }

  # Print deprecation notices to the Rails logger.
  config.active_support.deprecation = :log

  # Raise an error on page load if there are pending migrations
  config.active_record.migration_error = :page_load

  # Debug mode disables concatenation and preprocessing of assets.
  # This option may cause significant delays in view rendering with a large
  # number of complex assets.
  config.assets.debug = true

  config.active_record.default_timezone = :local

  require 'middleware/i18n_js'
  require 'middleware/security'
  config.middleware.insert_before ActionDispatch::Static, Middleware::I18nJs
  config.middleware.insert_before Rack::Runtime, Middleware::Security
end
```

217:F:\git\coin\exchange\peatio-master\config\environments\production.rb

```
Peatio::Application.configure do
  # Settings specified here will take precedence over those in config/application.rb

  # Code is not reloaded between requests.
  config.cache_classes = true

  # Eager load code on boot. This eager loads most of Rails and
  # your application in memory, allowing both thread web servers
  # and those relying on copy on write to perform better.
  # Rake tasks automatically ignore this option for performance.
  config.eager_load = true

  # Full error reports are disabled and caching is turned on.
  config.consider_all_requests_local       = false
  config.action_controller.perform_caching = true

  # Enable Rack::Cache to put a simple HTTP cache in front of your application
  # Add `rack-cache` to your Gemfile before enabling this.
  # For large-scale production use, consider using a caching reverse proxy like nginx, varnish or
squid.
  # config.action_dispatch.rack_cache = true

  # Disable Rails's static asset server (Apache or nginx will already do this).
  config.serve_static_assets = false

  # Compress JavaScripts and CSS.
  config.assets.js_compressor = Uglifier.new(:mangle => false)
  # config.assets.css_compressor = :sass

  # Do not fallback to assets pipeline if a precompiled asset is missed.
  config.assets.compile = false

  # Generate digests for assets URLs.
  config.assets.digest = true

  # Version of your assets, change this if you want to expire all your assets.
  config.assets.version = '1.0'

  # Specifies the header that your server uses for sending files.
  # config.action_dispatch.x_sendfile_header = "X-Sendfile" # for apache
  # config.action_dispatch.x_sendfile_header = 'X-Accel-Redirect' # for nginx

  # Force all access to the app over SSL, use Strict-Transport-Security, and use secure cookies.
  config.force_ssl = false

  # Set to :debug to see everything in the log.
  config.log_level = :info

  # Prepend all log lines with the following tags.
  # config.log_tags = [ :subdomain, :uuid ]

  # Use a different logger for distributed setups.
  # config.logger = ActiveSupport::TaggedLogging.new(SyslogLogger.new)

  # Use a different cache store in production.
  # config.cache_store = :memory_store
  config.cache_store = :redis_store, ENV['REDIS_URL']

  config.session_store :redis_store, :key => '_peatio_session', :expire_after =>
ENV['SESSION_EXPIRE'].to_i.minutes

  # Enable serving of images, stylesheets, and JavaScripts from an asset server.
  # config.action_controller.asset_host = "http://assets.example.com"

  # Precompile additional assets.
  # application.js, application.css, and all non-JS/CSS in app/assets folder are already added.
  config.assets.precompile += %w( funds.js market.js market.css admin.js admin.css html5.js
api_v2.css api_v2.js .svg .eot .woff .ttf )

  # Ignore bad email addresses and do not raise email delivery errors.
  # Set this to true and configure the email server for immediate delivery to raise delivery errors.
  # config.action_mailer.raise_delivery_errors = false
  config.action_mailer.default_url_options = { host: ENV["URL_HOST"], protocol:
ENV['URL_SCHEMA'] }

  config.action_mailer.delivery_method = :smtp
  config.action_mailer.smtp_settings = {
    port:           ENV["SMTP_PORT"],
    domain:         ENV["SMTP_DOMAIN"],
    address:        ENV["SMTP_ADDRESS"],
    user_name:      ENV["SMTP_USERNAME"],
    password:       ENV["SMTP_PASSWORD"],
    authentication: ENV["SMTP_AUTHENTICATION"]
  }

  # Enable locale fallbacks for I18n (makes lookups for any locale fall back to
  # the I18n.default_locale when a translation can not be found).
  config.i18n.fallbacks = true

  # Send deprecation notices to registered listeners.
  config.active_support.deprecation = :notify

  # Disable automatic flushing of the log to improve performance.
  # config.autoflush_log = false
  # Use default logging formatter so that PID and timestamp are not suppressed.
  config.log_formatter = ::Logger::Formatter.new
  config.active_record.default_timezone = :local

  config.middleware.insert_before Rack::Runtime, Middleware::Security
end

218:F:\git\coin\exchange\peatio-master\config\environments\test.rb
Peatio::Application.configure do
  # Settings specified here will take precedence over those in config/application.rb

  # The test environment is used exclusively to run your application's
  # test suite. You never need to work with it otherwise. Remember that
  # your test database is "scratch space" for the test suite and is wiped
  # and recreated between test runs. Don't rely on the data there!
  config.cache_classes = true

  # Do not eager load code on boot. This avoids loading your whole application
  # just for the purpose of running a single test. If you are using a tool that
  # preloads Rails for running tests, you may have to set it to true.
  config.eager_load = false

  # Configure static asset server for tests with Cache-Control for performance.
  config.serve_static_assets  = true
  config.static_cache_control = "public, max-age=3600"

  # Show full error reports and disable caching.
  config.consider_all_requests_local       = true
  config.action_controller.perform_caching = false

  # Raise exceptions instead of rendering exception templates.
  config.action_dispatch.show_exceptions = false

  # Disable request forgery protection in test environment.
  config.action_controller.allow_forgery_protection = false

  # Tell Action Mailer not to deliver emails to the real world.
  # The :test delivery method accumulates sent emails in the
  # ActionMailer::Base.deliveries array.
  config.action_mailer.delivery_method = :test
  config.action_mailer.default_url_options = { :host => ENV["URL_HOST"] }
  # Print deprecation notices to the stderr.
  config.active_support.deprecation = :stderr

  config.session_store :cookie_store, :key => '_peatio_session', :expire_after =>
ENV['SESSION_EXPIRE'].to_i.minutes
end

219:F:\git\coin\exchange\peatio-master\config\initializers\action_controller.rb
# ActionController::Base are used by both Peatio controllers and
# Doorkeeper controllers.
class ActionController::Base

  before_action :set_language

  private

  def set_language
    cookies[:lang] = params[:lang] unless params[:lang].blank?
    locale = cookies[:lang] ||
http_accept_language.compatible_language_from(I18n.available_locales)
    I18n.locale = locale if locale && I18n.available_locales.include?(locale.to_sym)
  end

  def set_redirect_to
    if request.get?
      uri = URI(request.url)
      cookies[:redirect_to] = "#{uri.path}?#{uri.query}"
    end
  end

end

220:F:\git\coin\exchange\peatio-master\config\initializers\activerecord.rb
module ActiveModel
  module Translation
    alias :han :human_attribute_name
  end
end

ActiveRecord::Base.extend ActiveHash::Associations::ActiveRecordExtensions
```

221:F:\git\coin\exchange\peatio-master\config\initializers\backtrace_silencers.rb

```ruby
# Be sure to restart your server when you modify this file.

# You can add backtrace silencers for libraries that you're using but don't wish to see in your backtraces.
# Rails.backtrace_cleaner.add_silencer { |line| line =~ /my_noisy_library/ }

# You can also remove all the silencers if you're trying to debug a problem that might stem from framework code.
# Rails.backtrace_cleaner.remove_silencers!
```

222:F:\git\coin\exchange\peatio-master\config\initializers\carrierwave.rb

```ruby
CarrierWave.configure do |config|
  config.storage = :file
  config.cache_dir = "#{Rails.root}/tmp/uploads"
end
```

223:F:\git\coin\exchange\peatio-master\config\initializers\check_env.rb

```ruby
environments = %w(
  PUSHER_APP
  PUSHER_KEY
  PUSHER_SECRET
)

environments.select! do |key|
  ENV[key] =~ /^YOUR/
end

unless environments.empty?
  puts "==================== WARNING ===================="
  puts "  please check below config in config/application.yml"
  puts ""
  environments.each do |key| puts "  #{key}" end
  puts "=================================================="
  raise "config missing"
end
```

224:F:\git\coin\exchange\peatio-master\config\initializers\datagrid\filters\date_time_filter.rb

```ruby
class Datagrid::Filters::DateTimeFilter < Datagrid::Filters::BaseFilter
  def parse(value)
    if value.respond_to?(:utc)
      value = value.utc
    end

    if value.is_a?(String)
      return value
    else
      return value.to_s(:db)
    end
  end
end
```

225:F:\git\coin\exchange\peatio-master\config\initializers\date_time_format.rb

```
Date::DATE_FORMATS[:short] = '%m-%d'
Time::DATE_FORMATS[:default] = "%Y-%m-%d %H:%M:%S"
```

226:F:\git\coin\exchange\peatio-master\config\initializers\dirty_ext.rb

```
module ActiveModel
  module Dirty
    def changes_attributes
      HashWithIndifferentAccess[changed.map { |attr| [attr, __send__(attr)] }]
    end

    def changes_attributes_as_json
      ca, json = changes_attributes, self.as_json
      json.each do |key, value|
        ca[key.to_s] = value if ca.key?(key)
      end
      ca
    end
  end
end
```

227:F:\git\coin\exchange\peatio-master\config\initializers\doorkeeper.rb

```
Doorkeeper.configure do
  # Change the ORM that doorkeeper will use.
  # Currently supported options are :active_record, :mongoid2, :mongoid3, :mongo_mapper
  orm :active_record

  # This block will be called to check whether the resource owner is authenticated or not.
  resource_owner_authenticator do
```

```
    user = Member.enabled.where(id: session[:member_id]).first
    if user && user.activated?
      Member.current = user
    else
      set_redirect_to
      redirect_to signin_path
    end
  end

  # If you want to restrict access to the web interface for adding oauth authorized applications, you
need to declare the block below.
  admin_authenticator do
    user = Member.enabled.where(id: session[:member_id]).first
    if user && user.activated? && user.admin?
      Member.current = user
    else
      set_redirect_to
      redirect_to signin_path
    end
  end

  # Authorization Code expiration time (default 10 minutes).
  authorization_code_expires_in 10.minutes

  # Access token expiration time (default 2 hours).
  # If you want to disable expiration, set this to nil.
  access_token_expires_in 1.week

  # Reuse access token for the same resource owner within an application (disabled by default)
  # Rationale: https://github.com/doorkeeper-gem/doorkeeper/issues/383
  # reuse_access_token

  # Issue access tokens with refresh token (disabled by default)
  use_refresh_token

  # Provide support for an owner to be assigned to each registered application (disabled by
default)
  # Optional parameter :confirmation => true (default false) if you want to enforce ownership of
  # a registered application
  # Note: you must also run the rails g doorkeeper:application_owner generator to provide the
necessary support
  # enable_application_owner :confirmation => false
  # Define access token scopes for your provider
  # For more information go to
  # https://github.com/doorkeeper-gem/doorkeeper/wiki/Using-Scopes
  default_scopes  :profile
  optional_scopes :history, :trade

  # Change the way client credentials are retrieved from the request object.
  # By default it retrieves first from the `HTTP_AUTHORIZATION` header, then
  # falls back to the `:client_id` and `:client_secret` params from the `params` object.
  # Check out the wiki for more information on customization
  # client_credentials :from_basic, :from_params

  # Change the way access token is authenticated from the request object.
  # By default it retrieves first from the `HTTP_AUTHORIZATION` header, then
  # falls back to the `:access_token` or `:bearer_token` params from the `params` object.
  # Check out the wiki for more information on customization
  # access_token_methods :from_bearer_authorization, :from_access_token_param,
:from_bearer_param

  # Change the native redirect uri for client apps
  # When clients register with the following redirect uri, they won't be redirected to any server and
the authorization code will be displayed within the provider
  # The value can be any string. Use nil to disable this feature. When disabled, clients must
provide a valid URL
  # (Similar behaviour:
https://developers.google.com/accounts/docs/OAuth2InstalledApp#choosingredirecturi)
  #
  # native_redirect_uri 'urn:ietf:wg:oauth:2.0:oob'

  # Specify what grant flows are enabled in array of Strings. The valid
  # strings and the flows they enable are:
  #
  # "authorization_code" => Authorization Code Grant Flow
  # "implicit"           => Implicit Grant Flow
  # "password"           => Resource Owner Password Credentials Grant Flow
  # "client_credentials" => Client Credentials Grant Flow
  #
  # If not specified, Doorkeeper enables all the four grant flows.
  #
  # grant_flows %w(authorization_code implicit password client_credentials)
  # Under some circumstances you might want to have applications auto-approved,
  # so that the user skips the authorization step.
  # For example if dealing with trusted a application.
  # skip_authorization do |resource_owner, client|
  #   client.superapp? or resource_owner.admin?
  # end

  # WWW-Authenticate Realm (default "Doorkeeper").
  # realm "Doorkeeper"

  # Allow dynamic query parameters (disabled by default)
  # Some applications require dynamic query parameters on their request_uri
  # set to true if you want this to be allowed
  # wildcard_redirect_uri false
end

require_relative '../../lib/doorkeeper/access_token'

228:F:\git\coin\exchange\peatio-master\config\initializers\easy_table.rb
module EasyTable
  module Components
    module Columns
      def column_with_custom(title, label_or_opts = nil, opts = {}, &block)
        if @options[:model]
          label_or_opts ||= {}
          label_or_opts.merge!({model: @options[:model]})
        end

        if @options[:scope]
          label_or_opts ||= {}
          label_or_opts.merge!({scope: @options[:scope]})
        end

        column_without_custom(title, label_or_opts, opts, &block)
      end

      alias_method_chain :column, :custom
    end

    module Base
      def translate_with_custom(key)
        if @opts[:model]
```

```
          @opts[:model].human_attribute_name(@title)
        elsif @opts[:scope]
          I18n.t("easy_table.#{@opts[:scope]}.#{@title}")
        else
          translate_without_custom(key)
        end
      end

      alias_method_chain :translate, :custom
    end
  end
end
```

229:F:\git\coin\exchange\peatio-master\config\initializers\filter_parameter_logging.rb

```
# Be sure to restart your server when you modify this file.

# Configure sensitive parameters which will be filtered from the log file.
Rails.application.config.filter_parameters += [:password, :pin]
```

230:F:\git\coin\exchange\peatio-master\config\initializers\inflections.rb

```
# Be sure to restart your server when you modify this file.

# Add new inflection rules using the following format
# (all these examples are active by default):
# ActiveSupport::Inflector.inflections do |inflect|
#   inflect.plural /^(ox)$/i, '\1en'
#   inflect.singular /^(ox)en/i, '\1'
#   inflect.irregular 'person', 'people'
#   inflect.uncountable %w( fish sheep )
# end
#
# These inflection rules are supported but not enabled by default:
# ActiveSupport::Inflector.inflections do |inflect|
#   inflect.acronym 'RESTful'
# end

ActiveSupport::Inflector.inflections do |inflect|
  inflect.acronym 'API'
  inflect.acronym 'v2'
  inflect.acronym 'AMQP'
end
```

231:F:\git\coin\exchange\peatio-master\config\initializers\kaminari_config.rb
```
Kaminari.configure do |config|
  config.default_per_page = 10
  # config.max_per_page = nil
  # config.window = 4
  # config.outer_window = 0
  # config.left = 0
  # config.right = 0
  # config.page_method_name = :page
  # config.param_name = :page
end
```

232:F:\git\coin\exchange\peatio-master\config\initializers\mime_types.rb
```
# Be sure to restart your server when you modify this file.

# Add new mime types for use in respond_to blocks:
# Mime::Type.register "text/richtext", :rtf
# Mime::Type.register_alias "text/html", :iphone
```

233:F:\git\coin\exchange\peatio-master\config\initializers\omniauth.rb
```
Rails.application.config.middleware.use OmniAuth::Builder do
  provider :identity, fields: [:email], on_failed_registration: IdentitiesController.action(:new)
  if ENV['WEIBO_AUTH'] == "true"
    provider :weibo, ENV['WEIBO_KEY'], ENV['WEIBO_SECRET']
  end
end

OmniAuth.config.on_failure = lambda do |env|
  SessionsController.action(:failure).call(env)
end

OmniAuth.config.logger = Rails.logger

module OmniAuth
  module Strategies
   class Identity
     def request_phase
       redirect '/signin'
     end

     def registration_form
```

```
        redirect '/signup'
      end
    end
  end
end

234:F:\git\coin\exchange\peatio-master\config\initializers\pusher.rb
Pusher.app_id = ENV['PUSHER_APP']
Pusher.key    = ENV['PUSHER_KEY']
Pusher.secret = ENV['PUSHER_SECRET']
Pusher.host   = ENV['PUSHER_HOST'] || 'api.pusherapp.com'
Pusher.port   = ENV['PUSHER_PORT'].present? ? ENV['PUSHER_PORT'].to_i : 80

235:F:\git\coin\exchange\peatio-master\config\initializers\secret_token.rb
Peatio::Application.config.secret_key_base =
'4adeecaaba6c4a5474d9c8d7893dd1f4243abdcd58187d8e628d3cf0be1855b5f0b780f63de790e8
eb3fde4ba032b4d183ce623ed321c296382d7946826fcc5e'

236:F:\git\coin\exchange\peatio-master\config\initializers\simple_form.rb
require 'simple_form_extensions'

# Use this setup block to configure all options available in SimpleForm.
SimpleForm.setup do |config|
  # Wrappers are used by the form builder to generate a
  # complete input. You can remove any component from the
  # wrapper, change the order or even add your own to the
  # stack. The options given below are used to wrap the
  # whole input.
  config.wrappers :default, class: 'form-group',
    hint_class: :field_with_hint, error_class: :field_with_errors do |b|
    ## Extensions enabled by default
    # Any of these extensions can be disabled for a
    # given input by passing: `f.input EXTENSION_NAME => false`.
    # You can make any of these extensions optional by
    # renaming `b.use` to `b.optional`.

    # Determines whether to use HTML5 (:email, :url, ...)
    # and required attributes
    b.use :html5

    # Calculates placeholders automatically from I18n
    # You can also pass a string as f.input placeholder: "Placeholder"
    b.use :placeholder

    ## Optional extensions
    # They are disabled unless you pass `f.input EXTENSION_NAME => :lookup`
    # to the input. If so, they will retrieve the values from the model
    # if any exists. If you want to enable the lookup for any of those
    # extensions by default, you can change `b.optional` to `b.use`.

    # Calculates maxlength from length validations for string inputs
    b.optional :maxlength

    # Calculates pattern from format validations for string inputs
    b.optional :pattern

    # Calculates min and max from length validations for numeric inputs
    b.optional :min_max

    # Calculates readonly automatically from readonly attributes
    b.optional :readonly

    ## Inputs
    #
    b.use :label, wrap_with: { tag: :div, class: 'col-xs-8 text-right'}
    b.use :input, wrap_with: { tag: :div, class: 'col-xs-14'}
    b.use :error, wrap_with: { tag: :span, class: 'error text-danger col-xs-14 col-xs-offset-8' }
    b.use :hint,  wrap_with: { tag: :span, class: 'hint col-xs-14 col-xs-offset-8' }
  end

  config.wrappers :search, class: 'form-group',
    hint_class: :field_with_hint, error_class: :field_with_errors do |b|
    b.use :placeholder
    b.optional :maxlength
    b.optional :pattern
    b.optional :min_max
    b.optional :readonly

    ## Inputs
    b.use :label
    b.use :input, wrap_with: { tag: :div }
    b.use :hint,  wrap_with: { tag: :span, class: 'hint' }
  end
```

```
# The default wrapper to be used by the FormBuilder.
config.default_wrapper = :default

# You can define the class to use on all labels. Default is nil.
config.label_class = 'control-label'

# Define the way to render check boxes / radio buttons with labels.
# Defaults to :nested for bootstrap config.
#   inline: input + label
#   nested: label > input
config.boolean_style = :nested

# Default class for buttons
config.button_class = 'btn'

# Method used to tidy up errors. Specify any Rails Array method.
# :first lists the first message for each field.
# Use :to_sentence to list all errors for each field.
# config.error_method = :first

# Default tag used for error notification helper.
config.error_notification_tag = :div

# CSS class to add for error notification helper.
config.error_notification_class = 'alert alert-error'

# ID to add for error notification helper.
# config.error_notification_id = nil

# Series of attempts to detect a default label method for collection.
# config.collection_label_methods = [ :to_label, :name, :title, :to_s ]

# Series of attempts to detect a default value method for collection.
# config.collection_value_methods = [ :id, :to_s ]

# You can wrap a collection of radio/check boxes in a pre-defined tag, defaulting to none.
# config.collection_wrapper_tag = nil

# You can define the class to use on all collection wrappers. Defaulting to none.
# config.collection_wrapper_class = nil

# You can wrap each item in a collection of radio/check boxes with a tag,
# defaulting to :span. Please note that when using :boolean_style = :nested,
# SimpleForm will force this option to be a label.
# config.item_wrapper_tag = :span

# You can define a class to use in all item wrappers. Defaulting to none.
# config.item_wrapper_class = nil

# How the label text should be generated altogether with the required text.
# config.label_text = lambda { |label, required| "#{required} #{label}" }

# You can define the class to use on all forms. Default is simple_form.
config.default_form_class = 'simple_form'

# You can define which elements should obtain additional classes
# config.generate_additional_classes_for = [:wrapper, :label, :input]

# Whether attributes are required by default (or not). Default is true.
# config.required_by_default = true

# Tell browsers whether to use the native HTML5 validations (novalidate form option).
# These validations are enabled in SimpleForm's internal config but disabled by default
# in this configuration, which is recommended due to some quirks from different browsers.
# To stop SimpleForm from generating the novalidate option, enabling the HTML5 validations,
# change this configuration to true.
config.browser_validations = false

# Collection of methods to detect if a file type was given.
# config.file_methods = [ :mounted_as, :file?, :public_filename ]

# Custom mappings for input types. This should be a hash containing a regexp
# to match as key, and the input type that will be used when the field name
# matches the regexp as value.
# config.input_mappings = { /count/ => :integer }

# Custom wrappers for input types. This should be a hash containing an input
# type as key and the wrapper that will be used for all inputs with specified type.
# config.wrapper_mappings = { string: :prepend }

# Default priority for time_zone inputs.
# config.time_zone_priority = nil

# Default priority for country inputs.
```

```
  # config.country_priority = nil

  # When false, do not use translations for labels.
  # config.translate_labels = true

  # Automatically discover new inputs in Rails' autoload path.
  # config.inputs_discovery = true

  # Cache SimpleForm inputs discovery
  # config.cache_discovery = !Rails.env.development?

  # Default class for inputs
  config.input_class = 'form-control'
end

module SimpleForm
  class FormBuilder
    def lookup_model_names_with_custom_scope
      if scope = options[:scope]
        lookup_model_names_without_custom_scope + [scope.to_s]
      else
        lookup_model_names_without_custom_scope
      end
    end

    alias_method_chain :lookup_model_names, :custom_scope
  end
end

237:F:\git\coin\exchange\peatio-master\config\initializers\simple_form_bootstrap.rb
# Use this setup block to configure all options available in SimpleForm.
SimpleForm.setup do |config|
  config.wrappers :bootstrap, tag: 'div', class: 'control-group', error_class: 'error' do |b|
    b.use :html5
    b.use :placeholder
    b.use :label
    b.wrapper tag: 'div', class: 'controls' do |ba|
      ba.use :input
      ba.use :error, wrap_with: { tag: 'span', class: 'help-inline' }
      ba.use :hint,  wrap_with: { tag: 'p', class: 'help-block' }
    end
  end
  config.wrappers :prepend, tag: 'div', class: "control-group", error_class: 'error' do |b|
    b.use :html5
    b.use :placeholder
    b.use :label
    b.wrapper tag: 'div', class: 'controls' do |input|
      input.wrapper tag: 'div', class: 'input-prepend' do |prepend|
        prepend.use :input
      end
      input.use :hint,  wrap_with: { tag: 'span', class: 'help-block' }
      input.use :error, wrap_with: { tag: 'span', class: 'help-inline' }
    end
  end

  config.wrappers :append, tag: 'div', class: "control-group", error_class: 'error' do |b|
    b.use :html5
    b.use :placeholder
    b.use :label
    b.wrapper tag: 'div', class: 'controls' do |input|
      input.wrapper tag: 'div', class: 'input-append' do |append|
        append.use :input
      end
      input.use :hint,  wrap_with: { tag: 'span', class: 'help-block' }
      input.use :error, wrap_with: { tag: 'span', class: 'help-inline' }
    end
  end
end

module SimpleForm
  module ActionViewExtensions
    module FormHelper
      def simple_form_for_with_default_class(record, options = {}, &block)
        options[:html] ||= {}
        options[:html][:class] ||= 'form-horizontal'
        simple_form_for_without_default_class(record, options, &block)
      end
      alias_method_chain :simple_form_for, :default_class
    end
  end
end
```

238:F:\git\coin\exchange\peatio-master\config\initializers\string_extend.rb

```
class String
  def ellipsisize(len = 10)
    len = 10 unless len > 10 # assumes minimum chars at each end = 3
    gsub(%r{(....).{#{len-5},}(....)}, '\1...\2')
  end

  def mask(before: 5, after: 5)
    gsub(%r{(#{'.' * before}).*(#{'.' * after})}, '\1***\2')
  end

  def mask_address
    gsub(%r{(......).*(......)}, '\1***\2')
  end
end

module Enumerize
  class Attribute
    def value_options(options = {})
      values = if options.empty?
        @values
      else
        raise ArgumentError, 'Options cannot have both :only and :except' if options[:only] &&
options[:except]

        only = Array(options[:only]).map(&:to_s)
        except = Array(options[:except]).map(&:to_s)

        @values.reject do |value|
          if options[:only]
            !only.include?(value)
          elsif options[:except]
            except.include?(value)
          end
        end
      end
      values.map { |v| [v.text, v.value] }
    end
  end
end
```

239:F:\git\coin\exchange\peatio-master\config\initializers\withdraw_blacklist.rb

240:F:\git\coin\exchange\peatio-master\config\initializers\wrap_parameters.rb
```
# Be sure to restart your server when you modify this file.

# This file contains settings for ActionController::ParamsWrapper which
# is enabled by default.

# Enable parameter wrapping for JSON. You can disable this by setting :format to an empty array.
ActiveSupport.on_load(:action_controller) do
  wrap_parameters format: [:json] if respond_to?(:wrap_parameters)
end

# To enable root element in JSON for ActiveRecord objects.
# ActiveSupport.on_load(:active_record) do
#  self.include_root_in_json = true
# end
```

241:F:\git\coin\exchange\peatio-master\config\routes\admin.rb
```
namespace :admin do
  get '/', to: 'dashboard#index', as: :dashboard

  resources :documents
  resources :id_documents,     only: [:index, :show, :update]
  resource  :currency_deposit, only: [:new, :create]
  resources :proofs
  resources :tickets, only: [:index, :show] do
    member do
      patch :close
    end
    resources :comments, only: [:create]
  end

  resources :members, only: [:index, :show] do
    member do
      post :active
      post :toggle
    end

    resources :two_factors, only: [:destroy]
  end

  namespace :deposits do
    Deposit.descendants.each do |d|
    resources d.resource_name
   end
  end

  namespace :withdraws do
   Withdraw.descendants.each do |w|
    resources w.resource_name
   end
  end

  namespace :statistic do
   resource :members, :only => :show
   resource :orders, :only => :show
   resource :trades, :only => :show
   resource :deposits, :only => :show
   resource :withdraws, :only => :show
  end
end

242:F:\git\coin\exchange\peatio-master\config\routes.rb
Rails.application.eager_load! if Rails.env.development?

class ActionDispatch::Routing::Mapper
  def draw(routes_name)
   instance_eval(File.read(Rails.root.join("config/routes/#{routes_name}.rb")))
  end
end

Peatio::Application.routes.draw do
  use_doorkeeper

  root 'welcome#index'

  if Rails.env.development?
   mount MailsViewer::Engine => '/mails'
  end

  get '/signin' => 'sessions#new', :as => :signin
  get '/signup' => 'identities#new', :as => :signup
  get '/signout' => 'sessions#destroy', :as => :signout
  get '/auth/failure' => 'sessions#failure', :as => :failure
  match '/auth/:provider/callback' => 'sessions#create', via: [:get, :post]
```

```
resource :member, :only => [:edit, :update]
resource :identity, :only => [:edit, :update]

namespace :verify do
  resource :sms_auth,    only: [:show, :update]
  resource :google_auth, only: [:show, :update, :edit, :destroy]
end

namespace :authentications do
  resources :emails, only: [:new, :create]
  resources :identities, only: [:new, :create]
  resource :weibo_accounts, only: [:destroy]
end

scope :constraints => { id: /[a-zA-Z0-9]{32}/ } do
  resources :reset_passwords
  resources :activations, only: [:new, :edit, :update]
end

get '/documents/api_v2'
get '/documents/websocket_api'
get '/documents/oauth'
resources :documents, only: [:show]
resources :two_factors, only: [:show, :index, :update]

scope module: :private do
  resource  :id_document, only: [:edit, :update]

  resources :settings, only: [:index]
  resources :api_tokens do
    member do
      delete :unbind
    end
  end

  resources :fund_sources, only: [:create, :update, :destroy]

  resources :funds, only: [:index] do
    collection do
      post :gen_address
    end
```

```
end

namespace :deposits do
  Deposit.descendants.each do |d|
    resources d.resource_name do
      collection do
        post :gen_address
      end
    end
  end
end

namespace :withdraws do
  Withdraw.descendants.each do |w|
    resources w.resource_name
  end
end

resources :account_versions, :only => :index

resources :exchange_assets, :controller => 'assets' do
  member do
    get :partial_tree
  end
end

get '/history/orders' => 'history#orders', as: :order_history
get '/history/trades' => 'history#trades', as: :trade_history
get '/history/account' => 'history#account', as: :account_history

resources :markets, :only => :show, :constraints => MarketConstraint do
  resources :orders, :only => [:index, :destroy] do
    collection do
      post :clear
    end
  end
  resources :order_bids, :only => [:create] do
    collection do
      post :clear
    end
  end
  resources :order_asks, :only => [:create] do
```

```
        collection do
          post :clear
        end
      end
    end

    post '/pusher/auth', to: 'pusher#auth'

    resources :tickets, only: [:index, :new, :create, :show] do
      member do
        patch :close
      end
      resources :comments, only: [:create]
    end
  end

  draw :admin

  mount APIv2::Mount => APIv2::Mount::PREFIX

end
```

243:F:\git\coin\exchange\peatio-master\config\schedule.rb

```
# Use this file to easily define all of your cron jobs.
#
# It's helpful, but not entirely necessary to understand cron before proceeding.
# http://en.wikipedia.org/wiki/Cron

# Example:
#
# set :output, "/path/to/my/cron_log.log"
#
# every 2.hours do
#   command "/usr/bin/some_great_command"
#   runner "MyModel.some_method"
#   rake "some:great:rake:task"
# end
#
# every 4.days do
#   runner "AnotherModel.prune_old_records"
# end
# Learn more: http://github.com/javan/whenever

every 1.hours do
  command '/usr/local/rbenv/shims/backup perform -t database_backup'
end

every :day, at: '4am' do
  rake 'solvency:clean solvency:liability_proof'
end

244:F:\git\coin\exchange\peatio-master\db\migrate\20130624011823_create_members.rb
class CreateMembers < ActiveRecord::Migration
  def change
    create_table :members do |t|
      t.string :sn
      t.string :name
      t.string :email
      t.string :pin_digest
      t.integer :identity_id
      t.timestamps
    end

    create_table :accounts do |t|
      t.integer :member_id
      t.string  :currency
      t.decimal :balance, :precision => 32, :scale => 16
      t.decimal :locked, :precision => 32, :scale => 16
      t.timestamps
    end
  end
end

245:F:\git\coin\exchange\peatio-master\db\migrate\20130629015414_create_identities.rb
class CreateIdentities < ActiveRecord::Migration
  def change
    create_table :identities do |t|
      t.string :email
      t.string :password_digest
      t.boolean :is_active
      t.integer :retry_count
      t.boolean :is_locked
      t.datetime :locked_at
      t.datetime :last_verify_at
      t.timestamps
    end

    create_table :two_factors do |t|
      t.integer :identity_id
      t.string :otp_secret
      t.datetime :last_verify_at
    end
  end
end
```

246:F:\git\coin\exchange\peatio-master\db\migrate\20130810162023_create_reset_passwords.rb

```
class CreateResetPasswords < ActiveRecord::Migration
  def change
    create_table :reset_passwords do |t|
      t.string :email
      t.string :token
      t.datetime :expire_at
      t.integer :identity_id
      t.boolean :is_used

      t.timestamps
    end

    create_table :reset_pins do |t|
      t.string :email
      t.string :token
      t.datetime :expire_at
      t.integer :account_id
      t.boolean :is_used

      t.timestamps
    end
  end
end
```

247:F:\git\coin\exchange\peatio-master\db\migrate\20130901010953_create_orders.rb

```
class CreateOrders < ActiveRecord::Migration
  def change
    create_table :orders do |t|
      t.string :bid
      t.string :ask
      t.string :currency
      t.decimal :price, :precision => 32, :scale => 16
      t.decimal :volume, :precision => 32, :scale => 16
      t.decimal :origin_volume, :precision => 32, :scale => 16
      t.string :state
      t.datetime :done_at
      t.string :type
      t.integer :member_id
      t.timestamps
    end
  end
end
```

248:F:\git\coin\exchange\peatio-master\db\migrate\20130901154530_create_trades.rb

```
class CreateTrades < ActiveRecord::Migration
  def change
    create_table :trades do |t|
      t.decimal :price, :precision => 32, :scale => 16
      t.decimal :volume, :precision => 32, :scale => 16
      t.integer :ask_id
      t.integer :bid_id
      t.boolean :trend # true: up or equal | false: down
      t.string  :currency
      t.timestamps
    end

    create_table :members_trades do |t|
      t.integer :member_id
      t.integer :trade_id
      t.timestamps
    end
  end
end
```

249:F:\git\coin\exchange\peatio-master\db\migrate\20130903080937_create_account_versions.rb

```
class CreateAccountVersions < ActiveRecord::Migration
  def self.up
    create_table :account_versions do |t|
      t.string   :item_type, :null => false
      t.integer  :item_id,   :null => false
      t.string   :event,     :null => false
      t.string   :whodunnit
      t.text     :object
      t.datetime :created_at
      t.string   :reason
      t.integer  :ref_id
    end
    add_index :account_versions, [:item_type, :item_id]
  end

  def self.down
    remove_index :account_versions, [:item_type, :item_id]
    drop_table :account_versions
  end
end
```

250:F:\git\coin\exchange\peatio-master\db\migrate\20130904215802_add_is_active_to_two_factors.rb

```
class AddIsActiveToTwoFactors < ActiveRecord::Migration
  def change
    add_column :two_factors, :is_active, :boolean
  end
end
```

251:F:\git\coin\exchange\peatio-master\db\migrate\20130905025823_fix_account_id_by_reset_pin.rb

```
class FixAccountIdByResetPin < ActiveRecord::Migration
  def change
    rename_column :reset_pins, :account_id, :member_id
  end
end
```

252:F:\git\coin\exchange\peatio-master\db\migrate\20130905132250_add_balance_to_account_versions.rb

```
class AddBalanceToAccountVersions < ActiveRecord::Migration
  def change
    add_column :account_versions, :balance, :decimal, :precision => 32, :scale => 16
    add_column :account_versions, :amount, :decimal, :precision => 32, :scale => 16
  end
end
```

253:F:\git\coin\exchange\peatio-master\db\migrate\20130906073020_create_payment_addresses.rb

```
class CreatePaymentAddresses < ActiveRecord::Migration
  def change
    create_table :payment_addresses do |t|
      t.integer :account_id
      t.string :address

      t.timestamps
    end
  end
end
```

254:F:\git\coin\exchange\peatio-master\db\migrate\20130906073931_create_payment_transactions.rb

```
class CreatePaymentTransactions < ActiveRecord::Migration
  def change
    create_table :payment_transactions do |t|
      t.string :txid
      t.decimal :amount, :precision => 32, :scale => 16
      t.integer :confirmations
      t.string :address
      t.string :state

      t.timestamps
    end
  end
end
```

255:F:\git\coin\exchange\peatio-master\db\migrate\20130907110146_create_withdraws.rb

```
class CreateWithdraws < ActiveRecord::Migration
  def change
    create_table :withdraws do |t|
      t.integer :account_id
      t.decimal :amount, :precision => 32, :scale => 16
      t.string :payment_way
      t.string :payment_to
      t.string :state

      t.timestamps
    end
  end
end
```

```
256:F:\git\coin\exchange\peatio-master\db\migrate\20130907124647_create_deposits.rb
class CreateDeposits < ActiveRecord::Migration
  def change
    create_table :deposits do |t|
      t.integer :account_id
      t.decimal :amount, :precision => 32, :scale => 16
      t.string :payment_way
      t.string :payment_id
      t.string :state

      t.timestamps
    end
  end
end

257:F:\git\coin\exchange\peatio-
master\db\migrate\20130912144526_add_receive_at_to_payment_transactions.rb
class AddReceiveAtToPaymentTransactions < ActiveRecord::Migration
  def change
    add_column :payment_transactions, :receive_at, :datetime
  end
end

258:F:\git\coin\exchange\peatio-
master\db\migrate\20130915150504_add_payment_id_to_withdraws.rb
class AddPaymentIdToWithdraws < ActiveRecord::Migration
  def change
    add_column :withdraws, :payment_id, :string
  end
end

259:F:\git\coin\exchange\peatio-
master\db\migrate\20130918143551_add_ref_to_account_versions.rb
class AddRefToAccountVersions < ActiveRecord::Migration
  def change
    add_column :account_versions, :ref, :string
  end
end

260:F:\git\coin\exchange\peatio-master\db\migrate\20130919091853_add_sn_to_orders.rb
class AddSnToOrders < ActiveRecord::Migration
  def change
```

```
    add_column :orders, :sn, :string
  end
end
```

261:F:\git\coin\exchange\peatio-master\db\migrate\20130925154257_change_to_enumerize_in_orders.rb

```
class ChangeToEnumerizeInOrders < ActiveRecord::Migration
  def up
    change_column :orders, :bid, :integer
    change_column :orders, :ask, :integer
    change_column :orders, :state, :integer
    change_column :orders, :currency, :integer
    change_column :orders, :type, :string, :limit => 8
  end

  def down
    change_column :orders, :bid, :string
    change_column :orders, :ask, :string
    change_column :orders, :state, :string
    change_column :orders, :currency, :string
    change_column :orders, :type, :string, :limit => nil
  end
end
```

262:F:\git\coin\exchange\peatio-master\db\migrate\20130925165804_change_to_enumerize_in_trades.rb

```
class ChangeToEnumerizeInTrades < ActiveRecord::Migration
  def up
    change_column :trades, :trend, :integer
    change_column :trades, :currency, :integer
  end

  def down
    change_column :trades, :currency, :string
    change_column :trades, :trend, :boolean
  end
end
```

263:F:\git\coin\exchange\peatio-master\db\migrate\20130925171856_change_to_enumerize_in_accounts.rb

```
class ChangeToEnumerizeInAccounts < ActiveRecord::Migration
  def up
```

```
    change_column :accounts, :currency, :integer
  end

  def down
    change_column :accounts, :currency, :string
  end
end
```

264:F:\git\coin\exchange\peatio-master\db\migrate\20130925175113_change_to_enumerize_in_deposits.rb

```
class ChangeToEnumerizeInDeposits < ActiveRecord::Migration
  def up
    change_column :deposits, :payment_way, :integer
    change_column :deposits, :state, :integer
  end

  def down
    change_column :deposits, :payment_way, :string
    change_column :deposits, :state, :string
  end
end
```

265:F:\git\coin\exchange\peatio-master\db\migrate\20130926011813_change_to_enumerize_in_payment_transactions.rb

```
class ChangeToEnumerizeInPaymentTransactions < ActiveRecord::Migration
  def up
    change_column :payment_transactions, :state, :integer
  end

  def down
    change_column :payment_transactions, :state, :string
  end
end
```

266:F:\git\coin\exchange\peatio-master\db\migrate\20130926014845_change_to_enumerize_in_withdraws.rb

```
class ChangeToEnumerizeInWithdraws < ActiveRecord::Migration
  def up
    change_column :withdraws, :payment_way, :integer
    change_column :withdraws, :state, :integer
  end
  def down
    change_column :withdraws, :payment_way, :string
    change_column :withdraws, :state, :string
  end
end

267:F:\git\coin\exchange\peatio-
master\db\migrate\20130926075355_change_to_enumerize_in_account_versions.rb
class ChangeToEnumerizeInAccountVersions < ActiveRecord::Migration
  def up
    change_column :account_versions, :reason, :integer

    if index_exists?(:account_versions, [:item_type, :item_id])
      remove_index :account_versions, [:item_type, :item_id]
    end

    unless index_exists?(:account_versions, [:item_type, :item_id, :reason])
      add_index :account_versions, [:item_type, :item_id, :reason]
    end
  end

  def down
    change_column :account_versions, :reason, :string

    if index_exists?(:account_versions, [:item_type, :item_id, :reason])
      remove_index :account_versions, [:item_type, :item_id, :reason]
    end

    unless index_exists?(:account_versions, [:item_type, :item_id])
      add_index :account_versions, [:item_type, :item_id]
    end
  end
end

268:F:\git\coin\exchange\peatio-
master\db\migrate\20130926170008_change_ref_to_text_in_account_versions.rb
class ChangeRefToTextInAccountVersions < ActiveRecord::Migration
  def change
    change_column :account_versions, :ref, :text
  end
end
```

```
269:F:\git\coin\exchange\peatio-master\db\migrate\20130928080757_create_account_logs.rb
class CreateAccountLogs < ActiveRecord::Migration
  def change
    create_table :account_logs do |t|
      t.integer :member_id
      t.integer :account_id
      t.integer :reason
      t.decimal :balance, :precision => 32, :scale => 16
      t.decimal :locked, :precision => 32, :scale => 16
      t.decimal :amount, :precision => 32, :scale => 16
      t.references :modifiable, polymorphic: true
      t.text :detail
      t.timestamps

      t.index [:member_id, :reason]
      t.index [:account_id, :reason]
      t.index [:modifiable_id, :modifiable_type]
    end
  end
end

270:F:\git\coin\exchange\peatio-
master\db\migrate\20130928113620_delete_table_account_versions.rb
class DeleteTableAccountVersions < ActiveRecord::Migration
  def up
    drop_table :account_versions
  end

  def down
    raise ActiveRecord::IrreversibleMigration
  end
end

271:F:\git\coin\exchange\peatio-
master\db\migrate\20130928122042_rename_account_logs_to_account_versions.rb
class RenameAccountLogsToAccountVersions < ActiveRecord::Migration
  def change
    rename_table :account_logs, :account_versions
  end
end

272:F:\git\coin\exchange\peatio-master\db\migrate\20130928165236_add_alipay_to_members.rb
```

```
class AddAlipayToMembers < ActiveRecord::Migration
  def up
    add_column :identities, :pin_digest, :string
    remove_column :members, :pin_digest
  end

  def down
    remove_column :identities, :pin_digest
    add_column :members, :pin_digest, :string
  end
end
```

273:F:\git\coin\exchange\peatio-master\db\migrate\20130928190156_rename_member_id_to_identity_id.rb

```
class RenameMemberIdToIdentityId < ActiveRecord::Migration
  def up
    rename_column :reset_pins, :member_id, :identity_id
  end

  def down
    rename_column :reset_pins, :identity_id, :member_id
  end
end
```

274:F:\git\coin\exchange\peatio-master\db\migrate\20130928194048_add_alipay_address_to_members.rb

```
class AddAlipayAddressToMembers < ActiveRecord::Migration
  def change
    add_column :members, :alipay, :string
    add_column :members, :state, :integer
  end
end
```

275:F:\git\coin\exchange\peatio-master\db\migrate\20130929012418_create_invitations.rb

```
class CreateInvitations < ActiveRecord::Migration
  def change
    create_table :invitations do |t|
      t.boolean :is_used
      t.string :token
      t.string :email

      t.timestamps
    end
  end
end

276:F:\git\coin\exchange\peatio-master\db\migrate\20130930172651_rebuild_withdraws.rb
class RebuildWithdraws < ActiveRecord::Migration
  def up
    change_table :withdraws do |t|
      t.rename :payment_way, :address_type
      t.rename :payment_to, :address
      t.rename :payment_id, :tx_id
      t.string :address_label, :after => :address
      t.datetime :done_at, :after => :updated_at
    end

    create_table :withdraw_addresses do |t|
      t.string :label
      t.string :address
      t.integer :category
      t.integer :account_id
      t.boolean :is_locked
      t.timestamps
    end
  end

  def down
    change_table :withdraws do |t|
      t.rename :address_type, :payment_way
      t.rename :address, :payment_to
      t.rename :tx_id, :payment_id
      t.remove :address_label
      t.remove :done_at
    end

    drop_table :withdraw_addresses
  end
end

277:F:\git\coin\exchange\peatio-
master\db\migrate\20130930183833_migrate_withdraw_addresses.rb
class MigrateWithdrawAddresses < ActiveRecord::Migration
  def up
```

```
    change_table :members do |t|
      t.remove :alipay
    end
  end

  def down
    raise ActiveRecord::IrreversibleMigration
  end
end

278:F:\git\coin\exchange\peatio-
master\db\migrate\20131001103847_add_deleted_at_to_withdraw_addresses.rb
class AddDeletedAtToWithdrawAddresses < ActiveRecord::Migration
  def change
    add_column :withdraw_addresses, :deleted_at, :datetime
  end
end

279:F:\git\coin\exchange\peatio-master\db\migrate\20131002012809_add_fee_to_withdraws.rb
class AddFeeToWithdraws < ActiveRecord::Migration
  def change
    add_column :withdraws, :member_id, :integer, :after => :account_id
    add_column :withdraws, :currency, :integer, :after => :member_id
    add_column :withdraws, :fee, :decimal, :precision => 32, :scale => 16, :after => :amount
  end
end

280:F:\git\coin\exchange\peatio-master\db\migrate\20131002190141_rebuild_deposits.rb
class RebuildDeposits < ActiveRecord::Migration
  def change
    change_table :deposits do |t|
      t.integer :member_id, :after => :account_id
      t.integer :currency, :after => :member_id
      t.datetime :done_at
      t.rename :payment_way, :category
      t.rename :payment_id, :tx_id
    end
  end
end

281:F:\git\coin\exchange\peatio-master\db\migrate\20131003003357_add_address_to_deposits.rb
class AddAddressToDeposits < ActiveRecord::Migration
```

```
  def change
    change_table :deposits do |t|
      t.string :address, :after => :amount
      t.string :address_label, :after => :address
      t.rename :category, :address_type
    end
  end
end

282:F:\git\coin\exchange\peatio-
master\db\migrate\20131003021225_rename_txid_to_payment_transactions.rb
class RenameTxidToPaymentTransactions < ActiveRecord::Migration
  def up
    change_table :payment_transactions do |t|
      t.datetime :dont_at
    end
  end

  def down
    change_table :payment_transactions do |t|
      t.remove :dont_at
    end
  end
end

283:F:\git\coin\exchange\peatio-master\db\migrate\20131006183340_create_tokens.rb
class CreateTokens < ActiveRecord::Migration
  def up
    create_table :tokens do |t|
      t.string :token
      t.datetime :expire_at
      t.integer :identity_id
      t.boolean :is_used
      t.string :type

      t.timestamps
    end

    add_index :tokens, [:type, :token, :expire_at, :is_used]
  end

  def down
```

```
    drop_table :tokens
  end
end

284:F:\git\coin\exchange\peatio-master\db\migrate\20131009132505_create_documents.rb
class CreateDocuments < ActiveRecord::Migration
  def change
    create_table :documents do |t|
      t.string :key
      t.string :title
      t.text :body
      t.boolean :is_auth
      t.timestamps
    end
  end
end

285:F:\git\coin\exchange\peatio-master\db\migrate\20131022035138_add_in_out_to_accounts.rb
class AddInOutToAccounts < ActiveRecord::Migration
  def change
    add_column :accounts, :in, :decimal, :precision => 32, :scale => 16
    add_column :accounts, :out, :decimal, :precision => 32, :scale => 16
  end
end

286:F:\git\coin\exchange\peatio-
master\db\migrate\20131027012836_change_in_out_to_accounts.rb
class ChangeInOutToAccounts < ActiveRecord::Migration
  def up
    change_column :accounts, :in, :decimal, :precision => 32, :scale => 16
    change_column :accounts, :out, :decimal, :precision => 32, :scale => 16
  end

  def down
    change_column :accounts, :in, :decimal, :precision => 32, :scale => 16
    change_column :accounts, :out, :decimal, :precision => 32, :scale => 16
  end
end

287:F:\git\coin\exchange\peatio-
master\db\migrate\20131110214254_add_currency_to_payment_transactions.rb
class AddCurrencyToPaymentTransactions < ActiveRecord::Migration
```

```
  def change
    add_column :payment_transactions, :currency, :integer
  end
end
```

288:F:\git\coin\exchange\peatio-master\db\migrate\20131130190923_remove_pin_digest_from_identities.rb

```
class RemovePinDigestFromIdentities < ActiveRecord::Migration
  def up
    remove_column :identities, :pin_digest
  end

  def down
    add_column :identities, :pin_digest, :string
  end
end
```

289:F:\git\coin\exchange\peatio-master\db\migrate\20131201011127_drop_reset_pins.rb

```
class DropResetPins < ActiveRecord::Migration
  def up
    drop_table :reset_pins
  end

  def down
    create_table :reset_pins do |t|
      t.string :email
      t.string :token
      t.datetime :expire_at
      t.integer :identity_id
      t.boolean :is_used

      t.timestamps
    end
  end
end
```

290:F:\git\coin\exchange\peatio-master\db\migrate\20131204020953_add_currency_to_account_versions.rb

```
class AddCurrencyToAccountVersions < ActiveRecord::Migration
  def up
    add_column :account_versions, :currency, :integer
    remove_column :account_versions, :detail
  end

  def down
    raise ActiveRecord::IrreversibleMigration
  end
end
```

291:F:\git\coin\exchange\peatio-master\db\migrate\20131208012814_fix_payment_address_currency.rb

```
class FixPaymentAddressCurrency < ActiveRecord::Migration
  def change
    add_column :payment_addresses, :currency, :integer
  end
end
```

292:F:\git\coin\exchange\peatio-master\db\migrate\20131224162832_add_sn_to_withdraws.rb

```
class AddSnToWithdraws < ActiveRecord::Migration
  def change
    add_column :withdraws, :sn, :string, after: :id
  end
end
```

293:F:\git\coin\exchange\peatio-master\db\migrate\20140101175408_add_fee_to_account_versions.rb

```
class AddFeeToAccountVersions < ActiveRecord::Migration
  def change
    add_column :account_versions, :fee, :decimal, precision: 32, scale: 16, after: :locked
  end
end
```

294:F:\git\coin\exchange\peatio-master\db\migrate\20140102024125_add_fun_to_account_versions.rb

```
class AddFunToAccountVersions < ActiveRecord::Migration
  def change
    add_column :account_versions, :fun, :integer
  end
end
```

295:F:\git\coin\exchange\peatio-master\db\migrate\20140102172835_acts_as_taggable_on_migration.acts_as_taggable_on_engine.rb

```
# This migration comes from acts_as_taggable_on_engine (originally 1)
class ActsAsTaggableOnMigration < ActiveRecord::Migration
  def self.up
    create_table :tags do |t|
      t.string :name
    end

    create_table :taggings do |t|
      t.references :tag

      # You should make sure that the column created is
      # long enough to store the required class names.
      t.references :taggable, :polymorphic => true
      t.references :tagger, :polymorphic => true

      # Limit is created to prevent MySQL error on index
      # length for MyISAM table type: http://bit.ly/vgW2Ql
      t.string :context, :limit => 128

      t.datetime :created_at
    end

    add_index :taggings, :tag_id
    add_index :taggings, [:taggable_id, :taggable_type, :context]
  end

  def self.down
    drop_table :taggings
    drop_table :tags
  end
end
```

296:F:\git\coin\exchange\peatio-master\db\migrate\20140102172836_add_missing_unique_indices.acts_as_taggable_on_engine.rb

```
# This migration comes from acts_as_taggable_on_engine (originally 2)
class AddMissingUniqueIndices < ActiveRecord::Migration

  def self.up
    add_index :tags, :name, unique: true

    remove_index :taggings, :tag_id
    remove_index :taggings, [:taggable_id, :taggable_type, :context]
    add_index :taggings,
      [:tag_id, :taggable_id, :taggable_type, :context, :tagger_id, :tagger_type],
      unique: true, name: 'taggings_idx'
   end

  def self.down
    remove_index :tags, :name

    remove_index :taggings, name: 'tagging_idx'
    add_index :taggings, :tag_id
    add_index :taggings, [:taggable_id, :taggable_type, :context]
  end

end
```

297:F:\git\coin\exchange\peatio-master\db\migrate\20140105034746_drop_reset_passwords.rb

```
class DropResetPasswords < ActiveRecord::Migration
  def up
    if ActiveRecord::Base.connection.table_exists? :reset_passwords
      drop_table :reset_passwords
    end
  end

  def down
    raise ActiveRecord::IrreversibleMigration
  end
end
```

298:F:\git\coin\exchange\peatio-master\db\migrate\20140302094520_rename_identity_id_to_member_id.rb

```
class RenameIdentityIdToMemberId < ActiveRecord::Migration
  def change
    change_table :tokens do |t|
      t.rename :identity_id, :member_id
    end

    change_table :two_factors do |t|
      t.rename :identity_id, :member_id
    end
  end
end
```

299:F:\git\coin\exchange\peatio-master\db\migrate\20140302094729_migration_data_identity_id_to_member_id.rb

```ruby
class MigrationDataIdentityIdToMemberId < ActiveRecord::Migration
  def up
    execute <<-SQL
      DELETE FROM tokens WHERE type = 'ResetPin'
    SQL

    Token.all.each do |t|
      id = Member.find_by_identity_id(t.member_id)
      t.update_column :member_id, id
    end

    TwoFactor.all.each do |t|
      id = Member.find_by_identity_id(t.member_id)
      t.update_column :member_id, id
    end
  end

  def down
    raise ActiveRecord::IrreversibleMigration
  end
end
```

300:F:\git\coin\exchange\peatio-master\db\migrate\20140302161905_create_authentications.rb

```ruby
class CreateAuthentications < ActiveRecord::Migration
  def change
    create_table :authentications do |t|
      t.string :provider
      t.string :uid
      t.string :token
      t.string :secret
      t.integer :member_id

      t.timestamps
    end

    add_index :authentications, :member_id
    add_index :authentications, [:provider, :uid]
  end
end
```

301:F:\git\coin\exchange\peatio-master\db\migrate\20140303060739_add_activated_to_members.rb

```
class AddActivatedToMembers < ActiveRecord::Migration
  def change
    add_column :members, :activated, :boolean
  end
end
```

302:F:\git\coin\exchange\peatio-master\db\migrate\20140303080054_rename_is_active_to_activated.rb

```
class RenameIsActiveToActivated < ActiveRecord::Migration
  def change
    change_table :two_factors do |t|
      t.rename :is_active, :activated
    end
  end
end
```

303:F:\git\coin\exchange\peatio-master\db\migrate\20140304015055_create_documents_translations.rb

```
class CreateDocumentsTranslations < ActiveRecord::Migration
  def up
    Document.create_translation_table!(
      { :title => :string, :body => :text },
      { :migrate_data => true }
    )
  end

  def down
    Document.drop_translation_table! :migrate_data => true
  end
end
```

304:F:\git\coin\exchange\peatio-master\db\migrate\20140306020939_create_id_documents.rb

```
class CreateIdDocuments < ActiveRecord::Migration
  def change
    create_table :id_documents do |t|
      t.integer :category
      t.string :name
      t.string :sn
      t.integer :member_id
      t.timestamps
    end
  end
end

305:F:\git\coin\exchange\peatio-
master\db\migrate\20140306021833_add_verified_to_id_documents.rb
class AddVerifiedToIdDocuments < ActiveRecord::Migration
  def change
    add_column :id_documents, :verified, :boolean
  end
end

306:F:\git\coin\exchange\peatio-
master\db\migrate\20140312061206_add_aasm_state_to_withdraws.rb
class AddAasmStateToWithdraws < ActiveRecord::Migration
  def change
    add_column :withdraws, :aasm_state, :string
  end
end

307:F:\git\coin\exchange\peatio-master\db\migrate\20140312071704_add_sum_to_withdraws.rb
class AddSumToWithdraws < ActiveRecord::Migration
  def change
    add_column :withdraws, :sum, :decimal, precision: 32, scale: 16
  end
end

308:F:\git\coin\exchange\peatio-
master\db\migrate\20140319022202_add_partial_tree_to_accounts.rb
class AddPartialTreeToAccounts < ActiveRecord::Migration
  def change
    add_column :accounts, :partial_tree, :text
  end
end

309:F:\git\coin\exchange\peatio-master\db\migrate\20140319022302_create_proofs.rb
class CreateProofs < ActiveRecord::Migration
  def change
    create_table :proofs do |t|
      t.string  :root
      t.integer :currency
      t.boolean :ready, default: false

      t.timestamps
    end
  end
end
```

310:F:\git\coin\exchange\peatio-master\db\migrate\20140320142701_create_versions.rb

```
class CreateVersions < ActiveRecord::Migration
  def change
    create_table :versions do |t|
      t.string   :item_type, :null => false
      t.integer  :item_id,   :null => false
      t.string   :event,     :null => false
      t.string   :whodunnit
      t.text     :object
      t.datetime :created_at
    end
    add_index :versions, [:item_type, :item_id]
  end
end
```

311:F:\git\coin\exchange\peatio-master\db\migrate\20140324060148_rename_withdraw_addresses_to_fund_sources.rb

```
class RenameWithdrawAddressesToFundSources < ActiveRecord::Migration
  def change
    rename_table :withdraw_addresses, :fund_sources
  end
end
```

312:F:\git\coin\exchange\peatio-master\db\migrate\20140324062812_rename_address_column_by_withdraws.rb

```
class RenameAddressColumnByWithdraws < ActiveRecord::Migration
  def change
    add_column :fund_sources, :member_id, :integer, :after => :id
    add_column :fund_sources, :currency, :integer, :after => :member_id
    rename_column :fund_sources, :label, :extra
    rename_column :fund_sources, :address, :uid

    rename_column :withdraws, :address, :fund_source_uid
    rename_column :withdraws, :address_label, :fund_source_extra
    rename_column :withdraws, :address_type, :withdraw_channel_id
  end
end

313:F:\git\coin\exchange\peatio-master\db\migrate\20140326170234_change_deposits.rb
class ChangeDeposits < ActiveRecord::Migration
  def change
    rename_column :deposits, :address, :fund_source_uid
    rename_column :deposits, :address_label, :fund_source_extra
    rename_column :deposits, :address_type, :channel_id
    rename_column :deposits, :tx_id, :txid
    add_column :deposits, :fee, :decimal, :precision => 32, :scale => 16, :after => :amount
    add_column :deposits, :aasm_state, :string, :after => :state
  end
end

314:F:\git\coin\exchange\peatio-
master\db\migrate\20140326191837_add_deposit_id_to_payment_transactions.rb
class AddDepositIdToPaymentTransactions < ActiveRecord::Migration
  def change
    add_column :payment_transactions, :aasm_state, :string, :after => :state
    add_column :payment_transactions, :channel_id, :integer, :after => :aasm_state
  end
end

315:F:\git\coin\exchange\peatio-master\db\migrate\20140327044440_change_withdraws.rb
class ChangeWithdraws < ActiveRecord::Migration
  def change
    rename_column :withdraws, :withdraw_channel_id, :channel_id
    rename_column :withdraws, :tx_id, :txid
    rename_column :withdraws, :fund_source_uid, :fund_uid
    rename_column :withdraws, :fund_source_extra, :fund_extra
  end
end

316:F:\git\coin\exchange\peatio-
master\db\migrate\20140327062025_add_memo_and_remove_fund_source_to_deposits.rb
class AddMemoAndRemoveFundSourceToDeposits < ActiveRecord::Migration
  def change
    add_column :deposits, :memo, :string
    rename_column :deposits, :fund_source_uid, :fund_uid
    rename_column :deposits, :fund_source_extra, :fund_extra
  end
end

317:F:\git\coin\exchange\peatio-
master\db\migrate\20140327065708_rename_fund_sources_category_to_channel_id.rb
class RenameFundSourcesCategoryToChannelId < ActiveRecord::Migration
  def change
    rename_column :fund_sources, :category, :channel_id
  end
end

318:F:\git\coin\exchange\peatio-
master\db\migrate\20140327105217_remove_fund_sources_account_id.rb
class RemoveFundSourcesAccountId < ActiveRecord::Migration
  def change
    remove_column :fund_sources, :account_id
  end
end

319:F:\git\coin\exchange\peatio-master\db\migrate\20140328101707_add_type_to_deposits.rb
class AddTypeToDeposits < ActiveRecord::Migration
  def change
    add_column :deposits, :type, :string
  end
end

320:F:\git\coin\exchange\peatio-master\db\migrate\20140329070543_remove_channel_id.rb
class RemoveChannelId < ActiveRecord::Migration
  def change
    remove_column :deposits, :channel_id
    remove_column :payment_transactions, :channel_id
  end
end

321:F:\git\coin\exchange\peatio-
master\db\migrate\20140331084541_fund_sources_is_locked_default_to_false.rb
class FundSourcesIsLockedDefaultToFalse < ActiveRecord::Migration
  def change
    change_column_default :fund_sources, :is_locked, false
  end
end
```

```
322:F:\git\coin\exchange\peatio-master\db\migrate\20140402043033_create_partial_trees.rb
class CreatePartialTrees < ActiveRecord::Migration
  def up
    create_table :partial_trees do |t|
      t.integer :proof_id, null: false
      t.integer :account_id, null: false
      t.text :json, null: false

      t.timestamps
    end

    remove_column :accounts, :partial_tree

    Proof.delete_all
  end
end

323:F:\git\coin\exchange\peatio-master\db\migrate\20140403031847_create_api_tokens.rb
class CreateAPITokens < ActiveRecord::Migration
  def change
    create_table :api_tokens do |t|
      t.integer :member_id, null: false
      t.string :access_key, null: false, limit: 50
      t.string :secret_key, null: false, limit: 50

      t.timestamps
    end

    add_index :api_tokens, :access_key, unique: true
    add_index :api_tokens, :secret_key, unique: true
  end
end

324:F:\git\coin\exchange\peatio-master\db\migrate\20140403070840_add_type_to_withdraws.rb
class AddTypeToWithdraws < ActiveRecord::Migration
  def up
    add_column :withdraws, :type, :string

    Withdraw.all.each do |withdraw|
      type = withdraw.currency == 'btc' ? 'Withdraws::Satoshi' : 'Withdraws::Bank'
      withdraw.update_column :type, type
    end
  end

  def down
    remove_column :withdraws, :type
  end
end

325:F:\git\coin\exchange\peatio-
master\db\migrate\20140404074816_add_currency_index_to_trades.rb
class AddCurrencyIndexToTrades < ActiveRecord::Migration
  def change
    add_index :trades, :currency
  end
end

326:F:\git\coin\exchange\peatio-
master\db\migrate\20140404101823_add_ask_member_id_and_bid_member_id_to_trades.rb
class AddAskMemberIdAndBidMemberIdToTrades < ActiveRecord::Migration
  def change
    add_column :trades, :ask_member_id, :integer
    add_column :trades, :bid_member_id, :integer

    add_index :trades, :ask_member_id
    add_index :trades, :bid_member_id
  end
end

327:F:\git\coin\exchange\peatio-
master\db\migrate\20140405053744_remove_withdraws_state_and_channel_id.rb
class RemoveWithdrawsStateAndChannelId < ActiveRecord::Migration
  def change
    remove_column :withdraws, :channel_id
    remove_column :withdraws, :state
  end
end

328:F:\git\coin\exchange\peatio-master\db\migrate\20140407011310_add_source_to_orders.rb
class AddSourceToOrders < ActiveRecord::Migration
  def change
    add_column :orders, :source, :string, null: false
    Order.update_all(source: 'Web')
  end
```

```
end

329:F:\git\coin\exchange\peatio-
master\db\migrate\20140416143239_add_country_code_to_members.rb
class AddCountryCodeToMembers < ActiveRecord::Migration
  def change
    add_column :members, :country_code, :integer
  end
end

330:F:\git\coin\exchange\peatio-
master\db\migrate\20140416143352_add_phone_number_to_members.rb
class AddPhoneNumberToMembers < ActiveRecord::Migration
  def change
    add_column :members, :phone_number, :string
  end
end

331:F:\git\coin\exchange\peatio-
master\db\migrate\20140416151403_add_phone_number_verified_to_members.rb
class AddPhoneNumberVerifiedToMembers < ActiveRecord::Migration
  def change
    add_column :members, :phone_number_verified, :boolean
  end
end

332:F:\git\coin\exchange\peatio-master\db\migrate\20140416194209_remove_table_invitations.rb
class RemoveTableInvitations < ActiveRecord::Migration
  def change
    drop_table :invitations
  end
end

333:F:\git\coin\exchange\peatio-
master\db\migrate\20140416194300_remove_table_members_trades.rb
class RemoveTableMembersTrades < ActiveRecord::Migration
  def change
    drop_table :members_trades
  end
end

334:F:\git\coin\exchange\peatio-master\db\migrate\20140418082715_add_sum_to_proofs.rb
```

```
class AddSumToProofs < ActiveRecord::Migration
  def change
    add_column :proofs, :sum, :string
    add_column :partial_trees, :sum, :string
  end
end
```

335:F:\git\coin\exchange\peatio-master\db\migrate\20140421061712_add_index_on_accounts.rb

```
class AddIndexOnAccounts < ActiveRecord::Migration
  def change
    add_index :accounts, [:member_id, :currency]
    add_index :accounts, :member_id
  end
end
```

336:F:\git\coin\exchange\peatio-master\db\migrate\20140421080408_add_type_to_two_factors.rb

```
class AddTypeToTwoFactors < ActiveRecord::Migration
  def change
    add_column :two_factors, :type, :string
  end
end
```

337:F:\git\coin\exchange\peatio-master\db\migrate\20140428203350_add_desc_and_keyword_to_documents.rb

```
class AddDescAndKeywordToDocuments < ActiveRecord::Migration
  def change
    add_column :documents, :desc, :text
    add_column :documents, :keywords, :text

    add_column :document_translations, :desc, :text
    add_column :document_translations, :keywords, :text
  end
end
```

338:F:\git\coin\exchange\peatio-master\db\migrate\20140507120249_add_addresses_to_proofs.rb

```
class AddAddressesToProofs < ActiveRecord::Migration
  def change
    add_column :proofs, :addresses, :text
  end
end
```

```
339:F:\git\coin\exchange\peatio-master\db\migrate\20140524014413_add_ord_type_to_orders.rb
class AddOrdTypeToOrders < ActiveRecord::Migration
  def change
    add_column :orders, :ord_type, :string, limit: 10
  end
end

340:F:\git\coin\exchange\peatio-master\db\migrate\20140530133210_add_locked_to_orders.rb
class AddLockedToOrders < ActiveRecord::Migration
  def change
    add_column :orders, :locked,        :decimal, precision: 32, scale: 16
    add_column :orders, :origin_locked, :decimal, precision: 32, scale: 16
  end
end

341:F:\git\coin\exchange\peatio-
master\db\migrate\20140531054739_add_used_funds_to_trades.rb
class AddUsedFundsToTrades < ActiveRecord::Migration
  def change
    add_column :trades, :funds, :decimal, precision: 32, scale: 16
  end
end

342:F:\git\coin\exchange\peatio-
master\db\migrate\20140618004355_add_displayname_to_members.rb
class AddDisplaynameToMembers < ActiveRecord::Migration
  def change
    add_column :members, :display_name, :string, after: :name
  end
end

343:F:\git\coin\exchange\peatio-master\db\migrate\20140702035833_add_balance_to_proofs.rb
class AddBalanceToProofs < ActiveRecord::Migration
  def change
    add_column :proofs, :balance, :string, limit: 30
  end
end

344:F:\git\coin\exchange\peatio-
master\db\migrate\20140703065321_add_order_id_indices_to_trades.rb
class AddOrderIdIndicesToTrades < ActiveRecord::Migration
  def change
```

```
    add_index :trades, :ask_id
    add_index :trades, :bid_id
  end
end
```

345:F:\git\coin\exchange\peatio-master\db\migrate\20140703070953_add_funds_received_to_orders.rb

```
class AddFundsReceivedToOrders < ActiveRecord::Migration
  def change
    add_column :orders, :funds_received, :decimal, precision: 32, scale: 16, default: 0
  end
end
```

346:F:\git\coin\exchange\peatio-master\db\migrate\20140707115022_create_audit_logs.rb

```
class CreateAuditLogs < ActiveRecord::Migration
  def change
    create_table :audit_logs do |t|
      # Common Properties
      t.string :type
      t.integer :operator_id
      t.timestamps
      t.integer :auditable_id
      t.string :auditable_type

      # For Deposit and Withdraw
      t.string :source_state
      t.string :target_state
    end

    add_index :audit_logs, :operator_id
    add_index :audit_logs, [:auditable_id, :auditable_type]
  end
end
```

347:F:\git\coin\exchange\peatio-master\db\migrate\20140709084906_create_tickets.rb

```
class CreateTickets < ActiveRecord::Migration
  def change
    create_table :tickets do |t|
      t.string :title
      t.text :content
      t.string :aasm_state
      t.integer :author_id
      t.timestamps
    end
  end
end

348:F:\git\coin\exchange\peatio-master\db\migrate\20140709085158_create_comments.rb
class CreateComments < ActiveRecord::Migration
  def change
    create_table :comments do |t|
      t.text :content
      t.integer :author_id
      t.integer :ticket_id

      t.timestamps
    end
  end
end

349:F:\git\coin\exchange\peatio-
master\db\migrate\20140712030803_add_disabled_to_members.rb
class AddDisabledToMembers < ActiveRecord::Migration
  def change
    add_column :members, :disabled, :boolean, default: false
  end
end

350:F:\git\coin\exchange\peatio-master\db\migrate\20140714143823_unread_migration.rb
class UnreadMigration < ActiveRecord::Migration
  def self.up
    create_table :read_marks, :force => true do |t|
      t.integer  :readable_id
      t.integer  :member_id,       :null => false
      t.string   :readable_type, :null => false, :limit => 20
      t.datetime :timestamp
    end

    add_index :read_marks, [:member_id]
    add_index :read_marks, [:readable_type, :readable_id]

  end
  def self.down
    drop_table :read_marks
  end
end

351:F:\git\coin\exchange\peatio-
master\db\migrate\20140715002401_add_more_fields_to_id_documents_table.rb
class AddMoreFieldsToIdDocumentsTable < ActiveRecord::Migration
  def change
    add_column :id_documents, :birth_date, :date
    add_column :id_documents, :address, :text
    add_column :id_documents, :city,    :string
    add_column :id_documents, :country, :string
    add_column :id_documents, :zipcode, :string
    add_column :id_documents, :id_bill_type, :integer
  end
end

352:F:\git\coin\exchange\peatio-
master\db\migrate\20140715040545_remove_name_field_from_members_table.rb
class RemoveNameFieldFromMembersTable < ActiveRecord::Migration
  def up
    remove_column :members, :name
  end

  def down
    add_column :members, :name, :string
  end
end

353:F:\git\coin\exchange\peatio-
master\db\migrate\20140715083857_add_aasm_state_to_id_document.rb
class AddAasmStateToIdDocument < ActiveRecord::Migration
  def change
    add_column :id_documents, :aasm_state, :string
  end
end

354:F:\git\coin\exchange\peatio-master\db\migrate\20140717033231_add_assets_table.rb
class AddAssetsTable < ActiveRecord::Migration
  def change
    create_table :assets do |t|
      t.string  :type
      t.integer :attachable_id
      t.string  :attachable_type
      t.string  :file
    end
  end
end
```

355:F:\git\coin\exchange\peatio-master\db\migrate\20140718134132_rename_id_documents_column_category_to_id_document_type.rb

```
class RenameIdDocumentsColumnCategoryToIdDocumentType < ActiveRecord::Migration
  def change
    rename_column :id_documents, :category, :id_document_type
  end
end
```

356:F:\git\coin\exchange\peatio-master\db\migrate\20140718141345_rename_id_documents_column_from_sn_to_id_document_number.rb

```
class RenameIdDocumentsColumnFromSnToIdDocumentNumber < ActiveRecord::Migration
  def change
    rename_column :id_documents, :sn, :id_document_number
  end
end
```

357:F:\git\coin\exchange\peatio-master\db\migrate\20140721125900_remove_column_verified_from_id_documents.rb

```
class RemoveColumnVerifiedFromIdDocuments < ActiveRecord::Migration
  def change
    remove_column :id_documents, :verified
  end
end
```

358:F:\git\coin\exchange\peatio-master\db\migrate\20140724033014_add_trusted_ip_list_to_api_tokens.rb

```
class AddTrustedIpListToAPITokens < ActiveRecord::Migration
  def change
    add_column :api_tokens, :trusted_ip_list, :string
  end
end
```

359:F:\git\coin\exchange\peatio-master\db\migrate\20140803202610_remove_channel_id_from_fund_sources.rb
```
class RemoveChannelIdFromFundSources < ActiveRecord::Migration
  def change
    remove_column :fund_sources, :channel_id
  end
end
```

360:F:\git\coin\exchange\peatio-master\db\migrate\20140804002557_add_api_disabled_to_members.rb
```
class AddAPIDisabledToMembers < ActiveRecord::Migration
  def change
    add_column :members, :api_disabled, :boolean, default: false
  end
end
```

361:F:\git\coin\exchange\peatio-master\db\migrate\20140804151249_change_default_of_withdrao_fee.rb
```
class ChangeDefaultOfWithdraoFee < ActiveRecord::Migration
  def change
    change_column :withdraws, :sum, :decimal, precision: 32, scale: 16, default: 0, null: false
  end
end
```

362:F:\git\coin\exchange\peatio-master\db\migrate\20140806141035_add_index_to_orders.rb
```
class AddIndexToOrders < ActiveRecord::Migration
  def change
    add_index :orders, :member_id, using: :btree
    add_index :orders, [:currency, :state], using: :btree
  end
end
```

363:F:\git\coin\exchange\peatio-master\db\migrate\20140806141419_add_index_to_trades.rb
```
class AddIndexToTrades < ActiveRecord::Migration
  def change
    add_index :trades, :created_at, using: :btree
  end
end
```

364:F:\git\coin\exchange\peatio-master\db\migrate\20140819085359_add_index_to_order_state.rb
```
class AddIndexToOrderState < ActiveRecord::Migration
```

```
  def change
    add_index :orders, :state
  end
end

365:F:\git\coin\exchange\peatio-
master\db\migrate\20140819090417_add_index_on_orders_member_id_and_state.rb
class AddIndexOnOrdersMemberIdAndState < ActiveRecord::Migration
  def change
    add_index :orders, [:member_id, :state]
  end
end

366:F:\git\coin\exchange\peatio-master\db\migrate\20140826083906_add_label_to_api_token.rb
class AddLabelToAPIToken < ActiveRecord::Migration
  def change
    add_column :api_tokens, :label, :string
  end
end

367:F:\git\coin\exchange\peatio-
master\db\migrate\20140826093508_add_refreshed_at_to_two_factors.rb
class AddRefreshedAtToTwoFactors < ActiveRecord::Migration
  def change
    add_column :two_factors, :refreshed_at, :timestamp
  end
end

368:F:\git\coin\exchange\peatio-
master\db\migrate\20140902112641_create_simple_captcha_data.rb
class CreateSimpleCaptchaData < ActiveRecord::Migration
  def self.up
    create_table :simple_captcha_data do |t|
      t.string :key, :limit => 40
      t.string :value, :limit => 6
      t.timestamps
    end

    add_index :simple_captcha_data, :key, :name => "idx_key"
  end

  def self.down
```

```
    drop_table :simple_captcha_data
  end
end

369:F:\git\coin\exchange\peatio-
master\db\migrate\20140920062130_add_type_to_payment_transactions.rb
class AddTypeToPaymentTransactions < ActiveRecord::Migration
  def up
    add_column :payment_transactions, :type, :string, limit: 60
    PaymentTransaction.update_all type: 'PaymentTransaction::Default'
    add_index :payment_transactions, :type
  end

  def down
    remove_index :payment_transactions, :type
    remove_column :payment_transactions, :type
  end
end

370:F:\git\coin\exchange\peatio-
master\db\migrate\20141002075102_add_tx_out_to_payment_transactions.rb
class AddTxOutToPaymentTransactions < ActiveRecord::Migration
  def change
    add_column :payment_transactions, :txout, :integer
    add_index :payment_transactions, [:txid, :txout]
  end
end

371:F:\git\coin\exchange\peatio-
master\db\migrate\20141003040822_add_payment_transaction_id_to_deposits.rb
class AddPaymentTransactionIdToDeposits < ActiveRecord::Migration
  def change
    add_column :deposits, :payment_transaction_id, :integer
  end
end

372:F:\git\coin\exchange\peatio-master\db\migrate\20141003061259_add_txout_to_deposits.rb
class AddTxoutToDeposits < ActiveRecord::Migration
  def change
    add_column :deposits, :txout, :integer
    add_index :deposits, [:txid, :txout]
  end
```

```
end

373:F:\git\coin\exchange\peatio-
master\db\migrate\20141010083930_remove_phone_number_verified_from_members.rb
class RemovePhoneNumberVerifiedFromMembers < ActiveRecord::Migration
  def change
    remove_column :members, :phone_number_verified
  end
end

374:F:\git\coin\exchange\peatio-
master\db\migrate\20141012124243_set_token_is_used_to_false_as_default.rb
class SetTokenIsUsedToFalseAsDefault < ActiveRecord::Migration
  def change
    change_column :tokens, :is_used, :boolean, default: false
  end
end

375:F:\git\coin\exchange\peatio-
master\db\migrate\20141014085101_add_nickname_to_members.rb
class AddNicknameToMembers < ActiveRecord::Migration
  def change
    add_column :members, :nickname, :string
  end
end

376:F:\git\coin\exchange\peatio-
master\db\migrate\20141015034040_add_nickname_to_authentications.rb
class AddNicknameToAuthentications < ActiveRecord::Migration
  def change
    add_column :authentications, :nickname, :string
  end
end

377:F:\git\coin\exchange\peatio-
master\db\migrate\20141105023306_create_doorkeeper_tables.rb
class CreateDoorkeeperTables < ActiveRecord::Migration
  def change
    create_table :oauth_applications do |t|
      t.string  :name,         null: false
      t.string  :uid,          null: false
      t.string  :secret,       null: false
      t.text    :redirect_uri, null: false
      t.timestamps
    end

    add_index :oauth_applications, :uid, unique: true

    create_table :oauth_access_grants do |t|
      t.integer  :resource_owner_id, null: false
      t.integer  :application_id,    null: false
      t.string   :token,             null: false
      t.integer  :expires_in,        null: false
      t.text     :redirect_uri,      null: false
      t.datetime :created_at,        null: false
      t.datetime :revoked_at
      t.string   :scopes
    end

    add_index :oauth_access_grants, :token, unique: true

    create_table :oauth_access_tokens do |t|
      t.integer  :resource_owner_id
      t.integer  :application_id
      t.string   :token,             null: false
      t.string   :refresh_token
      t.integer  :expires_in
      t.datetime :revoked_at
      t.datetime :created_at,        null: false
      t.string   :scopes
    end

    add_index :oauth_access_tokens, :token, unique: true
    add_index :oauth_access_tokens, :resource_owner_id
    add_index :oauth_access_tokens, :refresh_token, unique: true
  end
end
```

378:F:\git\coin\exchange\peatio-master\db\migrate\20141105090746_add_oauth_columns_to_api_tokens.rb

```
class AddOauthColumnsToAPITokens < ActiveRecord::Migration
  def change
    add_column :api_tokens, :oauth_access_token_id, :integer
    add_column :api_tokens, :expire_at, :datetime
    add_column :api_tokens, :scopes, :string
  end
end

379:F:\git\coin\exchange\peatio-
master\db\migrate\20141107031140_add_deleted_at_to_api_tokens_and_oauth_tokens.rb
class AddDeletedAtToAPITokensAndOauthTokens < ActiveRecord::Migration
  def change
    add_column :api_tokens, :deleted_at, :datetime
    add_column :oauth_access_tokens, :deleted_at, :datetime
  end
end

380:F:\git\coin\exchange\peatio-master\db\migrate\20141119155043_create_running_accounts.rb
class CreateRunningAccounts < ActiveRecord::Migration
  def change
    create_table :running_accounts do |t|
      t.integer :category
      t.decimal :income, precision: 32, scale: 16, null: false, default: 0
      t.decimal :expenses, precision: 32, scale: 16, null: false, default: 0
      t.integer :currency
      t.references :member, index: true
      t.references :source, polymorphic: true, index: true
      t.string :note

      t.timestamps
    end
  end
end

381:F:\git\coin\exchange\peatio-
master\db\migrate\20141203042029_rename_deposits_memo_to_confirmations.rb
class RenameDepositsMemoToConfirmations < ActiveRecord::Migration
  def up
    rename_column :deposits, :memo, :confirmations
  end

  def down
    rename_column :deposits, :confirmations, :memo
  end
end
```

382:F:\git\coin\exchange\peatio-master\db\migrate\20141216120736_add_trades_count_to_orders.rb

```
class AddTradesCountToOrders < ActiveRecord::Migration
  def change
    add_column :orders, :trades_count, :integer, default: 0
  end
end
```

383:F:\git\coin\exchange\peatio-master\db\migrate\20150117151634_add_signup_histories.rb

```
class AddSignupHistories < ActiveRecord::Migration
  def change
    create_table :signup_histories do |t|
      t.references :member, index: true
      t.string :ip
      t.string :accept_language
      t.string :ua
      t.datetime :created_at
    end
  end
end
```

384:F:\git\coin\exchange\peatio-master\db\migrate\20150205011423_add_account_id_index_on_account_versions.rb

```
class AddAccountIdIndexOnAccountVersions < ActiveRecord::Migration
  def change
    add_index :account_versions, :account_id
  end
end
```

385:F:\git\coin\exchange\peatio-master\db\migrate\20150405053726_add_default_withdraw_fund_source_id_to_accounts.rb

```
class AddDefaultWithdrawFundSourceIdToAccounts < ActiveRecord::Migration
  def change
    add_column :accounts, :default_withdraw_fund_source_id, :integer
  end
end
```

386:F:\git\coin\exchange\peatio-master\db\schema.rb

```
# encoding: UTF-8
# This file is auto-generated from the current state of the database. Instead
# of editing this file, please use the migrations feature of Active Record to
# incrementally modify your database, and then regenerate this schema definition.
```

```
#
# Note that this schema.rb definition is the authoritative source for your
# database schema. If you need to create the application database on another
# system, you should be using db:schema:load, not running all the migrations
# from scratch. The latter is a flawed and unsustainable approach (the more migrations
# you'll amass, the slower it'll run and the greater likelihood for issues).
#
# It's strongly recommended that you check this file into your version control system.

ActiveRecord::Schema.define(version: 20150405053726) do

  create_table "account_versions", force: true do |t|
    t.integer  "member_id"
    t.integer  "account_id"
    t.integer  "reason"
    t.decimal  "balance",         precision: 32, scale: 16
    t.decimal  "locked",          precision: 32, scale: 16
    t.decimal  "fee",             precision: 32, scale: 16
    t.decimal  "amount",          precision: 32, scale: 16
    t.integer  "modifiable_id"
    t.string   "modifiable_type"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.integer  "currency"
    t.integer  "fun"
  end

  add_index "account_versions", ["account_id", "reason"], name:
"index_account_versions_on_account_id_and_reason", using: :btree
  add_index "account_versions", ["account_id"], name: "index_account_versions_on_account_id",
using: :btree
  add_index "account_versions", ["member_id", "reason"], name:
"index_account_versions_on_member_id_and_reason", using: :btree
  add_index "account_versions", ["modifiable_id", "modifiable_type"], name:
"index_account_versions_on_modifiable_id_and_modifiable_type", using: :btree

  create_table "accounts", force: true do |t|
    t.integer  "member_id"
    t.integer  "currency"
    t.decimal  "balance",                         precision: 32, scale: 16
    t.decimal  "locked",                          precision: 32, scale: 16
    t.datetime "created_at"
    t.datetime "updated_at"
    t.decimal  "in",                              precision: 32, scale: 16
    t.decimal  "out",                             precision: 32, scale: 16
    t.integer  "default_withdraw_fund_source_id"
  end

  add_index "accounts", ["member_id", "currency"], name:
"index_accounts_on_member_id_and_currency", using: :btree
  add_index "accounts", ["member_id"], name: "index_accounts_on_member_id", using: :btree

  create_table "api_tokens", force: true do |t|
    t.integer  "member_id",                          null: false
    t.string   "access_key",            limit: 50, null: false
    t.string   "secret_key",            limit: 50, null: false
    t.datetime "created_at"
    t.datetime "updated_at"
    t.string   "trusted_ip_list"
    t.string   "label"
    t.integer  "oauth_access_token_id"
    t.datetime "expire_at"
    t.string   "scopes"
    t.datetime "deleted_at"
  end

  add_index "api_tokens", ["access_key"], name: "index_api_tokens_on_access_key", unique:
true, using: :btree
  add_index "api_tokens", ["secret_key"], name: "index_api_tokens_on_secret_key", unique: true,
using: :btree

  create_table "assets", force: true do |t|
    t.string  "type"
    t.integer "attachable_id"
    t.string  "attachable_type"
    t.string  "file"
  end

  create_table "audit_logs", force: true do |t|
    t.string   "type"
    t.integer  "operator_id"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.integer  "auditable_id"
    t.string   "auditable_type"
    t.string   "source_state"
    t.string   "target_state"
  end

  add_index "audit_logs", ["auditable_id", "auditable_type"], name:
"index_audit_logs_on_auditable_id_and_auditable_type", using: :btree
  add_index "audit_logs", ["operator_id"], name: "index_audit_logs_on_operator_id", using: :btree

  create_table "authentications", force: true do |t|
    t.string   "provider"
    t.string   "uid"
    t.string   "token"
    t.string   "secret"
    t.integer  "member_id"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.string   "nickname"
  end

  add_index "authentications", ["member_id"], name: "index_authentications_on_member_id",
using: :btree
  add_index "authentications", ["provider", "uid"], name:
"index_authentications_on_provider_and_uid", using: :btree

  create_table "comments", force: true do |t|
    t.text     "content"
    t.integer  "author_id"
    t.integer  "ticket_id"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "deposits", force: true do |t|
    t.integer  "account_id"
    t.integer  "member_id"
    t.integer  "currency"
    t.decimal  "amount",                 precision: 32, scale: 16
    t.decimal  "fee",                    precision: 32, scale: 16
    t.string   "fund_uid"
    t.string   "fund_extra"
    t.string   "txid"
    t.integer  "state"
    t.string   "aasm_state"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.datetime "done_at"
    t.string   "confirmations"
    t.string   "type"
    t.integer  "payment_transaction_id"
    t.integer  "txout"
  end

  add_index "deposits", ["txid", "txout"], name: "index_deposits_on_txid_and_txout", using: :btree

  create_table "document_translations", force: true do |t|
    t.integer  "document_id", null: false
    t.string   "locale",      null: false
    t.datetime "created_at"
    t.datetime "updated_at"
    t.string   "title"
    t.text     "body"
    t.text     "desc"
    t.text     "keywords"
  end

  add_index "document_translations", ["document_id"], name:
"index_document_translations_on_document_id", using: :btree
  add_index "document_translations", ["locale"], name: "index_document_translations_on_locale",
using: :btree

  create_table "documents", force: true do |t|
    t.string   "key"
    t.string   "title"
    t.text     "body"
    t.boolean  "is_auth"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.text     "desc"
    t.text     "keywords"
  end

  create_table "fund_sources", force: true do |t|
    t.integer  "member_id"
    t.integer  "currency"
    t.string   "extra"
    t.string   "uid"
    t.boolean  "is_locked",  default: false
    t.datetime "created_at"
    t.datetime "updated_at"
    t.datetime "deleted_at"
  end

  create_table "id_documents", force: true do |t|
    t.integer  "id_document_type"
    t.string   "name"
    t.string   "id_document_number"
    t.integer  "member_id"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.date     "birth_date"
    t.text     "address"
    t.string   "city"
    t.string   "country"
    t.string   "zipcode"
    t.integer  "id_bill_type"
    t.string   "aasm_state"
  end

  create_table "identities", force: true do |t|
    t.string   "email"
    t.string   "password_digest"
    t.boolean  "is_active"
    t.integer  "retry_count"
    t.boolean  "is_locked"
    t.datetime "locked_at"
    t.datetime "last_verify_at"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "members", force: true do |t|
    t.string   "sn"
    t.string   "display_name"
    t.string   "email"
    t.integer  "identity_id"
```

```
    t.datetime "created_at"
    t.datetime "updated_at"
    t.integer  "state"
    t.boolean  "activated"
    t.integer  "country_code"
    t.string   "phone_number"
    t.boolean  "disabled",     default: false
    t.boolean  "api_disabled", default: false
    t.string   "nickname"
  end

  create_table "oauth_access_grants", force: true do |t|
    t.integer  "resource_owner_id", null: false
    t.integer  "application_id",    null: false
    t.string   "token",             null: false
    t.integer  "expires_in",        null: false
    t.text     "redirect_uri",      null: false
    t.datetime "created_at",        null: false
    t.datetime "revoked_at"
    t.string   "scopes"
  end

  add_index "oauth_access_grants", ["token"], name: "index_oauth_access_grants_on_token",
unique: true, using: :btree

  create_table "oauth_access_tokens", force: true do |t|
    t.integer  "resource_owner_id"
    t.integer  "application_id"
    t.string   "token",             null: false
    t.string   "refresh_token"
    t.integer  "expires_in"
    t.datetime "revoked_at"
    t.datetime "created_at",        null: false
    t.string   "scopes"
    t.datetime "deleted_at"
  end

  add_index "oauth_access_tokens", ["refresh_token"], name:
"index_oauth_access_tokens_on_refresh_token", unique: true, using: :btree
  add_index "oauth_access_tokens", ["resource_owner_id"], name:
"index_oauth_access_tokens_on_resource_owner_id", using: :btree
  add_index "oauth_access_tokens", ["token"], name: "index_oauth_access_tokens_on_token",
unique: true, using: :btree

  create_table "oauth_applications", force: true do |t|
    t.string   "name",         null: false
    t.string   "uid",          null: false
    t.string   "secret",       null: false
    t.text     "redirect_uri", null: false
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  add_index "oauth_applications", ["uid"], name: "index_oauth_applications_on_uid", unique: true,
using: :btree

  create_table "orders", force: true do |t|
    t.integer  "bid"
    t.integer  "ask"
    t.integer  "currency"
    t.decimal  "price",                    precision: 32, scale: 16
    t.decimal  "volume",                   precision: 32, scale: 16
    t.decimal  "origin_volume",            precision: 32, scale: 16
    t.integer  "state"
    t.datetime "done_at"
    t.string   "type",          limit: 8
    t.integer  "member_id"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.string   "sn"
    t.string   "source",                                                       null: false
    t.string   "ord_type",      limit: 10
    t.decimal  "locked",                   precision: 32, scale: 16
    t.decimal  "origin_locked",            precision: 32, scale: 16
    t.decimal  "funds_received",           precision: 32, scale: 16, default: 0.0
    t.integer  "trades_count",                                       default: 0
  end

  add_index "orders", ["currency", "state"], name: "index_orders_on_currency_and_state", using:
:btree
  add_index "orders", ["member_id", "state"], name: "index_orders_on_member_id_and_state",
using: :btree
  add_index "orders", ["member_id"], name: "index_orders_on_member_id", using: :btree
  add_index "orders", ["state"], name: "index_orders_on_state", using: :btree
  create_table "partial_trees", force: true do |t|
    t.integer  "proof_id",   null: false
    t.integer  "account_id", null: false
    t.text     "json",       null: false
    t.datetime "created_at"
    t.datetime "updated_at"
    t.string   "sum"
  end

  create_table "payment_addresses", force: true do |t|
    t.integer  "account_id"
    t.string   "address"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.integer  "currency"
  end

  create_table "payment_transactions", force: true do |t|
    t.string   "txid"
    t.decimal  "amount",                   precision: 32, scale: 16
    t.integer  "confirmations"
    t.string   "address"
    t.integer  "state"
    t.string   "aasm_state"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.datetime "receive_at"
    t.datetime "dont_at"
    t.integer  "currency"
    t.string   "type",          limit: 60
    t.integer  "txout"
  end

  add_index "payment_transactions", ["txid", "txout"], name:
"index_payment_transactions_on_txid_and_txout", using: :btree
  add_index "payment_transactions", ["type"], name: "index_payment_transactions_on_type",
using: :btree

  create_table "proofs", force: true do |t|
    t.string   "root"
    t.integer  "currency"
    t.boolean  "ready",                default: false
    t.datetime "created_at"
    t.datetime "updated_at"
    t.string   "sum"
    t.text     "addresses"
    t.string   "balance",    limit: 30
  end

  create_table "read_marks", force: true do |t|
    t.integer  "readable_id"
    t.integer  "member_id",                null: false
    t.string   "readable_type", limit: 20, null: false
    t.datetime "timestamp"
  end

  add_index "read_marks", ["member_id"], name: "index_read_marks_on_member_id", using:
:btree
  add_index "read_marks", ["readable_type", "readable_id"], name:
"index_read_marks_on_readable_type_and_readable_id", using: :btree

  create_table "running_accounts", force: true do |t|
    t.integer  "category"
    t.decimal  "income",      precision: 32, scale: 16, default: 0.0, null: false
    t.decimal  "expenses",    precision: 32, scale: 16, default: 0.0, null: false
    t.integer  "currency"
    t.integer  "member_id"
    t.integer  "source_id"
    t.string   "source_type"
    t.string   "note"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  add_index "running_accounts", ["member_id"], name: "index_running_accounts_on_member_id",
using: :btree
  add_index "running_accounts", ["source_id", "source_type"], name:
"index_running_accounts_on_source_id_and_source_type", using: :btree

  create_table "signup_histories", force: true do |t|
    t.integer  "member_id"
    t.string   "ip"
    t.string   "accept_language"
    t.string   "ua"
    t.datetime "created_at"
  end

  add_index "signup_histories", ["member_id"], name: "index_signup_histories_on_member_id",
using: :btree

  create_table "simple_captcha_data", force: true do |t|
    t.string   "key",        limit: 40
    t.string   "value",      limit: 6
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  add_index "simple_captcha_data", ["key"], name: "idx_key", using: :btree

  create_table "taggings", force: true do |t|
    t.integer  "tag_id"
    t.integer  "taggable_id"
    t.string   "taggable_type"
    t.integer  "tagger_id"
    t.string   "tagger_type"
    t.string   "context",       limit: 128
    t.datetime "created_at"
  end

  add_index "taggings", ["tag_id", "taggable_id", "taggable_type", "context", "tagger_id",
"tagger_type"], name: "taggings_idx", unique: true, using: :btree

  create_table "tags", force: true do |t|
    t.string "name"
  end

  add_index "tags", ["name"], name: "index_tags_on_name", unique: true, using: :btree

  create_table "tickets", force: true do |t|
    t.string   "title"
    t.text     "content"
    t.string   "aasm_state"
    t.integer  "author_id"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "tokens", force: true do |t|
    t.string   "token"
    t.datetime "expire_at"
    t.integer  "member_id"
    t.boolean  "is_used",    default: false
    t.string   "type"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  add_index "tokens", ["type", "token", "expire_at", "is_used"], name:
"index_tokens_on_type_and_token_and_expire_at_and_is_used", using: :btree

  create_table "trades", force: true do |t|
    t.decimal  "price",         precision: 32, scale: 16
    t.decimal  "volume",        precision: 32, scale: 16
    t.integer  "ask_id"
    t.integer  "bid_id"
    t.integer  "trend"
    t.integer  "currency"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.integer  "ask_member_id"
    t.integer  "bid_member_id"
    t.decimal  "funds",         precision: 32, scale: 16
  end

  add_index "trades", ["ask_id"], name: "index_trades_on_ask_id", using: :btree
  add_index "trades", ["ask_member_id"], name: "index_trades_on_ask_member_id", using: :btree
  add_index "trades", ["bid_id"], name: "index_trades_on_bid_id", using: :btree
  add_index "trades", ["bid_member_id"], name: "index_trades_on_bid_member_id", using: :btree
  add_index "trades", ["created_at"], name: "index_trades_on_created_at", using: :btree
  add_index "trades", ["currency"], name: "index_trades_on_currency", using: :btree

  create_table "two_factors", force: true do |t|
    t.integer  "member_id"
    t.string   "otp_secret"
    t.datetime "last_verify_at"
    t.boolean  "activated"
    t.string   "type"
    t.datetime "refreshed_at"
  end

  create_table "versions", force: true do |t|
    t.string   "item_type",  null: false
    t.integer  "item_id",    null: false
    t.string   "event",      null: false
    t.string   "whodunnit"
    t.text     "object"
    t.datetime "created_at"
  end

  add_index "versions", ["item_type", "item_id"], name:
"index_versions_on_item_type_and_item_id", using: :btree

  create_table "withdraws", force: true do |t|
    t.string   "sn"
    t.integer  "account_id"
    t.integer  "member_id"
    t.integer  "currency"
    t.decimal  "amount",     precision: 32, scale: 16
    t.decimal  "fee",        precision: 32, scale: 16
    t.string   "fund_uid"
    t.string   "fund_extra"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.datetime "done_at"
    t.string   "txid"
    t.string   "aasm_state"
    t.decimal  "sum",        precision: 32, scale: 16, default: 0.0, null: false
    t.string   "type"
  end

end

387:F:\git\coin\exchange\peatio-master\db\seeds.rb
ADMIN_EMAIL = 'admin@peatio.dev'
ADMIN_PASSWORD = 'Pass@word8'

admin_identity = Identity.find_or_create_by(email: ADMIN_EMAIL)
admin_identity.password = admin_identity.password_confirmation = ADMIN_PASSWORD
admin_identity.is_active = true
admin_identity.save!

admin_member = Member.find_or_create_by(email: ADMIN_EMAIL)
admin_member.authentications.build(provider: 'identity', uid: admin_identity.id)
admin_member.save!

if Rails.env == 'development'
  NORMAL_PASSWORD = 'Pass@word8'

  foo = Identity.create(email: 'foo@peatio.dev', password: NORMAL_PASSWORD,
password_confirmation: NORMAL_PASSWORD, is_active: true)
  foo_member = Member.create(email: foo.email)
  foo_member.authentications.build(provider: 'identity', uid: foo.id)
  foo_member.tag_list.add 'vip'
  foo_member.tag_list.add 'hero'
  foo_member.save

  bar = Identity.create(email: 'bar@peatio.dev', password: NORMAL_PASSWORD,
password_confirmation: NORMAL_PASSWORD, is_active: true)
  bar_member = Member.create(email: bar.email)
  bar_member.authentications.build(provider: 'identity', uid: bar.id)
  bar_member.tag_list.add 'vip'
  bar_member.tag_list.add 'hero'
  bar_member.save
end

388:F:\git\coin\exchange\peatio-master\lib\aasm\locking.rb
module AASM::Locking
  def aasm_write_state(state)
    lock!
    super(state)
  end
end

389:F:\git\coin\exchange\peatio-master\lib\benchmark\amqp_mock.rb
class AMQPQueue
  class <<self
    def queues
      @queues ||= Hash.new {|h, k| h[k] = [] }
    end

    def enqueue(qid, payload)
```

```
      queues[qid] << payload
    end

    def publish(eid, payload, attrs={})
      # do nothing
    end
  end
end
```

390:F:\git\coin\exchange\peatio-master\lib\benchmark\execution.rb

```
require_relative 'amqp_mock'

module Benchmark
  class Execution < Matching

    def initialize(label, num, round, process_num)
      super(label, num, round)
      @process_num = process_num
    end

    def collect_time
      time = Dir[Rails.root.join('tmp', 'concurrent_executor_*')].map do |f|
        File.open(f, 'r') {|ff| ff.read.to_f }
      end.max
      puts "elapsed: #{time}"
      Benchmark::Tms.new(0, 0, 0, 0, time)
    end

    def execute_trades
      t1 = Trade.count

      @instructions.in_groups(@process_num, false).each_with_index do |insts, i|
        unless Process.fork
          ActiveRecord::Base.connection.reconnect!
          puts "Executor #{i+1} started."

          t1 = Time.now
          insts.each do |payload|
            ::Matching::Executor.new(payload).execute!
          end
          elapsed = Time.now - t1
          File.open(Rails.root.join('tmp', "concurrent_executor_#{i+1}"), 'w') {|f| f.write(elapsed.to_f) }

          puts "Executor #{i+1} finished work, stop."
          exit 0
        end
      end
      pid_and_status = Process.waitall

      ActiveRecord::Base.connection.reconnect!
      @trades = Trade.count - t1

      collect_time
    end

    def run_execute_trades
      puts "\n>> Execute Trade Instructions"
      Benchmark.benchmark(Benchmark::CAPTION, 20, Benchmark::FORMAT) do |x|
        @times[:execution] = [ execute_trades ]
        puts "#{@instructions.size} trade instructions executed by #{@process_num} executors,
#{@trades} trade created."
      end
    end

  end
end

391:F:\git\coin\exchange\peatio-master\lib\benchmark\helpers.rb
module Benchmark
  module Helpers

    def create_members
      @members = {ask: [], bid: []}

      (@num/2).times do
        @members[:ask] << SweatFactory.make_member
        @members[:bid] << SweatFactory.make_member
      end
    end

    def lock_funds
      @members[:ask].each do |m|
        m.get_account(:btc).update_attributes(locked: 100)
      end
      @members[:bid].each do |m|
        m.get_account(:cny).update_attributes(locked: 1000000)
      end
    end

    def create_orders
      @orders = []

      price_and_volume = []
      (@num/2).times do
        price = 3000+rand(3000)
        volume = 1+rand(10)
        price_and_volume << [price, volume]
      end

      # Create asks and bids seperately, so asks will accumulate in memory before get matched
      @members[:ask].each_with_index do |m, i|
        price, volume = price_and_volume[i]
        o = SweatFactory.make_order(OrderAsk, volume: volume, price: price, member: m)
        o.save!
        @orders << o
      end
      @members[:bid].each_with_index do |m, i|
        price, volume = price_and_volume[i]
        o = SweatFactory.make_order(OrderBid, volume: volume, price: price, member: m)
        o.save!
        @orders << o
      end
    end

    def matching_orders
      matches = 0
      instructions = []

      worker = Worker::Matching.new

      @processed = Order.active.count
      Order.active.each do |order|
        worker.process({action: 'submit', order: order.to_matching_attributes}, {}, {})
      end
```

```
      @instructions = AMQPQueue.queues[:trade_executor]
      @matches      = @instructions.size
    end

    def execute_trades
      t1 = Trade.count

      @instructions.each do |payload|
        ::Matching::Executor.new(payload).execute!
      end

      @trades = Trade.count - t1
    end

  end
end

392:F:\git\coin\exchange\peatio-master\lib\benchmark\integration.rb
module Benchmark
  class Integration
    include Helpers

    def initialize(num)
      @num = num
    end

    def run
      Benchmark.benchmark(Benchmark::CAPTION, 20, Benchmark::FORMAT) do |x|
        x.report("create members") { create_members }
        x.report("lock funds")     { lock_funds }
        x.report("create orders")  { create_orders }
      end

      Signal.trap("INT") do
        AMQPQueue.channel.work_pool.kill
        puts "\nFinished."
      end

      t1 = Time.now
      count = 0
      AMQPQueue.channel.queue('', auto_delete: 
true).bind(AMQPQueue.exchange(:trade)).subscribe do |info, what, payload|
        t = Time.now - t1
        count += 1
        orate = "%.2f" % (@num.to_f/t)
        trate = "%.2f" % (count.to_f/t)
        print "\rTime elapsed: #{t}s   Orders: total #{@num}, rate #{orate}o/s   Trades: total #{count},
rate #{trate}t/s          "
      end

      @orders.each do |o|
        AMQPQueue.enqueue(:matching, action: 'submit', order: o.to_matching_attributes)
      end

      AMQPQueue.channel.work_pool.join
    end

  end
end

393:F:\git\coin\exchange\peatio-master\lib\benchmark\matching.rb
require_relative 'amqp_mock'

module Benchmark
  class Matching
    include Helpers

    def initialize(label, num, round)
      @label = label.to_s
      @num = num
      @round = round
      @times = Hash.new {|h,k| h[k] = [] }
    end

    def run
      run_prepare_orders
      run_matching_orders
      run_execute_trades
      save
    end

    def run_prepare_orders
      (1..@round).map do |i|
        puts "\n>> Round #{i}"
        Benchmark.benchmark(Benchmark::CAPTION, 20, Benchmark::FORMAT) do |x|
          @times[:create_members] << x.report("create members") { create_members }
          @times[:lock_funds]     << x.report("lock funds") { lock_funds }
          @times[:create_orders]  << x.report("create orders") { create_orders }
          nil
        end
      end
    end

    def run_matching_orders
      puts "\n>> Match Them All"
      Benchmark.benchmark(Benchmark::CAPTION, 20, Benchmark::FORMAT) do |x|
        t = x.report { matching_orders }
        @times[:matching] = [t]
        puts "#{@matches} matches run for #{@processed} orders, #{@instructions.size} trade
instruction generated."
      end
    end

    def run_execute_trades
      puts "\n>> Execute Trade Instructions"
      Benchmark.benchmark(Benchmark::CAPTION, 20, Benchmark::FORMAT) do |x|
        t = x.report { execute_trades }
        @times[:execution] = [t]
        puts "#{@instructions.size} trade instructions executed, #{@trades} trade created."
      end
    end

    def save
      avg = {}

      File.open(Rails.root.join('tmp', "matching_result_#{@label}"), 'w') do |f|
        @times.each do |k, v|
          avg[k] = averages(v)
          f.puts avg[k].join(" ")
        end
      end

      puts "\n>> Average throughput (ops: orders per second, eps: execution per second)"
      puts "create members: %.2fops" % [@num/avg[:create_members].last]
      puts "lock funds:     %.2fops" % [@num/avg[:lock_funds].last]
      puts "create orders:  %.2fops" % [@num/avg[:create_orders].last]
```

```
      puts "submit orders:  %.2fops" % [@num/(avg[:lock_funds].last+avg[:create_orders].last)]
      puts "matching:       %.2fops" % [@processed/avg[:matching].last] if avg[:matching]
      puts "execution:      %.2feps" % [@instructions.size/avg[:execution].last] if avg[:execution]
      puts "* submit order = lock funds + create order"
    end

    def averages(times)
      utime_avg = times.map(&:utime).sum / times.size
      stime_avg = times.map(&:stime).sum / times.size
      real_avg  = times.map(&:real).sum  / times.size
      [utime_avg, stime_avg, real_avg]
    end

  end
end
```

394:F:\git\coin\exchange\peatio-master\lib\benchmark\sweat_factory.rb

```
module Benchmark
  class SweatFactory

    @@seq = 0

    class <<self
      def make_member
        @@seq += 1
        member = Member.create!(
          email: "user#{@@seq}@example.com",
          name: "Matching Benchmark #{@@seq}"
        )
      end

      def make_order(klass, attrs={})
        klass.new({
          bid: :cny,
          ask: :btc,
          state: Order::WAIT,
          currency: :btccny,
          origin_volume: attrs[:volume],
          source: 'Web'
        }.merge(attrs))
      end
    end
  end
end

395:F:\git\coin\exchange\peatio-master\lib\daemons\amqp_daemon.rb
#!/usr/bin/env ruby

# You might want to change this
ENV["RAILS_ENV"] ||= "development"

root = File.expand_path(File.dirname(__FILE__))
root = File.dirname(root) until File.exists?(File.join(root, 'config'))
Dir.chdir(root)

require File.join(root, "config", "environment")

raise "bindings must be provided." if ARGV.size == 0

Rails.logger = logger = Logger.new STDOUT

conn = Bunny.new AMQPConfig.connect
conn.start

ch = conn.create_channel
id = $0.split(':')[2]
prefetch = AMQPConfig.channel(id)[:prefetch] || 0
ch.prefetch(prefetch) if prefetch > 0
logger.info "Connected to AMQP broker (prefetch: #{prefetch > 0 ? prefetch : 'default'})"

terminate = proc do
  # logger is forbidden in signal handling, just use puts here
  puts "Terminating threads .."
  ch.work_pool.kill
  puts "Stopped."
end
Signal.trap("INT",  &terminate)
Signal.trap("TERM", &terminate)

workers = []
ARGV.each do |id|
  worker = AMQPConfig.binding_worker(id)
  queue  = ch.queue *AMQPConfig.binding_queue(id)
  if args = AMQPConfig.binding_exchange(id)
    x = ch.send *args

    case args.first
    when 'direct'
      queue.bind x, routing_key: AMQPConfig.routing_key(id)
    when 'topic'
      AMQPConfig.topics(id).each do |topic|
        queue.bind x, routing_key: topic
      end
    else
      queue.bind x
    end
  end

  clean_start = AMQPConfig.data[:binding][id][:clean_start]
  queue.purge if clean_start

  manual_ack  = AMQPConfig.data[:binding][id][:manual_ack]
  queue.subscribe(manual_ack: manual_ack) do |delivery_info, metadata, payload|
    logger.info "Received: #{payload}"
    begin
      worker.process JSON.parse(payload), metadata, delivery_info
    rescue Exception => e
      logger.fatal e
      logger.fatal e.backtrace.join("\n")
    ensure
      ch.ack(delivery_info.delivery_tag) if manual_ack
    end
  end

  workers << worker
end

%w(USR1 USR2).each do |signal|
  Signal.trap(signal) do
    puts "#{signal} received."
    handler = "on_#{signal.downcase}"
    workers.each {|w| w.send handler if w.respond_to?(handler) }
  end
end
```

```
ch.work_pool.join

396:F:\git\coin\exchange\peatio-master\lib\daemons\global_state.rb
#!/usr/bin/env ruby

ENV["RAILS_ENV"] ||= "development"

root = File.expand_path(File.dirname(__FILE__))
root = File.dirname(root) until File.exists?(File.join(root, 'config'))
Dir.chdir(root)

require File.join(root, "config", "environment")

$running = true
Signal.trap("TERM") do
  $running = false
end

while($running) do
  all_tickers = {}
  Market.all.each do |market|
    global = Global[market.id]
    global.trigger_orderbook
    all_tickers[market.id] = market.unit_info.merge(global.ticker)
  end
  Global.trigger 'tickers', all_tickers

  sleep 3
end

397:F:\git\coin\exchange\peatio-master\lib\daemons\hot_wallets.rb
#!/usr/bin/env ruby

# You might want to change this
ENV["RAILS_ENV"] ||= "development"

root = File.expand_path(File.dirname(__FILE__))
root = File.dirname(root) until File.exists?(File.join(root, 'config'))
Dir.chdir(root)

require File.join(root, "config", "environment")
```

```
$running = true
Signal.trap("TERM") do
  $running = false
end

while($running) do
  Currency.all.each do |currency|
    currency.refresh_balance if currency.coin?
  end

  sleep 5
end
```

398:F:\git\coin\exchange\peatio-master\lib\daemons\k.rb

```
#!/usr/bin/env ruby

ENV["RAILS_ENV"] ||= "development"

root = File.expand_path(File.dirname(__FILE__))
root = File.dirname(root) until File.exists?(File.join(root, 'config'))
Dir.chdir(root)

require File.join(root, "config", "environment")

Rails.logger = @logger = Logger.new STDOUT

@r ||= KlineDB.redis

$running = true
Signal.trap("TERM") do
  $running = false
end



def key(market, period = 1)
  "peatio:#{market}:k:#{period}"
end

def last_ts(market, period = 1)
  latest = @r.lindex key(market, period), -1
  latest && Time.at(JSON.parse(latest)[0])
```

```
end

def next_ts(market, period = 1)
  if ts = last_ts(market, period)
    ts += period.minutes
  else
    if first_trade = Trade.with_currency(market).first
      ts = Trade.with_currency(market).first.created_at.to_i
      period == 10080 ? Time.at(ts).beginning_of_week : Time.at(ts -  ts % (period * 60))
    end
  end
end

def _k1_set(market, start, period)
  ts = JSON.parse(@r.lindex(key(market, 1), 0)).first

  left = offset = (start.to_i - ts) / 60
  left = 0 if left < 0

  right = offset + period - 1

  right < 0 ? [] : @r.lrange(key(market, 1), left, right).map{|str| JSON.parse(str)}
end

def k1(market, start)
  trades = Trade.with_currency(market).where('created_at >= ? AND created_at < ?', start,
1.minutes.since(start)).pluck(:price, :volume)
  return nil if trades.count == 0

  prices, volumes = trades.transpose
  [start.to_i, prices.first.to_f, prices.max.to_f, prices.min.to_f, prices.last.to_f,
volumes.sum.to_f.round(4)]
end

def kn(market, start, period = 5)
  arr = _k1_set(market, start, period)
  return nil if arr.empty?

  _, _, high, low, _, volumes = arr.transpose
  [start.to_i, arr.first[1], high.max, low.min, arr.last[4], volumes.sum.round(4)]
end
```

```ruby
def get_point(market, period, ts)
  point = period == 1 ? k1(market, ts) : kn(market, ts, period)

  if point.nil?
    point = JSON.parse @r.lindex(key(market, period), -1)
    point = [ts.to_i, point[4], point[4], point[4], point[4], 0]
  end

  point
end

def append_point(market, period, ts)
  k = key(market, period)
  point = get_point(market, period, ts)

  @logger.info "append #{k}: #{point.to_json}"
  @r.rpush k, point.to_json

  if period == 1
    # 24*60 = 1440
    if point = @r.lindex(key(market, period), -1441)
      Rails.cache.write "peatio:#{market}:ticker:open", JSON.parse(point)[4]
    end
  end
end

def update_point(market, period, ts)
  k = key(market, period)
  point = get_point(market, period, ts)

  @logger.info "update #{k}: #{point.to_json}"
  @r.rpop k
  @r.rpush k, point.to_json
end

def fill(market, period = 1)
  ts = next_ts(market, period)

  # 30 seconds is a protect buffer to allow update_point to update the previous
  # period one last time, after the previous period passed. After the protect
  # buffer a new point of current period will be created, the previous point
  # is freezed.
  #
  # The protect buffer also allows MySQL slave have enough time to sync data.
  while (ts + 30.seconds) <= Time.now
    append_point(market, period, ts)
    ts = next_ts(market, period)
  end

  update_point(market, period, last_ts(market, period))
end

while($running) do
  Market.all.each do |market|
    ts = next_ts(market.id, 1)
    next unless ts

    [1, 5, 15, 30, 60, 120, 240, 360, 720, 1440, 4320, 10080].each do |period|
      fill(market.id, period)
    end
  end

  sleep 15
end

399:F:\git\coin\exchange\peatio-master\lib\daemons\payment_transaction.rb
#!/usr/bin/env ruby

# You might want to change this
ENV["RAILS_ENV"] ||= "development"

root = File.expand_path(File.dirname(__FILE__))
root = File.dirname(root) until File.exists?(File.join(root, 'config'))
Dir.chdir(root)

require File.join(root, "config", "environment")

$running = true
Signal.trap("TERM") do
  $running = false
end

while($running) do
  PaymentTransaction::Normal.with_aasm_state(:unconfirm, :confirming).each do |tx|
    begin
      tx.with_lock do
        tx.check!
      end
    rescue
      puts "Error on PaymentTransaction::Normal: #{$!}"
      puts $!.backtrace.join("\n")
      next
    end
  end

  sleep 5
end
```

400:F:\git\coin\exchange\peatio-master\lib\daemons\stats.rb

```
#!/usr/bin/env ruby

ENV["RAILS_ENV"] ||= "development"

root = File.expand_path(File.dirname(__FILE__))
root = File.dirname(root) until File.exists?(File.join(root, 'config'))
Dir.chdir(root)

require File.join(root, "config", "environment")

Rails.logger = @logger = Logger.new STDOUT


$running = true
Signal.trap("TERM") do
  $running = false
end

workers = []
workers << Worker::MemberStats.new
Currency.all.each do |currency|
  workers << Worker::FundStats.new(currency)
  workers << Worker::WalletStats.new(currency)
end
Market.all.each do |market|
  workers << Worker::TradeStats.new(market)
  workers << Worker::TopStats.new(market)
```

```
end

while($running) do
  workers.each do |worker|
    begin
      worker.run
    rescue
      Rails.logger.error "#{worker.class.name} failed to run: #{$!}"
      Rails.logger.error $!.backtrace[0,20].join("\n")
    end
  end

  sleep 30
end

401:F:\git\coin\exchange\peatio-master\lib\daemons\websocket_api.rb
#!/usr/bin/env ruby

# You might want to change this
ENV["RAILS_ENV"] ||= "development"

root = File.expand_path(File.dirname(__FILE__))
root = File.dirname(root) until File.exists?(File.join(root, 'config'))
Dir.chdir(root)

#require 'em-synchrony'
#require 'em-synchrony/mysql2'
#require 'em-synchrony/activerecord'

require 'socket'
require File.join(root, "config", "environment")

#db_config = Rails.configuration.database_configuration[Rails.env].merge(
  #'adapter' => 'em_mysql2'
#)
#ActiveRecord::Base.establish_connection(db_config)

Rails.logger = logger = Logger.new STDOUT

EM.error_handler do |e|
  logger.error "Error: #{e}"
  logger.error e.backtrace[0,20].join("\n")
```

```ruby
end

EM.run do
  conn = AMQP.connect AMQPConfig.connect
  logger.info "Connected to AMQP broker."

  ch = AMQP::Channel.new conn
  ch.prefetch(1)

  config = {host: ENV['WEBSOCKET_HOST'], port: ENV['WEBSOCKET_PORT']}
  if ENV['WEBSOCKET_SSL_KEY'] && ENV['WEBSOCKET_SSL_CERT']
    config[:secure] = true
    config[:tls_options] = {
      private_key_file: Rails.root.join(ENV['WEBSOCKET_SSL_KEY']).to_s,
      cert_chain_file: Rails.root.join(ENV['WEBSOCKET_SSL_CERT']).to_s
    }
  end

  EM::WebSocket.run(config) do |ws|
    logger.debug "New WebSocket connection: #{ws.inspect}"

    protocol = ::APIv2::WebSocketProtocol.new(ws, ch, logger)

    ws.onopen do
      if ws.pingable?
        port, ip = Socket.unpack_sockaddr_in(ws.get_peername)

        EM.add_periodic_timer 10 do
          ws.ping "#{ip}:#{port}"
        end

        ws.onpong do |message|
          logger.debug "pong: #{message}"
        end
      end

      protocol.challenge
    end

    ws.onmessage do |message|
      protocol.handle message
    end
    ws.onerror do |error|
      case error
      when EM::WebSocket::WebSocketError
        logger.info "WebSocket error: #{$!}"
        logger.info $!.backtrace[0,20].join("\n")
        logger.info $!.inspect
      else
        logger.info $!
      end
    end

    ws.onclose do
      logger.info "WebSocket closed"
    end
  end
end
```

402:F:\git\coin\exchange\peatio-master\lib\daemons\withdraw_audit.rb

```
#!/usr/bin/env ruby

ENV["RAILS_ENV"] ||= "development"

root = File.expand_path(File.dirname(__FILE__))
root = File.dirname(root) until File.exists?(File.join(root, 'config'))
Dir.chdir(root)

require File.join(root, "config", "environment")

$running = true
Signal.trap("TERM") do
  $running = false
end

while($running) do
  Withdraw.submitted.each do |withdraw|
    begin
      withdraw.audit!
    rescue
      puts "Error on withdraw audit: #{$!}"
      puts $!.backtrace.join("\n")
    end
  end

  sleep 5
end

403:F:\git\coin\exchange\peatio-master\lib\datagrid\column_i18n.rb
module Datagrid
  module ColumnI18n
    extend ActiveSupport::Concern

    module ClassMethods
      def column_localtime(name, options = {}, &block)
        column(name, options) do |model|
          val = block ? block.call(model) : model.send(name)
          if options[:i18n]
            I18n.l(val.localtime, options[:i18n])
          else
            I18n.l(val.localtime)
          end
        end
      end

      def column_i18n(name, options = {}, &block)
        column(name, options) do |model|
          val = block ? block.call(model) : model.send(name)
          if options[:i18n]
            I18n.l(val, options[:i18n])
          else
            I18n.l(val)
          end
        end
      end
    end
  end
end

404:F:\git\coin\exchange\peatio-master\lib\datagrid\naming.rb
module Datagrid
  module Naming
    extend ActiveSupport::Concern
    extend ::ActiveModel::Naming
    module ClassMethods
      def grid_name
        I18n.t("activerecord.models.#{model_name.i18n_key}")
      end
    end
  end
end
```

405:F:\git\coin\exchange\peatio-master\lib\doorkeeper\access_token.rb

```
# Extend Doorkeeper::AccessToken to add a new access token type:
#   urn:peatio:api:v2:token
#
# This type will return APIv2 token in format "<access_key>:<secret_key>", then
# users can authenticate themselves using the keys and APIv2 authentication
# protocol.

module Doorkeeper
  class AccessToken
    paranoid

    attr_accessor :api_token

    after_create :link_api_token

    def token_type
      'urn:peatio:api:v2:token'
    end

    def revoke(clock = DateTime)
      super

      self.api_token = APIToken.from_oauth_token(self)
      api_token.try(:destroy)
    end

    private

    def generate_token
      requsted_scopes = scopes.to_s
      raise "Invalid scope: #{requsted_scopes}" if requsted_scopes == 'all'
      member         = Member.find resource_owner_id
      self.api_token = member.api_tokens.create!(label: application.name, scopes:
requsted_scopes)
      self.token     = api_token.to_oauth_token
    end

    def link_api_token
      api_token.update_attributes(oauth_access_token_id: id, expire_at: expired_time)
    end

  end
end

406:F:\git\coin\exchange\peatio-master\lib\extras\simple_form_extensions.rb
module SearchButton
  def search_button(*args, &block)
    template.content_tag :div, :class => "form-group" do
      template.content_tag :div, :class => "form-submit" do
        submit(*args)
      end
    end
  end
end

module WrappedButton

  def wrapped_button(*args, &block)
    template.content_tag :div, :class => "form-group" do
      template.content_tag :div, :class => "form-submit col-xs-22" do

        options = args.extract_options!
        loading = self.object.new_record? ? I18n.t('simple_form.creating') :
I18n.t('simple_form.updating')
        options[:"data-loading-text"] = [loading, options[:"data-loading-text"]].compact
        options[:class] = ['btn btn-default btn-lg pull-right', options[:class]].compact

        args << options

        block_view = block ? template.capture(&block) : nil
        submit_view = options.delete(:no_submit) ? nil : submit(*args)

        cancel_view =
          if cancel_link = options.delete(:cancel)
            class_text = 'btn btn-info btn-lg pull-right'
            cancel_text = options.delete(:cancel_text) || I18n.t('simple_form.buttons.cancel')
            template.link_to(cancel_text, cancel_link, class: class_text)
          end

        [submit_view, cancel_view, block_view].join.html_safe
      end
    end
  end
end
SimpleForm::FormBuilder.send :include, SearchButton
SimpleForm::FormBuilder.send :include, WrappedButton
```

407:F:\git\coin\exchange\peatio-master\lib\generators\deposit\deposit_generator.rb

```
class DepositGenerator < Rails::Generators::NamedBase
  source_root File.expand_path('../templates', __FILE__)
  argument :code, :type => :string
  argument :symbol, :type => :string, :default => '#'

  def copy_initializer_file
    template "model.rb.erb", "app/models/deposits/#{name.underscore}.rb"
    template "controller.rb.erb",
"app/controllers/private/deposits/#{name.underscore.pluralize}_controller.rb"
    template "locales/zh-CN.yml.erb", "config/locales/deposits/#{name.underscore.pluralize}/zh-
CN.yml"
    template "locales/en.yml.erb", "config/locales/deposits/#{name.underscore.pluralize}/en.yml"
    template "views/new.html.slim.erb",
"app/views/private/deposits/#{name.underscore.pluralize}/new.slim"
  end
end
```

408:F:\git\coin\exchange\peatio-master\lib\generators\withdraw\withdraw_generator.rb

```
class WithdrawGenerator < Rails::Generators::NamedBase
  source_root File.expand_path('../templates', __FILE__)
  argument :code, :type => :string
  argument :symbol, :type => :string, :default => '#'

  def copy_initializer_file
    template "model.rb.erb", "app/models/withdraws/#{name.underscore}.rb"
    template "controller.rb.erb",
"app/controllers/private/withdraws/#{name.underscore.pluralize}_controller.rb"
```

```
    template "locales/zh-CN.yml.erb", "config/locales/withdraws/#{name.underscore.pluralize}/zh-
CN.yml"
    template "locales/en.yml.erb", "config/locales/withdraws/#{name.underscore.pluralize}/en.yml"
    template "views/new.html.slim.erb",
"app/views/private/withdraws/#{name.underscore.pluralize}/new.slim"
    template "views/edit.html.slim.erb",
"app/views/private/withdraws/#{name.underscore.pluralize}/edit.slim"
  end
end

409:F:\git\coin\exchange\peatio-master\lib\js_locale_helper.rb
module JsLocaleHelper

  def self.load_yaml(locale)
    locale_str = locale.to_s
    trans        =
YAML::load(File.open("#{Rails.root}/config/locales/client.#{locale_str}.yml"))[locale_str]['js']
    custom_trans =
YAML::load(File.open("#{Rails.root}/config/locales/custom/client.#{locale_str}.yml"))[locale_str]['js']
    {locale_str => trans.deep_merge(custom_trans)}
  rescue => e
    puts e.message
    puts e.backtrace.join("\n")

    {locale_str => {}}
  end

  def self.output_locale(locale=:en)
    result = ""
    result << "I18n.translations = #{load_yaml(locale).to_json};\n"
    result << "I18n.locale = '#{locale}';\n"
    result
  end

end

410:F:\git\coin\exchange\peatio-master\lib\kline_db.rb
module KlineDB
  class << self

    def redis
      @redis ||= Redis.new url: ENV["REDIS_URL"], db: 1
    end

    def kline(market, period)
      key = "peatio:#{market}:k:#{period}"
      length = redis.llen(key)
      data = redis.lrange(key, length - 5000, -1).map{|str| JSON.parse(str)}
    end

  end
end
```

411:F:\git\coin\exchange\peatio-master\lib\market_constraint.rb

```
class MarketConstraint
  def self.matches?(request)
    id = request.path_parameters[:market_id] || request.path_parameters[:id]
    market = Market.find_by_id(id)
    if market
      request.path_parameters[:market] = id
      request.path_parameters[:ask] = market.base_unit
      request.path_parameters[:bid] = market.quote_unit
    else
      false
    end
  end
end
```

412:F:\git\coin\exchange\peatio-master\lib\middleware\i18n_js.rb

```
module Middleware
  class I18nJs
    def initialize(app)
      @app = app
    end

    def call(env)
      update_cache(env['ORIGINAL_FULLPATH']) if matching?(env['ORIGINAL_FULLPATH'])
      @app.call(env)
    end

    private

    def matching?(path)
```

```
      path =~ /^\/assets\/locales\/[a-z\-A-Z]*\.js/
    end

    def cache_dir
      @cache_dir ||= Rails.root.join("public/assets/locales")
    end

    def update_cache(path)
      locale = path.scan(/[a-z\-A-Z]*\.js/).first.gsub('.js', '')
      file_path = "#{cache_dir}/#{locale}.js"

      FileUtils.mkdir_p(cache_dir)
      File.open(file_path, "w+") do |file|
        file << JsLocaleHelper.output_locale(locale)
      end
    end
  end
end
```

413:F:\git\coin\exchange\peatio-master\lib\middleware\security.rb

```
module Middleware
  class Security

    def initialize(app)
      @app = app
    end

    def call(env)
      env['HTTP_HOST'] = host
      env['HTTP_X_FORWARDED_HOST'] = host
      @app.call(env)
    end

    def host
      @host ||= ENV['URL_PORT'] ? "#{ENV['URL_HOST']}:#{ENV['URL_PORT']}" :
ENV['URL_HOST']
    end

  end
end
```

414:F:\git\coin\exchange\peatio-master\lib\peatio\version.rb

```
module Peatio
  VERSION = "0.2.1"
end

415:F:\git\coin\exchange\peatio-master\spec\api\api_v2\auth\authenticator_spec.rb
require 'spec_helper'

describe APIv2::Auth::Authenticator do
  Authenticator = APIv2::Auth::Authenticator

  let(:token) { create(:api_token) }
  let(:tonce) { time_to_milliseconds }

  let(:endpoint) { stub('endpoint', options: {route_options: {scopes: ['identity']}})}
  let(:request) { stub('request', request_method: 'GET', path_info: '/', env: {'api.endpoint' =>
endpoint}) }
  let(:payload) {
"GET|/api/|access_key=#{token.access_key}&foo=bar&hello=world&tonce=#{tonce}" }

  let(:params) do
    Hashie::Mash.new({
      "access_key" => token.access_key,
      "tonce"      => tonce,
      "foo"        => "bar",
      "hello"      => "world",
      "route_info" => Grape::Route.new,
      "signature"  => APIv2::Auth::Utils.hmac_signature(token.secret_key, payload)
    })
  end

  subject { Authenticator.new(request, params) }

  its(:authenticate!)         { should == token }
  its(:token)                 { should == token }
  its(:canonical_verb)        { should == 'GET' }
  its(:canonical_uri)         { should == '/' }
  its(:canonical_query)       { should ==
"access_key=#{token.access_key}&foo=bar&hello=world&tonce=#{tonce}" }

  it "should not be authentic without access key" do
    params[:access_key] = ''
    lambda {
      subject.authenticate!
    }.should raise_error(APIv2::InvalidAccessKeyError)
  end

  it "should not be authentic without signature" do
    subject
    params[:signature] = nil
    lambda {
      subject.authenticate!
    }.should raise_error(APIv2::IncorrectSignatureError)
  end

  it "should not be authentic without tonce" do
    params[:tonce] = nil
    params[:signature] = APIv2::Auth::Utils.hmac_signature(token.secret_key,
"GET|/|access_key=#{token.access_key}&foo=bar&hello=world&tonce=")
    lambda {
      subject.authenticate!
    }.should raise_error(APIv2::InvalidTonceError)
  end

  it "should return false on unmatched signature" do
    params[:signature] = 'fake'
    lambda {
      subject.authenticate!
    }.should raise_error(APIv2::IncorrectSignatureError)
  end

  it "should be invalid if tonce is not within 30s" do
    params[:tonce] = time_to_milliseconds(31.seconds.ago)
    lambda {
      Authenticator.new(request, params).check_tonce!
    }.should raise_error(APIv2::InvalidTonceError)

    params[:tonce] = time_to_milliseconds(31.seconds.since)
    lambda {
      Authenticator.new(request, params).check_tonce!
    }.should raise_error(APIv2::InvalidTonceError)
  end

  it "should not be authentic on repeated tonce" do
    params[:tonce] = time_to_milliseconds(Time.now)
    subject.check_tonce!

    lambda {
      subject.check_tonce!
    }.should raise_error(APIv2::TonceUsedError)
  end

  it "should not be authentic for invalid token" do
    params[:access_key] = 'fake'
    subject.token.should be_nil
    lambda {
      subject.authenticate!
    }.should raise_error(APIv2::InvalidAccessKeyError)
  end

  it "should be authentic if associated member is disabled" do
    token.member.update_attributes disabled: true
    lambda {
      subject.token.should_not be_nil
      subject.authenticate!
    }.should_not raise_error
  end

  it "should not be authentic if api access is disabled" do
    token.member.update_attributes api_disabled: true
    lambda {
      subject.authenticate!
    }.should raise_error(APIv2::DisabledAccessKeyError)
  end

  it "should not be authentic if token is expired" do
    token.update_attributes expire_at: 1.second.ago
    lambda {
      subject.authenticate!
    }.should raise_error(APIv2::ExpiredAccessKeyError)
  end

  it "should not be authentic if token is soft deleted" do
    token.destroy
    APIToken.find_by_id(token.id).should be_nil
    APIToken.with_deleted.find_by_id(token.id).should == token
    lambda {
      subject.authenticate!
    }.should raise_error(APIv2::InvalidAccessKeyError)
  end
end

416:F:\git\coin\exchange\peatio-master\spec\api\api_v2\auth\middleware_spec.rb
require 'spec_helper'

describe APIv2::Auth::Middleware do

  class TestApp < Grape::API
    helpers APIv2::Helpers
    use APIv2::Auth::Middleware

    get '/' do
      authenticate!
      current_user.email
    end
  end

  let(:app) do
    TestApp.new
  end

  let(:token) { create(:api_token) }

  it "should refuse request without credentials" do
    get '/'
    response.code.should == '401'
    response.body.should == "{\"error\":{\"code\":2001,\"message\":\"Authorization failed\"}}"
  end

  it "should refuse request with incorrect credentials" do
    get '/', access_key: token.access_key, tonce: time_to_milliseconds, signature: 'wrong'
    response.code.should == '401'
    response.body.should == "{\"error\":{\"code\":2005,\"message\":\"Signature wrong is
incorrect.\"}}"
  end

  it "should authorize request with correct param credentials" do
    signed_get '/', token: token
    response.should be_success
    response.body.should == token.member.email
  end

end

417:F:\git\coin\exchange\peatio-master\spec\api\api_v2\auth\utils_spec.rb
require 'spec_helper'

describe APIv2::Auth::Utils do
  Utils = APIv2::Auth::Utils

  context '.generate_access_key' do
    it "should be a string longer than 40 characters" do
      Utils.generate_access_key.should match(/^[a-zA-Z0-9]{40}$/)
    end
  end

  context '.generate_secret_key' do
    it "should be a string longer than 40 characters" do
      Utils.generate_secret_key.should match(/^[a-zA-Z0-9]{40}$/)
    end
  end
end

418:F:\git\coin\exchange\peatio-master\spec\api\api_v2\deposits_spec.rb
require 'spec_helper'

describe APIv2::Deposits do

  let(:member) { create(:member) }
  let(:other_member) { create(:member) }
  let(:token)  { create(:api_token, member: member) }

  describe "GET /api/v2/deposits" do

    before do
      create(:deposit, member: member, currency: 'btc')
      create(:deposit, member: member, currency: 'cny')
      create(:deposit, member: member, currency: 'cny', txid: 1, amount: 520)
      create(:deposit, member: member, currency: 'btc', created_at: 2.day.ago,  txid: 'test', amount: 
111)
      create(:deposit, member: other_member, currency: 'cny', txid: 10)
```

```
end

it "should require deposits authentication" do
  get '/api/v2/deposits', token: token
  response.code.should =='401'
end

it "login deposits" do
  signed_get '/api/v2/deposits', token: token
  response.should be_success
end

it "deposits num" do
  signed_get '/api/v2/deposits', token: token
  JSON.parse(response.body).size.should == 3
end

it "should return limited deposits" do
  signed_get '/api/v2/deposits', params: {limit: 1}, token: token
  JSON.parse(response.body).size.should == 1
end

it "should filter deposits by state" do
  signed_get '/api/v2/deposits', params: {state: 'cancelled'}, token: token
  JSON.parse(response.body).size.should == 0

  d = create(:deposit, member: member, currency: 'btc')
  d.submit!
  signed_get '/api/v2/deposits', params: {state: 'submitted'}, token: token
  json = JSON.parse(response.body)
  json.size.should == 1
  json.first['txid'].should == d.txid
end

it "deposits currency cny" do
  signed_get '/api/v2/deposits', params: {currency: 'cny'}, token: token
  result = JSON.parse(response.body)
  result.should have(2).deposits
  result.all? {|d| d['currency'] == 'cny' }.should be_true
end

it "should return 404 if txid not exist" do
```

```
      signed_get '/api/v2/deposit', params: {txid: 5}, token: token
      response.code.should == '404'
    end

    it "should return 404 if txid not belongs_to you " do
      signed_get '/api/v2/deposit', params: {txid: 10}, token: token
      response.code.should == '404'
    end

    it "should ok txid if exist" do
      signed_get '/api/v2/deposit', params: {txid: 1}, token: token

      response.code.should == '200'
      JSON.parse(response.body)['amount'].should == '520.0'
    end

    it "should return deposit no time limit " do
      signed_get '/api/v2/deposit', params: {txid: 'test'}, token: token

      response.code.should == '200'
      JSON.parse(response.body)['amount'].should == '111.0'
    end
  end
end
```

419:F:\git\coin\exchange\peatio-master\spec\api\api_v2\entities\account_spec.rb

```
require 'spec_helper'

describe APIv2::Entities::Account do

  let(:account) { create(:account_btc) }

  subject { OpenStruct.new APIv2::Entities::Account.represent(account).serializable_hash }

  its(:currency) { should == 'btc' }
  its(:balance)  { should == '100.0'}
  its(:locked)   { should == '0.0' }

end
```

420:F:\git\coin\exchange\peatio-master\spec\api\api_v2\entities\member_spec.rb

```
require 'spec_helper'
```

```
describe APIv2::Entities::Member do

  let(:member) { create(:verified_member) }

  subject { OpenStruct.new APIv2::Entities::Member.represent(member).serializable_hash }

  before { Currency.stubs(:codes).returns(%w(cny btc)) }

  its(:sn)        { should == member.sn }
  its(:name)      { should == member.name }
  its(:email)     { should == member.email }
  its(:activated) { should == true }
  its(:accounts)  { should =~ [{:currency=>"cny", :balance=>"0.0", :locked=>"0.0"},
{:currency=>"btc", :balance=>"0.0", :locked=>"0.0"}] }

end

421:F:\git\coin\exchange\peatio-master\spec\api\api_v2\entities\order_spec.rb
require 'spec_helper'

describe APIv2::Entities::Order do

  let(:order)  { create(:order_ask, currency: 'btccny', price: '12.326'.to_d, volume: '3.14',
origin_volume: '12.13') }

  context "default exposure" do
    subject { OpenStruct.new APIv2::Entities::Order.represent(order, {}).serializable_hash }

    its(:id)               { should == order.id }
    its(:price)            { should == order.price }
    its(:avg_price)        { should == ::Trade::ZERO }
    its(:volume)           { should == order.origin_volume }
    its(:remaining_volume) { should == order.volume }
    its(:executed_volume)  { should == (order.origin_volume - order.volume)}
    its(:state)            { should == order.state }
    its(:market)           { should == order.market }
    its(:created_at)       { should == order.created_at.iso8601 }
    its(:side)             { should == 'sell' }
    its(:trades)           { should be_nil }
    its(:trades_count)     { should == 0 }
  end
  context "full exposure" do
    it "should expose related trades" do
      create(:trade, ask: order, volume: '8.0', price: '12')
      create(:trade, ask: order, volume: '0.99', price: '12.56')

      json = APIv2::Entities::Order.represent(order, type: :full).serializable_hash
      json[:trades].should have(2).trades
    end
  end

end
```

422:F:\git\coin\exchange\peatio-master\spec\api\api_v2\entities\trade_spec.rb

```
require 'spec_helper'

describe APIv2::Entities::Trade do

  let(:trade) { create(:trade, ask: create(:order_ask), bid: create(:order_bid)) }

  subject { OpenStruct.new APIv2::Entities::Trade.represent(trade, side: 'sell').serializable_hash }

  its(:id)              { should == trade.id }
  its(:price)           { should == trade.price }
  its(:volume)          { should == trade.volume }
  its(:funds)           { should == trade.funds }
  its(:market)          { should == trade.currency }
  its(:created_at)      { should == trade.created_at.iso8601 }
  its(:side)            { should == 'sell' }
  its(:order_id)        { should be_nil }

end
```

423:F:\git\coin\exchange\peatio-master\spec\api\api_v2\helpers_spec.rb

```
require 'spec_helper'

module APIv2

  class AuthTest < Grape::API
    get("/auth_test") do
      authenticate!
      current_user
    end
  end

  class Mount
    mount AuthTest
  end

end

describe APIv2::Helpers do

  context "#authentic?" do

    let(:tonce)  { time_to_milliseconds }
    let!(:token) { create(:api_token) }

    context "Authenticate using headers" do
      pending
    end

    context "Authenticate using params" do
      let(:payload) {
"GET|/api/v2/auth_test|access_key=#{token.access_key}&foo=bar&hello=world&tonce=#{tonce}" }
      let(:signature) { APIv2::Auth::Utils.hmac_signature(token.secret_key, payload) }

      it "should response successfully" do
        get '/api/v2/auth_test', access_key: token.access_key, signature: signature, foo: 'bar', hello:
'world', tonce: tonce
        response.should be_success
      end

      it "should set current user" do
        get '/api/v2/auth_test', access_key: token.access_key, signature: signature, foo: 'bar', hello:
'world', tonce: tonce
        response.body.should == token.member.reload.to_json
      end

      it "should fail authorization" do
        get '/api/v2/auth_test'
        response.code.should == '401'
        response.body.should == "{\"error\":{\"code\":2001,\"message\":\"Authorization failed\"}}"
      end
    end

  end

end

424:F:\git\coin\exchange\peatio-master\spec\api\api_v2\markets_spec.rb
require 'spec_helper'

describe APIv2::Markets do

  describe "GET /api/v2/markets" do
    it "should all available markets" do
      get '/api/v2/markets'
      response.should be_success
      response.body.should == '[{"id":"btccny","name":"BTC/CNY"}]'
    end
  end

end

425:F:\git\coin\exchange\peatio-master\spec\api\api_v2\members_spec.rb
require 'spec_helper'

describe APIv2::Members do

  let(:member) do
    create(:verified_member).tap {|m|
      m.get_account(:btc).update_attributes(balance: 12.13,   locked: 3.14)
      m.get_account(:cny).update_attributes(balance: 2014.47, locked: 0)
    }
  end
  let(:token)  { create(:api_token, member: member) }

  describe "GET /members/me" do
    before { Currency.stubs(:codes).returns(%w(cny btc)) }

    it "should require auth params" do
      get '/api/v2/members/me'
      response.code.should == '400'
      response.body.should == '{"error":{"code":1001,"message":"access_key is missing, tonce is
missing, signature is missing"}}'
```

```
    end

    it "should require authentication" do
      get '/api/v2/members/me', access_key: 'test', tonce: time_to_milliseconds, signature: 'test'
      response.code.should == '401'
      response.body.should == '{"error":{"code":2008,"message":"The access key test does not
exist."}}'
    end

    it "should return current user profile with accounts info" do
      signed_get "/api/v2/members/me", token: token
      response.should be_success

      result = JSON.parse(response.body)
      result['sn'].should == member.sn
      result['activated'].should == true
      result['accounts'].should =~ [
        {"currency" => "cny", "balance" => "2014.47", "locked" => "0.0"},
        {"currency" => "btc", "balance" =>"12.13",    "locked" => "3.14"}
      ]
    end
  end

end

426:F:\git\coin\exchange\peatio-master\spec\api\api_v2\mount_spec.rb
require 'spec_helper'

module APIv2
  class Mount

    get "/null" do
      ''
    end

    get "/broken" do
      raise Error, code: 2014310, text: 'MtGox bankrupt'
    end

  end
end
```

```
describe APIv2::Mount do

  it "should use auth and attack middleware" do
    APIv2::Mount.middleware.should == [[APIv2::Auth::Middleware], [Rack::Attack]]
  end

  it "should allow 3rd party ajax call" do
    get "/api/v2/null"
    response.should be_success
    response.headers['Access-Control-Allow-Origin'].should == '*'
  end

  context "handle exception on request processing" do
    it "should render json error message" do
      get "/api/v2/broken"
      response.code.should == '400'
      JSON.parse(response.body).should == {'error' => {'code' => 2014310, 'message' => "MtGox
bankrupt"}}
    end
  end

  context "handle exception on request routing" do
    it "should render json error message" do
      get "/api/v2/non/exist"
      response.code.should == '404'
      response.body.should == "Not Found"
    end
  end

end

427:F:\git\coin\exchange\peatio-master\spec\api\api_v2\orders_spec.rb
require 'spec_helper'

describe APIv2::Orders do

  let(:member) { create(:member) }
  let(:token)  { create(:api_token, member: member) }

  describe "GET /api/v2/orders" do
    before do
      create(:order_bid, currency: 'btccny', price: '11'.to_d, volume: '123.123456789', member:
```

```
member)
      create(:order_bid, currency: 'btccny', price: '12'.to_d, volume: '123.123456789', member:
member, state: Order::CANCEL)
      create(:order_ask, currency: 'btccny', price: '13'.to_d, volume: '123.123456789', member:
member)
      create(:order_ask, currency: 'btccny', price: '14'.to_d, volume: '123.123456789', member:
member, state: Order::DONE)
    end

    it "should require authentication" do
      get "/api/v2/orders", market: 'btccny'
      response.code.should == '401'
    end

    it "should validate market param" do
      signed_get '/api/v2/orders', params: {market: 'mtgox'}, token: token
      response.code.should == '400'
      JSON.parse(response.body).should == {"error" => {"code" => 1001,"message" => "market
does not have a valid value"}}
    end

    it "should validate state param" do
      signed_get '/api/v2/orders', params: {market: 'btccny', state: 'test'}, token: token
      response.code.should == '400'
      JSON.parse(response.body).should == {"error" => {"code" => 1001,"message" => "state does
not have a valid value"}}
    end

    it "should return active orders by default" do
      signed_get '/api/v2/orders', params: {market: 'btccny'}, token: token
      response.should be_success
      JSON.parse(response.body).size.should == 2
    end

    it "should return complete orders" do
      signed_get '/api/v2/orders', params: {market: 'btccny', state: Order::DONE}, token: token
      response.should be_success
      JSON.parse(response.body).first['state'].should == Order::DONE
    end

    it "should return paginated orders" do
      signed_get '/api/v2/orders', params: {market: 'btccny', limit: 1, page: 1}, token: token
      response.should be_success
      JSON.parse(response.body).first['price'].should == '11.0'

      signed_get '/api/v2/orders', params: {market: 'btccny', limit: 1, page: 2}, token: token
      response.should be_success
      JSON.parse(response.body).first['price'].should == '13.0'
    end

    it "should sort orders" do
      signed_get '/api/v2/orders', params: {market: 'btccny', order_by: 'asc'}, token: token
      response.should be_success
      orders = JSON.parse(response.body)
      orders[0]['id'].should < orders[1]['id']

      signed_get '/api/v2/orders', params: {market: 'btccny', order_by: 'desc'}, token: token
      response.should be_success
      orders = JSON.parse(response.body)
      orders[0]['id'].should > orders[1]['id']
    end

  end

  describe "GET /api/v2/order" do
    let(:order)  { create(:order_bid, currency: 'btccny', price: '12.326'.to_d, volume: '3.14',
origin_volume: '12.13', member: member, trades_count: 1) }
    let!(:trade) { create(:trade, bid: order) }

    it "should get specified order" do
      signed_get "/api/v2/order", params: {id: order.id}, token: token
      response.should be_success

      result = JSON.parse(response.body)
      result['id'].should == order.id
      result['executed_volume'].should == '8.99'
    end

    it "should include related trades" do
      signed_get "/api/v2/order", params: {id: order.id}, token: token

      result = JSON.parse(response.body)
      result['trades_count'].should == 1
      result['trades'].should have(1).trade
      result['trades'].first['id'].should == trade.id
      result['trades'].first['side'].should == 'buy'
    end

    it "should get 404 error when order doesn't exist" do
      signed_get "/api/v2/order", params: {id: 99999}, token: token
      response.code.should == '404'
    end
  end

  describe "POST /api/v2/orders/multi" do
    before do
      member.get_account(:btc).update_attributes(balance: 100)
      member.get_account(:cny).update_attributes(balance: 100000)
    end

    it "should create a sell order and a buy order" do
      params = {
        market: 'btccny',
        orders: [
          {side: 'sell', volume: '12.13', price: '2014'},
          {side: 'buy',  volume: '17.31', price: '2005'}
        ]
      }

      expect {
        signed_post '/api/v2/orders/multi', token: token, params: params
        response.should be_success

        result = JSON.parse(response.body)
        result.should have(2).orders
        result.first['side'].should   == 'sell'
        result.first['volume'].should == '12.13'
        result.last['side'].should    == 'buy'
        result.last['volume'].should  == '17.31'
      }.to change(Order, :count).by(2)
    end

    it "should create nothing on error" do
      params = {
        market: 'btccny',
        orders: [
          {side: 'sell', volume: '12.13', price: '2014'},
          {side: 'buy',  volume: '17.31', price: 'test'} # <- invalid price
        ]
      }

      expect {
        AMQPQueue.expects(:enqueue).times(0)
        signed_post '/api/v2/orders/multi', token: token, params: params
        response.code.should == '400'
        response.body.should == '{"error":{"code":2002,"message":"Failed to create order. Reason:
  Validation failed: Price must be greater than 0"}}'
      }.not_to change(Order, :count)
    end
  end

  describe "POST /api/v2/orders" do
    it "should create a sell order" do
      member.get_account(:btc).update_attributes(balance: 100)

      expect {
        signed_post '/api/v2/orders', token: token, params: {market: 'btccny', side: 'sell', volume:
  '12.13', price: '2014'}
        response.should be_success
        JSON.parse(response.body)['id'].should == OrderAsk.last.id
      }.to change(OrderAsk, :count).by(1)
    end

    it "should create a buy order" do
      member.get_account(:cny).update_attributes(balance: 100000)

      expect {
        signed_post '/api/v2/orders', token: token, params: {market: 'btccny', side: 'buy', volume:
  '12.13', price: '2014'}
        response.should be_success
        JSON.parse(response.body)['id'].should == OrderBid.last.id
      }.to change(OrderBid, :count).by(1)
    end

    it "should set order source to APIv2" do
      member.get_account(:cny).update_attributes(balance: 100000)
      signed_post '/api/v2/orders', token: token, params: {market: 'btccny', side: 'buy', volume:
  '12.13', price: '2014'}
```

```
      OrderBid.last.source.should == 'APIv2'
    end

    it "should return cannot lock funds error" do
      expect {
        signed_post '/api/v2/orders', params: {market: 'btccny', side: 'sell', volume: '12.13', price:
'2014'}
        response.code.should == '400'
        response.body.should == '{"error":{"code":2002,"message":"Failed to create order. Reason:
cannot lock funds (amount: 12.13)"}}'
      }.not_to change(OrderAsk, :count).by(1)
    end

    it "should give a number as volume parameter" do
      signed_post '/api/v2/orders', params: {market: 'btccny', side: 'sell', volume: 'test', price: '2014'}
      response.code.should == '400'
      response.body.should == '{"error":{"code":2002,"message":"Failed to create order. Reason:
Validation failed: Volume must be greater than 0"}}'
    end

    it "should give a number as price parameter" do
      signed_post '/api/v2/orders', params: {market: 'btccny', side: 'sell', volume: '12.13', price: 'test'}
      response.code.should == '400'
      response.body.should == '{"error":{"code":2002,"message":"Failed to create order. Reason:
Validation failed: Price must be greater than 0"}}'
    end
  end

  describe "POST /api/v2/order/delete" do
    let!(:order)  { create(:order_bid, currency: 'btccny', price: '12.326'.to_d, volume: '3.14',
origin_volume: '12.13', locked: '20.1082', origin_locked: '38.0882', member: member) }

    context "succesful" do
      before do
        member.get_account(:cny).update_attributes(locked: order.price*order.volume)
      end

      it "should cancel specified order" do
        AMQPQueue.expects(:enqueue).with(:matching, action: 'cancel', order:
order.to_matching_attributes)
        expect {
          signed_post "/api/v2/order/delete", params: {id: order.id}, token: token
          response.should be_success
          JSON.parse(response.body)['id'].should == order.id
        }.not_to change(Order, :count)
      end
    end

    context "failed" do
      it "should return order not found error" do
        signed_post "/api/v2/order/delete", params: {id: '0'}, token: token
        response.code.should == '400'
        JSON.parse(response.body)['error']['code'].should == 2003
      end
    end

  end

  describe "POST /api/v2/orders/clear" do

    before do
      create(:order_ask, currency: 'btccny', price: '12.326', volume: '3.14', origin_volume: '12.13',
member: member)
      create(:order_bid, currency: 'btccny', price: '12.326', volume: '3.14', origin_volume: '12.13',
member: member)

      member.get_account(:btc).update_attributes(locked: '5')
      member.get_account(:cny).update_attributes(locked: '50')
    end

    it "should cancel all my orders" do
      member.orders.each do |o|
        AMQPQueue.expects(:enqueue).with(:matching, action: 'cancel', order:
o.to_matching_attributes)
      end

      expect {
        signed_post "/api/v2/orders/clear", token: token
        response.should be_success

        result = JSON.parse(response.body)
        result.should have(2).orders
      }.not_to change(Order, :count)
    end
    it "should cancel all my asks" do
      member.orders.where(type: 'OrderAsk').each do |o|
        AMQPQueue.expects(:enqueue).with(:matching, action: 'cancel', order: 
o.to_matching_attributes)
      end

      expect {
        signed_post "/api/v2/orders/clear", token: token, params: {side: 'sell'}
        response.should be_success

        result = JSON.parse(response.body)
        result.should have(1).orders
        result.first['id'].should == member.orders.where(type: 'OrderAsk').first.id
      }.not_to change(Order, :count)
    end

  end
end

428:F:\git\coin\exchange\peatio-master\spec\api\api_v2\order_books_spec.rb
require 'spec_helper'

describe APIv2::OrderBooks do

  describe "GET /api/v2/order_book" do
    before do
      5.times { create(:order_bid) }
      5.times { create(:order_ask) }
    end

    it "should return ask and bid orders on specified market" do
      get '/api/v2/order_book', market: 'btccny'
      response.should be_success

      result = JSON.parse(response.body)
      result['asks'].should have(5).asks
      result['bids'].should have(5).bids
    end

    it "should return limited asks and bids" do
      get '/api/v2/order_book', market: 'btccny', asks_limit: 1, bids_limit: 1
      response.should be_success

      result = JSON.parse(response.body)
      result['asks'].should have(1).asks
      result['bids'].should have(1).bids
    end
  end

  describe "GET /api/v2/depth" do
    let(:asks) { [['100', '2.0'], ['120', '1.0']] }
    let(:bids) { [['90', '3.0'], ['50', '1.0']] }

    before do
      global = mock("global", asks: asks, bids: bids)
      Global.stubs(:[]).returns(global)
    end

    it "should sort asks and bids from highest to lowest" do
      get '/api/v2/depth', market: 'btccny'
      response.should be_success

      result = JSON.parse(response.body)
      result['asks'].should == asks.reverse
      result['bids'].should == bids
    end
  end

end
```

429:F:\git\coin\exchange\peatio-master\spec\api\api_v2\tickers_spec.rb

```
require 'spec_helper'

describe APIv2::Tickers do

  describe "GET /api/v2/tickers" do
    it "returns ticker of all markets" do
      get "/api/v2/tickers"
      response.should be_success
      JSON.parse(response.body)['btccny']['at'].should_not be_nil
      JSON.parse(response.body)['btccny']['ticker'].should == {"buy"=>"0.0", "sell"=>"0.0",
"low"=>"0.0", "high"=>"0.0", "last"=>"0.0", "vol"=>"0.0"}
    end
  end

  describe "GET /api/v2/tickers/:market" do
    it "should return market tickers" do
      get "/api/v2/tickers/btccny"
      response.should be_success
      JSON.parse(response.body)['ticker'].should == {"buy"=>"0.0", "sell"=>"0.0", "low"=>"0.0",
"high"=>"0.0", "last"=>"0.0", "vol"=>"0.0"}
    end
  end

end

430:F:\git\coin\exchange\peatio-master\spec\api\api_v2\trades_spec.rb
require 'spec_helper'

describe APIv2::Trades do

  let(:member) do
    create(:verified_member).tap {|m|
      m.get_account(:btc).update_attributes(balance: 12.13,   locked: 3.14)
      m.get_account(:cny).update_attributes(balance: 2014.47, locked: 0)
    }
  end
  let(:token)  { create(:api_token, member: member) }

  let(:ask) { create(:order_ask, currency: 'btccny', price: '12.326'.to_d, volume: '123.123456789',
member: member) }
  let(:bid) { create(:order_bid, currency: 'btccny', price: '12.326'.to_d, volume: '123.123456789',
member: member) }

  let!(:ask_trade) { create(:trade, ask: ask, created_at: 2.days.ago) }
  let!(:bid_trade) { create(:trade, bid: bid, created_at: 1.day.ago) }

  describe 'GET /api/v2/trades' do
    it "should return all recent trades" do
      get '/api/v2/trades', market: 'btccny'
      response.should be_success
      JSON.parse(response.body).should have(2).trades
    end

    it "should return 1 trade" do
```

```
      get '/api/v2/trades', market: 'btccny', limit: 1
      response.should be_success
      JSON.parse(response.body).should have(1).trade
    end

    it "should return trades before timestamp" do
      another = create(:trade, bid: bid, created_at: 6.hours.ago)
      get '/api/v2/trades', market: 'btccny', timestamp: 8.hours.ago.to_i, limit: 1
      response.should be_success
      json = JSON.parse(response.body)
      json.should have(1).trade
      json.first['id'].should == bid_trade.id
    end

    it "should return trades between id range" do
      another = create(:trade, bid: bid)
      get '/api/v2/trades', market: 'btccny', from: ask_trade.id, to: another.id
      response.should be_success
      json = JSON.parse(response.body)
      json.should have(1).trade
      json.first['id'].should == bid_trade.id
    end

    it "should sort trades in reverse creation order" do
      get '/api/v2/trades', market: 'btccny'
      response.should be_success
      JSON.parse(response.body).first['id'].should == bid_trade.id
    end

    it "should get trades by from and limit" do
      another = create(:trade, bid: bid, created_at: 6.hours.ago)
      get '/api/v2/trades', market: 'btccny', from: ask_trade.id, limit: 1, order_by: 'asc'
      response.should be_success
      JSON.parse(response.body).first['id'].should == bid_trade.id
    end
  end

  describe 'GET /api/v2/trades/my' do
    it "should require authentication" do
      get '/api/v2/trades/my', market: 'btccny', access_key: 'test', tonce: time_to_milliseconds,
signature: 'test'
      response.code.should == '401'
      response.body.should == '{"error":{"code":2008,"message":"The access key test does not
exist."}}'
    end

    it "should return all my recent trades" do
      signed_get '/api/v2/trades/my', params: {market: 'btccny'}, token: token
      response.should be_success

      result = JSON.parse(response.body)
      result.find {|t| t['id'] == ask_trade.id }['side'].should == 'ask'
      result.find {|t| t['id'] == ask_trade.id }['order_id'].should == ask.id
      result.find {|t| t['id'] == bid_trade.id }['side'].should == 'bid'
      result.find {|t| t['id'] == bid_trade.id }['order_id'].should == bid.id
    end

    it "should return 1 trade" do
      signed_get '/api/v2/trades/my', params: {market: 'btccny', limit: 1}, token: token
      response.should be_success
      JSON.parse(response.body).should have(1).trade
    end

    it "should return trades before timestamp" do
      signed_get '/api/v2/trades/my', params: {market: 'btccny', timestamp: 30.hours.ago.to_i}, token:
token
      response.should be_success
      JSON.parse(response.body).should have(1).trade
    end

    it "should return limit out of range error" do
      signed_get '/api/v2/trades/my', params: {market: 'btccny', limit: 1024}, token: token
      response.code.should == '400'
      response.body.should == '{"error":{"code":1001,"message":"limit must be in range: 1..1000"}}'
    end
  end

end

431:F:\git\coin\exchange\peatio-master\spec\controllers\activations_controller_spec.rb
require 'spec_helper'

module Private
  describe ActivationsController do
```

```
    describe "GET /activations/new" do
      describe 'non-login user' do
        before { get :new }

        it { expect(response).to redirect_to(root_path) }
        it { expect(flash[:alert]).to match('login required') }
      end

      describe 'logged-in user but not activated yet' do
        let(:member) { create :member }
        let(:mail) { ActionMailer::Base.deliveries.last }
        before {
          session[:member_id] = member.id
          get :new
        }

        it { expect(member).not_to be_activated }
        it { expect(response).to redirect_to(settings_path) }
        it { expect(mail.subject).to match('Account Activation') }
      end

      describe 'logged-in user and verified already' do
        let(:member) { create :member, :activated }
        before {
          session[:member_id] = member.id
          get :new
        }

        it { expect(response).to redirect_to(settings_path) }
        it { expect(flash[:notice]).to match('has been verified successfully') }
      end
    end

  end
end

432:F:\git\coin\exchange\peatio-master\spec\controllers\admin\id_documents_controller_spec.rb
require 'spec_helper'

describe Admin::IdDocumentsController do
  let(:member) { create(:admin_member) }
  before {
    session[:member_id] = member.id
    two_factor_unlocked
  }

  describe 'GET index' do
    before { get :index }

    it { should respond_with :ok }
    it { should render_template(:index) }
  end

end

433:F:\git\coin\exchange\peatio-master\spec\controllers\admin\members_controller_spec.rb
require 'spec_helper'

describe Admin::MembersController do
  let(:member) { create(:admin_member) }
  before { session[:member_id] = member.id }

end

434:F:\git\coin\exchange\peatio-master\spec\controllers\admin\two_factors_spec.rb
require 'spec_helper'

describe Admin::TwoFactorsController do
  let(:member) { create(:admin_member) }
  let(:sms_two_factor) { member.sms_two_factor }
  let(:app_two_factor) { member.app_two_factor }

  before do
    session[:member_id] = member.id
    two_factor_unlocked
    app_two_factor.active!
    sms_two_factor.active!
    request.env["HTTP_REFERER"] = "where_i_came_from"
  end

  it { expect(sms_two_factor).to be_activated }
  it { expect(app_two_factor).to be_activated }
  it 'deactive sms two_factor' do
    delete :destroy, member_id: member.id, id: sms_two_factor.id
    expect(sms_two_factor.reload).not_to be_activated
  end

  it 'deactive app two_factor' do
    delete :destroy, member_id: member.id, id: app_two_factor.id
    expect(app_two_factor.reload).not_to be_activated
  end
end
```

435:F:\git\coin\exchange\peatio-master\spec\controllers\application_controller_spec.rb

```
require 'spec_helper'

describe ApplicationController do
  describe "CoinRPC::ConnectionRefusedError handling" do
    controller do
      def index
        raise CoinRPC::ConnectionRefusedError
      end
    end

    it 'renders errors/connection' do
      get :index
      expect(response).to render_template 'errors/connection'
    end
  end
end
```

436:F:\git\coin\exchange\peatio-master\spec\controllers\authentications\emails_controller_spec.rb

```
require 'spec_helper'

module Authentications
  describe EmailsController do
    let(:member) { create(:member, email: nil, activated: false) }
    before { session[:member_id] = member.id }

    describe 'GET new' do
      subject { get :new }

      it { should be_success }
      it  do
        get :new
        flash[:info].should == t('authentications.emails.new.setup_email')
      end
    end

    describe 'POST create' do
      let(:data) {
        { email: { address: 'xman@xman.com', user_id: '2' } }
      }

      it "should update current_user's email" do
        post :create, data
        member.reload
        member.email.should == 'xman@xman.com'
        member.activated.should be_false
      end
    end

  end
end

437:F:\git\coin\exchange\peatio-
master\spec\controllers\authentications\identities_controller_spec.rb
require 'spec_helper'

describe Authentications::IdentitiesController do
  let(:email) { 'xman@xman.com' }
  let(:member) { create(:verified_member, email: email) }
  before { session[:member_id] = member.id }

  describe 'GET new' do
    subject(:do_request) { get :new }
    it { should be_success }
    it "should set the identity" do
      do_request
      assigns(:identity).new_record?.should be_true
      assigns(:identity).email.should == email
    end
  end

  describe "POST create" do
```

```
    let(:password) { '111111' }
    let(:attrs) {
      { identity: { password: password, password_confirmation: password}}
    }

    subject(:do_request) { post :create, attrs}

    it "should create the ideneity" do
      expect do
        do_request
      end.to change(Identity, :count).by(1)
    end

    it "should be recirect to settings path with flash" do
      do_request
      response.should redirect_to(settings_path)
      flash[:notice].should == t("authentications.identities.create.success")
    end
  end

end
```

438:F:\git\coin\exchange\peatio-master\spec\controllers\authentications\weibo_accounts_controller.rb

```
require 'spec_helper'

module Authentications
  describe WeiboAccountsController do
    let(:member) { create(:member, email: nil, activated: false) }
    before { session[:member_id] = member.id }

    describe "DELETE destroy" do
      let!(:authentication) { create(:authentication, provider: 'weibo', member_id: member.id)}
      subject(:do_request) { delete :destroy}
      context "Only one authentication " do
        it "should not remove the authentication" do
          expect do
            do_request
          end.not_to change(Authentication, :count)
        end

        it "should tell user the reason" do
          do_request
          flash[:alert].should == t("authentications.weibo.destroy.last_auth_alert")
        end
      end

      context "More than one authentications" do
        let!(:auth_ideneity) { create(:authentication, provider: 'identity', member_id: member.id)}

        it "should delete the weibo authentication" do
          expect do
            do_request
          end.to change(Authentication, :count).by(-1)
        end

        it "should set the flash message" do
          do_request
          flash[:notice].should == t("authentications.weibo.destroy.unbind_success")
        end

      end

      it "should redirect user to settings_path" do
        do_request
        response.should redirect_to(settings_path)
      end
    end

  end
end

439:F:\git\coin\exchange\peatio-master\spec\controllers\private\assets_controller_spec.rb
require 'spec_helper'

describe Private::AssetsController do
  let(:member) { create :member }
  before { session[:member_id] = member.id }

  context "logged in user visit" do
    describe "GET /exchange_assets" do
      before { get :index }

      it { should respond_with :ok }
    end
  end

  context "non-login user visit" do
    before { session[:member_id] = nil }

    describe "GET /exchange_assets" do
      before { get :index }

      it { should respond_with :ok }
      it { expect(assigns(:btc_account)).to be_nil }
      it { expect(assigns(:cny_account)).to be_nil }
    end
  end

end

440:F:\git\coin\exchange\peatio-master\spec\controllers\private\funds_controller_spec.rb
require 'spec_helper'

describe Private::FundsController do

  context "Verified user with two factor" do
    let(:member) { create(:member, :activated, :verified, :app_two_factor_activated) }
    before { session[:member_id] = member.id }

    before do
      get :index
    end

    it { expect(response).to be_ok }
  end

  context "Verified user without two factor auth" do
    let(:member) { create(:member, :activated, :verified) }
    before { session[:member_id] = member.id }

    before do
      get :index
    end

    it { expect(member.two_factors).not_to be_activated }
    it { expect(response).to redirect_to(settings_path) }
  end

end

441:F:\git\coin\exchange\peatio-master\spec\controllers\private\fund_sources_controller_spec.rb
require 'spec_helper'

describe Private::FundSourcesController do
  let(:member) { create(:member) }
  before { session[:member_id] = member.id }

  describe 'POST create' do
    it "should not create fund_source with blank extra" do
      params = { fund_source: { extra: '',
                                currency: :cny,
                                uid: '1234 1234 1234'} }

      expect {
        post :create, params
        expect(response).not_to be_ok
      }.not_to change(FundSource, :count)
    end

    it "should not create fund_source with blank uid" do
      params = { fund_source: { extra: 'bank_code_1',
                                currency: :cny,
                                uid: ''} }

      expect {
        post :create, params
        expect(response).not_to be_ok
      }.not_to change(FundSource, :count)
    end

    it "should create fund_source successful" do
      params = { fund_source: { extra: 'bank_code_1',
                                currency: :cny,
                                uid: '1234 1234 1234'} }

      expect {
        post :create, params
        expect(response).to be_ok
      }.to change(FundSource, :count).by(1)
    end
  end

  describe 'UPDATE' do
    let!(:fund_source) { create(:fund_source, member: member, currency: :btc) }
    let(:account) { member.accounts.with_currency(:btc).first }

    it 'update default_withdraw_fund_source_id to account' do
      put :update, {id: fund_source.id}
      expect(account.default_withdraw_fund_source_id).to eq(fund_source.id)
    end
  end

  describe 'DELETE' do
    let!(:fund_source) { create(:fund_source, member: member) }

    it "should delete fund_source" do
      expect {
        delete :destroy, {id: fund_source.id}
        expect(response).to be_ok
      }.to change(FundSource, :count).by(-1)
    end
  end

end

describe 'routes for FundSources', type: :routing do
  it { expect(post: '/fund_sources').to be_routable }
  it { expect(put: '/fund_sources/1').to be_routable }
  it { expect(delete: '/fund_sources/1').to be_routable }
end
```

442:F:\git\coin\exchange\peatio-master\spec\controllers\private\id_documents_controller_spec.rb

```
require 'spec_helper'

describe Private::IdDocumentsController do
  let(:member) { create(:member) }
  before { session[:member_id] = member.id }

  describe 'GET edit' do
```

```
    before { get :edit }

    it { should respond_with :ok }
    it { should render_template(:edit) }
  end

  describe 'post update' do
    let(:attrs) {
      {
        id_document: {name: 'foobar'}
      }
    }

    before { put :update, attrs }
    it { should redirect_to(settings_path) }
    it { expect(assigns[:id_document].aasm_state).to eq('verifying') }
  end

end

443:F:\git\coin\exchange\peatio-master\spec\controllers\private\markets_controller_spec.rb
require 'spec_helper'

describe Private::MarketsController do
  let(:member) { create :member }
  before { session[:member_id] = member.id }

  context "logged in user" do
    describe "GET /markets/btccny" do
      before { get :show, data }

      it { should respond_with :ok }
    end
  end

  context "non-login user" do
    before { session[:member_id] = nil }

    describe "GET /markets/btccny" do
      before { get :show, data }

      it { should respond_with :ok }
      it { expect(assigns(:member)).to be_nil }
    end
  end

  private

  def data
    {
      id: 'btccny',
      market: 'btccny',
      ask: 'btc',
      bid: 'cny'
    }
  end

end
```

444:F:\git\coin\exchange\peatio-master\spec\controllers\private\order_asks_controller_spec.rb

```
require 'spec_helper'

describe Private::OrderAsksController do

  let(:member) do
    create(:member).tap {|m|
      m.get_account('btc').update_attributes(balance: '20')
    }
  end

  let(:market) { Market.find('btccny') }
  let(:params) do
    { market_id: market.id,
      market:    market.id,
      ask:       market.base_unit,
      bid:       market.quote_unit,
      order_ask: { ord_type: 'limit', origin_volume: '12.13', price: '2014.47' }
    }
  end

  context 'POST :create' do
    it "should create a sell order" do
      expect {
        post :create, params, {member_id: member.id}
        response.should be_success
        response.body.should == '{"result":true,"message":"Success"}'
      }.to change(OrderAsk, :count).by(1)
    end

    it "should set order source to Web" do
      post :create, params, {member_id: member.id}
      assigns(:order).source.should == 'Web'
    end
  end

  context 'POST :clear' do
    it "should cancel all my asks in current market" do
      o1 = create(:order_ask, member: member, currency: market)
      o2 = create(:order_ask, member: member, currency: Market.find(:ptsbtc))
      member.should have(2).orders

      post :clear, {market_id: market.id}, {member_id: member.id}
      response.should be_success
      assigns(:orders).size.should == 1
      assigns(:orders).first.should == o1
    end
  end

end
```

445:F:\git\coin\exchange\peatio-master\spec\controllers\private\order_bids_controller_spec.rb

```
require 'spec_helper'

describe Private::OrderBidsController do

  let(:member) do
    create(:member).tap {|m|
      m.get_account('cny').update_attributes(balance: '30000')
    }
  end

  let(:market) { Market.find('btccny') }
  let(:params) do
    { market_id: market.id,
      market:    market.id,
      ask:       market.base_unit,
      bid:       market.quote_unit,
      order_bid: { ord_type: 'limit', origin_volume: '12.13', price: '2014.47' }
    }
  end

  context 'POST :create' do
    it "should create a buy order" do
      expect {
        post :create, params, {member_id: member.id}
        response.should be_success
        response.body.should == '{"result":true,"message":"Success"}'
      }.to change(OrderBid, :count).by(1)
    end

    it "should set order source to Web" do
      post :create, params, {member_id: member.id}
      assigns(:order).source.should == 'Web'
    end
  end

  context 'POST :clear' do
    it "should cancel all my bids in current market" do
      o1 = create(:order_bid, member: member, currency: market)
      o2 = create(:order_bid, member: member, currency: Market.find(:ptsbtc))
      member.should have(2).orders

      post :clear, {market_id: market.id}, {member_id: member.id}
      response.should be_success
      assigns(:orders).size.should == 1
      assigns(:orders).first.should == o1
    end
  end

end

446:F:\git\coin\exchange\peatio-master\spec\controllers\private\settings_controller_spec.rb
require 'spec_helper'

describe Private::SettingsController do
  let(:member) { create :member }
  before { session[:member_id] = member.id }
  describe 'GET /index' do
    before { get :index }

    it { should respond_with :ok }
    it { should render_template(:index) }
  end
end

447:F:\git\coin\exchange\peatio-master\spec\controllers\reset_password_controller_spec.rb
require 'spec_helper'

describe ResetPasswordsController do
  before do
    get :new
  end

  it { expect(response).to be_ok }

end

448:F:\git\coin\exchange\peatio-master\spec\controllers\two_factors_controller_spec.rb
require 'spec_helper'

describe TwoFactorsController do
  describe 'GET :show' do
    let(:member) { create :member, :sms_two_factor_activated }
    before { session[:member_id] = member.id }

    context 'send sms verify code' do
      let(:do_request) { get :show, {id: :sms, refresh: true} }

      it {
        AMQPQueue.expects(:enqueue).with(:sms_notification, anything)
        do_request
      }
    end

    context 'two factor auth not locked' do
      let(:do_request) { get :show, {id: :sms} }

      before { do_request }
      it { expect(response).to be_ok }
    end

    context 'two factor auth locked' do
      let(:do_request) { get :show, {id: :sms} }

      before {
        controller.stubs(:two_factor_failed_locked?).returns(true)
        do_request
      }

      render_views

      it { expect(response).not_to be_ok }
      it { expect(response.status).to eq(423) }
      it { expect(response.body).not_to be_blank }
    end
  end

  describe 'GET :index' do
    context 'member without two_factor' do
      let(:member) { create :member }
      before { session[:member_id] = member.id }

      before { get :index }

      it { expect(response).to redirect_to(settings_path) }
    end

    context 'member with sms_two_factor activated' do
      let(:member) { create :member, :sms_two_factor_activated }
      before { session[:member_id] = member.id }

      before { get :index }

      it { expect(response).to be_ok }
      it { expect(response).to render_template('index') }
    end
  end

  describe 'PUT :update' do
    let(:member) { create :member, :sms_two_factor_activated }
    context 'with wrong otp' do
      let(:attrs) { { id: :sms,
                      two_factor: { type: :sms,
                                    otp: 'wrong code' } } }

      before {
        session[:member_id] = member.id
        put :update, attrs
      }

      it { expect(response).to redirect_to(two_factors_path) }
      it { expect(flash[:alert]).to match('verification code error') }
    end

    context 'with right otp' do
      let(:attrs) { { id: :sms,
                      two_factor: { type: :sms,
                                    otp: member.sms_two_factor.otp_secret } } }

      before {
        session[:member_id] = member.id
        put :update, attrs
      }

      it { expect(response).to redirect_to(settings_path) }
      it { expect(session[:two_factor_unlock]).to be_true }
      it { expect(session[:two_factor_unlock_at]).not_to be_blank }
    end
  end
end

449:F:\git\coin\exchange\peatio-master\spec\controllers\verify\google_auths_controller_spec.rb
require 'spec_helper'

describe Verify::GoogleAuthsController do
  let(:member) { create :member }
  before { session[:member_id] = member.id }

  describe 'GET /show' do
    before { get :show }
    context 'not activated yet' do
      it { should respond_with :ok }
      it { should render_template(:show) }
      it "member should have two_factor prepared" do
        expect(member.two_factors).not_to be_empty
      end
    end

    context 'already activated' do
      let(:member) { create :member, :app_two_factor_activated }

      it { should redirect_to(settings_path) }
    end
  end

  describe 'get /edit' do
    context 'not activated' do
      before { get :edit }

      it { expect(member.app_two_factor).not_to be_activated }
      it { should redirect_to(settings_path) }
    end

    context 'activated' do
      let(:member) { create :member, :app_two_factor_activated }
      before { session[:member_id] = member.id }

      before { get :edit }

      it { should respond_with :ok }
      it { should render_template(:edit) }
    end
  end
end

450:F:\git\coin\exchange\peatio-master\spec\controllers\verify\sms_auths_controller_spec.rb
require 'spec_helper'

module Verify
  describe SmsAuthsController do

    describe 'GET verify/sms_auth' do
      let(:member) { create :verified_member }
      before { session[:member_id] = member.id }

      before do
        get :show
      end

      it { expect(response).to be_success }
      it { expect(response).to render_template(:show) }

      context 'already verified' do
        let(:member) { create :member, :sms_two_factor_activated }

        it { should redirect_to(settings_path) }
      end
    end

    describe 'PUT verify/sms_auth in send code phase' do
      let(:member) { create :member }
      let(:attrs) {
        {
          format: :js,
          sms_auth: {country: 'CN', phone_number: '123-1234-1234'},
          commit: 'send_code'
        }
      }

      subject { assigns(:sms_auth) }

      before {
        session[:member_id] = member.id
        put :update, attrs
      }

      it { should_not be_nil }
      its(:otp_secret) { should_not be_blank }

      context "with empty number" do
        let(:attrs) {
          {
            format: :js,
            sms_auth: {country: '', phone_number: ''},
          commit: 'send_code'
        }
      }

      before { put :update, attrs }

      it "should not be ok" do
        expect(response).not_to be_ok
      end
    end

    context "with wrong number" do
      let(:attrs) {
        {
          format: :js,
          sms_auth: {country: 'CN', phone_number: 'wrong number'},
          commit: 'send_code'
        }
      }

      before { put :update, attrs }

      it "should not be ok" do
        expect(response).not_to be_ok
      end

      it "should has error message" do
        expect(response.body).not_to be_blank
      end
    end

    context "with right number" do
      let(:attrs) {
        {
          format: :js,
          sms_auth: {country: 'CN', phone_number: '133.1234.1234'},
          commit: 'send_code'
        }
      }

      before do
        put :update, attrs
        end

        it { expect(response).to be_ok }
        it { expect(member.reload.phone_number).to eq('8613312341234') }
      end
    end

    describe 'POST verify/sms_auth in verify code phase' do
      let(:member) { create :member }
      let(:sms_auth) { member.sms_two_factor }
      before { session[:member_id] = member.id }

      context "with empty code" do
        let(:attrs) {
          {
            format: :js,
            sms_auth: {otp: ''}
          }
        }

        before do
          put :update, attrs
        end

        it "not return ok status" do
          expect(response).not_to be_ok
        end
      end

      context "with wrong code" do
        let(:attrs) {
          {
            format: :js,
            sms_auth: {otp: 'foobar'}
          }
        }

        before do
          put :update, attrs
        end

        it "not return ok status" do
```

```
          expect(response).not_to be_ok
        end

        it "has error message" do
          expect(response.body).not_to be_blank
        end
      end

      context "with right code" do
        let(:attrs) {
          {
            format: :js,
            sms_auth: {otp: sms_auth.otp_secret}
          }
        }

        before do
          put :update, attrs
        end

        it { expect(response).to be_ok }
        it { expect(assigns(:sms_auth)).to be_activated }
        it { expect(member.sms_two_factor).to be_activated }
      end
    end

  end
end
```

451:F:\git\coin\exchange\peatio-master\spec\factories\account.rb

```
FactoryGirl.define do
  factory :account do
    locked { "0.0".to_d }
    balance { "100.0".to_d }
    currency :cny

    factory :account_btc do
      currency :btc
    end
  end
end
```

```
452:F:\git\coin\exchange\peatio-master\spec\factories\api_tokens.rb
# Read about factories at https://github.com/thoughtbot/factory_girl

FactoryGirl.define do
  factory :api_token do
    member
    scopes 'all'
  end
end

453:F:\git\coin\exchange\peatio-master\spec\factories\authentications.rb
# Read about factories at https://github.com/thoughtbot/factory_girl

FactoryGirl.define do
  factory :authentication do
    provider "MyString"
    uid "MyString"
    token "MyString"
    secret "MyString"
    member_id 1
  end
end

454:F:\git\coin\exchange\peatio-master\spec\factories\comments.rb
# Read about factories at https://github.com/thoughtbot/factory_girl

FactoryGirl.define do
  factory :comment do
    sequence(:content) { |n| "Content #{n}" }
    ticket
    author
  end

end

455:F:\git\coin\exchange\peatio-master\spec\factories\deposits.rb
FactoryGirl.define do
  factory :deposit do
    member { create(:member) }
    account { member.get_account(currency) }
    currency { 'btc' }
    fund_uid { Faker::Lorem.characters }
    fund_extra { Faker::Lorem.characters }
    amount { (100..10000).to_a.sample.to_d }
    txid { Faker::Lorem.characters(16) }
  end
end
```

456:F:\git\coin\exchange\peatio-master\spec\factories\documents.rb

```
# Read about factories at https://github.com/thoughtbot/factory_girl

FactoryGirl.define do
  factory :document do
  end
end
```

457:F:\git\coin\exchange\peatio-master\spec\factories\fund_source.rb

```
FactoryGirl.define do
  factory :fund_source do
    extra 'bitcoin'
    uid { Faker::Bitcoin.address }
    is_locked false
    currency 'btc'

    member { create(:member) }

    trait :cny do
      extra 'bc'
      uid '123412341234'
      currency 'cny'
    end

    factory :cny_fund_source, traits: [:cny]
    factory :btc_fund_source
  end
end
```

458:F:\git\coin\exchange\peatio-master\spec\factories\identity.rb

```
FactoryGirl.define do
  factory :identity do
    email { Faker::Internet.email }
    password { 'Password123' }
    password_confirmation { 'Password123' }
    is_active true

    trait :deactive do
      is_active false
    end
  end
end

459:F:\git\coin\exchange\peatio-master\spec\factories\id_document.rb
FactoryGirl.define do
  factory :id_document do
    name { Faker::Name.name }
    id_document_type :id_card
    id_document_number { Faker::Number.number(15).to_s }
  end
end

460:F:\git\coin\exchange\peatio-master\spec\factories\member.rb
FactoryGirl.define do
  factory :member, aliases: [:author] do
    email { Faker::Internet.email }
    phone_number { Faker::Number.number(12).to_s }

    trait :activated do
      activated true
    end

    trait :app_two_factor_activated do
      after :create do |member|
        member.app_two_factor.active!
      end
    end

    trait :sms_two_factor_activated do
      after :create do |member|
        member.sms_two_factor.active!
      end
    end

    trait :verified do
      after :create do |member|
        id_doc = member.id_document
        id_doc.update attributes_for(:id_document)
        id_doc.submit!
        id_doc.approve!
      end
    end

    trait :admin do
      after :create do |member|
        ENV['ADMIN'] = (Member.admins << member.email).join(',')
      end
    end

    factory :activated_member, traits: [:activated]
    factory :verified_member, traits: [:activated, :verified]
    factory :admin_member, traits: [:admin]
  end
end
```

461:F:\git\coin\exchange\peatio-master\spec\factories\orders.rb

```
FactoryGirl.define do
 factory :order_bid do
  bid :cny
  ask :btc
  currency :btccny
  state :wait
  source 'Web'
  ord_type 'limit'
  price { '1'.to_d }
  volume { '1'.to_d }
  origin_volume { volume }
  locked { price.to_d*volume.to_d }
  origin_locked { locked }
 end

 factory :order_ask do
  bid :cny
  ask :btc
  currency :btccny
  state :wait
  source 'Web'
  ord_type 'limit'
```

```
    price { '1'.to_d }
    volume { '1'.to_d }
    origin_volume { volume }
    locked { volume }
    origin_locked { locked }
  end
end
```

462:F:\git\coin\exchange\peatio-master\spec\factories\partial_trees.rb

```
# Read about factories at https://github.com/thoughtbot/factory_girl

FactoryGirl.define do
  factory :partial_tree do
    json "MyText"
    proof_id 1
    account_id 1
  end
end
```

463:F:\git\coin\exchange\peatio-master\spec\factories\payment_addresses.rb

```
# Read about factories at https://github.com/thoughtbot/factory_girl

FactoryGirl.define do
  factory :payment_address do
    address "MyString"
    account { create(:member).get_account(:cny) }

    trait :btc_address do
      address { Faker::Bitcoin.address }
      account { create(:member).get_account(:btc) }
      currency Currency.find_by_code('btc').id
    end

    factory :btc_payment_address, traits: [:btc_address]
  end
end
```

464:F:\git\coin\exchange\peatio-master\spec\factories\payment_transactions.rb

```
FactoryGirl.define do
  factory :payment_transaction do
    txid { Faker::Lorem.characters(16) }
    txout 0
    currency { 'btc' }
    amount { 10.to_d }
    payment_address
  end
end

465:F:\git\coin\exchange\peatio-master\spec\factories\proofs.rb
# Read about factories at https://github.com/thoughtbot/factory_girl

FactoryGirl.define do
  factory :proof do
    root "MyString"
    state "MyString"
  end
end

466:F:\git\coin\exchange\peatio-master\spec\factories\tickets.rb
# Read about factories at https://github.com/thoughtbot/factory_girl

FactoryGirl.define do
  factory :ticket do
    sequence(:content) { |n| "Content #{n}" }
    author
  end
end

467:F:\git\coin\exchange\peatio-master\spec\factories\token.rb
FactoryGirl.define do
  factory :token do
    member
  end

  factory :activation,     class: Token::Activation,    parent: :token
  factory :reset_password, class: Token::ResetPassword, parent: :token
end

468:F:\git\coin\exchange\peatio-master\spec\factories\trade.rb
FactoryGirl.define do
  factory :trade do
    price "10.0"
    volume 1
    funds {price.to_d * volume.to_d}
    currency :btccny
    association :ask, factory: :order_ask
    association :bid, factory: :order_bid
    ask_member { ask.member }
    bid_member { bid.member }
  end
end
```

469:F:\git\coin\exchange\peatio-master\spec\factories\two_factor.rb

```
FactoryGirl.define do
  factory :two_factor do
    member

    trait :activated do
      activated true
    end
  end

  factory :two_factor_app, class: TwoFactor::App, parent: :two_factor, traits: [:activated]
  factory :two_factor_sms, class: TwoFactor::Sms, parent: :two_factor, traits: [:activated]
end
```

470:F:\git\coin\exchange\peatio-master\spec\factories\withdraw.rb

```
FactoryGirl.define do
  factory :satoshi_withdraw, class: Withdraws::Satoshi do
    sum { 10.to_d }
    currency :btc
    member { create :member }
    fund_source_id { create(:btc_fund_source).id }
    type 'Withdraws::Satoshi'

    account do
      member.get_account(:btc).tap do |a|
        a.balance = 50
        a.save(validate: false)

        a.versions.create \
          balance: a.balance,
          amount: a.balance,
          locked: 0,
          fee: 0,
          currency: a.currency,
          fun: Account::FUNS[:plus_funds]
      end
    end

    after(:build) do |x|
      x.stubs(:validate_address).returns(true)
    end
  end

  factory :bank_withdraw, class: Withdraws::Bank do
    member { create :member }
    currency :cny
    sum { 1000.to_d }
    fund_source_id { create(:cny_fund_source).id }
    type 'Withdraws::Bank'

    account do
      member.get_account(:cny).tap do |a|
        a.balance = 50000
        a.save(validate: false)

        a.versions.create \
          balance: a.balance,
          amount: a.balance,
          locked: 0,
          fee: 0,
          currency: a.currency,
          fun: Account::FUNS[:plus_funds]
      end
    end
  end
end
```

471:F:\git\coin\exchange\peatio-master\spec\features\admin\withdraw_spec.rb

```
require 'spec_helper'

describe 'withdraw' do
  let!(:member) { create :member, email: identity_normal.email }
  let!(:admin_member) { create :member, email: identity.email}
  let!(:identity_normal) { create :identity }
  let!(:identity) { create :identity, email: Member.admins.first }
  let!(:account) do
    member.get_account(:cny).tap { |a| a.update_attributes locked: 8000, balance: 10000 }
  end

  let!(:withdraw) { create :bank_withdraw, member: member, sum: 5000, aasm_state: :accepted,
account: account}

  before do
    Withdraw.any_instance.stubs(:validate_password).returns(true)
  end

  def visit_admin_withdraw_page
    pending 'skip withdraw dashboard'
    login identity
    click_on I18n.t('header.admin')

    within '.ops' do
      expect(page).to have_content(I18n.t('layouts.admin.menus.items.operating.withdraws'))
      click_on I18n.t('layouts.admin.menus.items.operating.withdraws')
    end
  end

  it 'admin view withdraws' do
    pending 'skip withdraw dashboard'
    visit_admin_withdraw_page

    expect(page).to have_content(withdraw.sn)
    expect(page).to have_content(withdraw.fund_extra)
    expect(page).to_not have_content(withdraw.fund_uid)

    click_on I18n.t('actions.view')
    expect(page).to have_content(withdraw.fund_uid)
    expect(page).to have_content(withdraw.fund_extra)
    expect(page).to have_content(I18n.t('actions.transact'))
    expect(page).to have_content(I18n.t('actions.reject'))
  end

  it 'admin approve withdraw' do
    pending 'skip withdraw dashboard'
    visit_admin_withdraw_page
    click_on I18n.t('actions.view')
    click_on I18n.t('actions.transact')

    expect(current_path).to eq(admin_withdraws_path)

    click_on I18n.t('actions.view')
    click_on I18n.t('actions.transact')

    expect(current_path).to eq(admin_withdraws_path)

    expect(account.reload.locked).to be_d '3000'
    expect(account.reload.balance).to be_d '10000'
  end

  it 'admin reject withdraw' do
    pending 'skip withdraw dashboard'
    visit_admin_withdraw_page

    click_on I18n.t('actions.view')
    click_on I18n.t('actions.reject')

    expect(current_path).to eq(admin_withdraws_path)
    expect(account.reload.locked).to be_d '3000'
    expect(account.reload.balance).to be_d '15000.0000'
  end
end

472:F:\git\coin\exchange\peatio-master\spec\features\market_spec.rb
require 'spec_helper'

feature 'show account info', js: true do
  let!(:identity) { create :identity }
  let!(:member) { create :member, :activated, email: identity.email  }

  let!(:bid_account) do
    member.get_account('cny').tap { |a|
      a.plus_funds 1000
      a.save!
    }
  end
  let!(:ask_account) do
    member.get_account('btc').tap { |a|
      a.plus_funds 2000
      a.save!
    }
  end

  let!(:ask_order) { create :order_ask, price: '23.6' }
  let!(:bid_order) { create :order_bid, price: '21.3' }
  let!(:ask_name)  { 'BTC' }

  let(:global) { Global[Market.find('btccny')] }

  scenario 'user can place a buy order by filling in the order form' do
    login identity
    click_on I18n.t('header.market')

    new_window=page.driver.browser.window_handles.last
    page.within_window new_window do
      expect do
        fill_in 'order_bid_price', :with => 22.2
        fill_in 'order_bid_origin_volume', :with => 45
        expect(page.find('#order_bid_total').value).to be_d (45 * 22.2).to_d

        click_button I18n.t('private.markets.bid_entry.action', currency: ask_name)
        sleep 0.1 # sucks :(
        expect(page.find('#bid_entry span.label-success').text).to eq
I18n.t('private.markets.show.success')
      end.to change{ OrderBid.all.count }.by(1)
    end
  end

  scenario 'user can place a sell order by filling in the order form' do
    login identity
    click_on I18n.t('header.market')

    new_window=page.driver.browser.window_handles.last
    page.within_window new_window do
      expect do
        fill_in 'order_ask_price', :with => 22.2
        fill_in 'order_ask_origin_volume', :with => 45
        expect(page.find('#order_ask_total').value).to be_d (45 * 22.2).to_d

        click_button I18n.t('private.markets.ask_entry.action', currency: ask_name)
        sleep 0.1 # sucks :(
        expect(page.find('#ask_entry span.label-success').text).to eq
I18n.t('private.markets.show.success')
      end.to change{ OrderAsk.all.count }.by(1)
    end
  end

  scenario 'user can fill order form by clicking on an existing orders in the order book' do
    global.stubs(:asks).returns([[ask_order.price, ask_order.volume]])
    global.stubs(:bids).returns([[bid_order.price, bid_order.volume]])
    Global.stubs(:[]).returns(global)

    login identity
    click_on I18n.t('header.market')

    new_window=page.driver.browser.window_handles.last
    page.within_window new_window do
      page.find('.asks tr[data-order="0"]').trigger 'click'
      expect(find('#order_bid_price').value).to be_d ask_order.price
      expect(find('#order_bid_origin_volume').value).to be_d ask_order.volume
      expect(find('#order_ask_price').value).to be_d ask_order.price
      expect(find('#order_ask_origin_volume').value).to be_d ask_order.volume

      page.find('.bids tr[data-order="0"]').trigger 'click'
      expect(find('#order_ask_price').value).to be_d bid_order.price
      expect(find('#order_ask_origin_volume').value).to be_d bid_order.volume
      expect(find('#order_bid_price').value).to be_d bid_order.price
      expect(find('#order_bid_origin_volume').value).to be_d bid_order.volume
    end
  end

  scenario 'user can view his account balance' do
    login identity
    click_on I18n.t('header.market')

    new_window=page.driver.browser.window_handles.last
    page.within_window new_window do
      # account balance at place order panel
      expect(page.find('#bid_entry .current-balance').text).to be_d bid_account.balance
      expect(page.find('#ask_entry .current-balance').text).to be_d ask_account.balance
    end
  end
```

```
end

473:F:\git\coin\exchange\peatio-master\spec\features\market_trade_history_spec.rb
require 'spec_helper'

feature 'show account info', js: true do
  let(:identity) { create :identity }
  let(:other_member) { create :member }
  let(:member) { create :member, email: identity.email}
  let!(:bid_account) do
    member.get_account('cny').tap { |a| a.update_attributes locked: 400, balance: 1000 }
  end
  let!(:ask_account) do
    member.get_account('btc').tap { |a| a.update_attributes locked: 400, balance: 2000 }
  end
  let!(:ask_order) { create :order_ask, price: '23.6', member: member }
  let!(:bid_order) { create :order_bid, price: '21.3' }
  let!(:ask_name) { I18n.t('currency.name.btc') }

  scenario 'user can cancel his own order' do
    pending

    login identity
    click_on I18n.t('header.market')

    AMQPQueue.expects(:enqueue).with(:matching, action: 'cancel', order:
ask_order.to_matching_attributes)

    new_window=page.driver.browser.window_handles.last
    page.within_window new_window do
      click_link page.all('#my_order_tabs_wrapper li').first.text
      expect(page.all('#my_orders .order').count).to eq(1) # can only see his order
      expect(page).to have_selector('#my_orders .fa-trash')

      page.all('#my_orders .fa-trash').first.click
    end
  end
end

474:F:\git\coin\exchange\peatio-master\spec\features\reset_password_spec.rb
require 'spec_helper'
```

```
describe 'password' do
  let!(:identity) { create :identity }
  let!(:password) { 'New1Password' }
  let!(:member) { create :member, email: identity.email }

  it 'can be reset by user' do
    signin identity
    click_on t('private.settings.index.passwords.go')

    fill_in 'identity_old_password', with: identity.password
    fill_in 'identity_password', with: password
    fill_in 'identity_password_confirmation', with: password
    click_on t('helpers.submit.identity.update')
    expect(page).to have_content(t('identities.update.notice'))

    signin identity, password: password
    check_signin
  end
end

475:F:\git\coin\exchange\peatio-master\spec\features\sign_in_spec.rb
require 'spec_helper'

describe 'Sign in' do
  let!(:identity) { create :identity }
  let!(:member) { create :member, email: identity.email, activated: true }

  it 'allows a user to sign in with email, password' do
    signin identity
    expect(current_path).to eq(settings_path)
  end

  it 'prevents a user to sign if his account is disabled' do
    member.update_attributes disabled: true
    signin identity
    expect(current_path).to eq(signin_path)
  end

  it "sends notification email after user sign in" do
    signin identity

    mail = ActionMailer::Base.deliveries.last
    expect(mail).to be_present
    expect(mail.to).to eq([identity.email])
    expect(mail.subject).to eq(I18n.t 'member_mailer.notify_signin.subject')
  end

  context 'when a user has 2-step verification setup and after signing in with email, password' do
    let!(:member) { create :member, email: identity.email }
    let!(:two_factor) { member.app_two_factor }

    before { two_factor.refresh! }

    it 'if he tries to perform 2-step verification after session expires, should redirect user back to
login step with error message', js: true do
      pending

      signin identity
      clear_cookie

      fill_in 'two_factor_otp', with: two_factor.now
      click_on I18n.t('helpers.submit.two_factor.create')

      expect(current_path).to eq(signin_path)
      expect(page).to have_content(t('verify.two_factors.create.timeout'))
    end
  end

  it 'display captcha after too many failed attempts' do
    3.times do signin identity, password: 'wrong' end
    expect(page).not_to have_content(t('simple_form.labels.session.captcha'))

    signin identity, password: 'wrong'
    expect(page).to have_content(t('simple_form.labels.session.captcha'))

    signin identity
    signout

    signin identity, password: 'wrong'
    expect(page).not_to have_content(t('simple_form.labels.session.captcha'))
  end

end
```

```
476:F:\git\coin\exchange\peatio-master\spec\features\sign_up_spec.rb
require 'spec_helper'

describe 'Sign up', js: true do

  let(:identity) { build(:identity) }

  def fill_in_sign_up_form
    visit root_path
    click_on I18n.t('header.signup')

    within('form#new_identity') do
      fill_in 'email', with: identity.email
      fill_in 'password', with: identity.password
      fill_in 'password_confirmation', with: identity.password_confirmation
      click_on I18n.t('header.signup')
    end
  end

  def email_activation_link
    mail = ActionMailer::Base.deliveries.last
    expect(mail).to be_present
    expect(mail.to).to eq([identity.email])
    expect(mail.subject).to eq(I18n.t 'token_mailer.activation.subject')

    path = "/activations/#{Token::Activation.last.token}/edit"
    link = "#{ENV['URL_SCHEMA']}://#{ENV['URL_HOST']}#{path}"

    expect(mail.body.to_s).to have_link(link)

    path
  end

  it 'allows a user to sign up and activate the account' do
    fill_in_sign_up_form
    visit email_activation_link
    check_signin
  end

  it 'allows a user to sign up and activate the account in a different browser' do
    fill_in_sign_up_form
    clear_cookie
    visit email_activation_link
    expect(page).to have_content(t('activations.edit.notice'))

    signin identity
    check_signin
  end

  it 'allows user to resend confirmation email' do
    fill_in_sign_up_form

    first_activation_link = email_activation_link

    Timecop.travel(31.minutes.from_now)

    click_on t('private.settings.index.email.resend')

    link = email_activation_link
    expect(link).to_not eq(first_activation_link)

    visit email_activation_link
    check_signin
  end

end
```

477:F:\git\coin\exchange\peatio-master\spec\features\tag_spec.rb

```
require 'spec_helper'

describe 'member tags' do
  let!(:identity) { create :identity }
  let!(:member) { create :member, email: identity.email, tag_list: 'hero' }

  it 'user can view self tags in settings index' do
    signin identity
    expect(page).to have_content 'Hero Member'
  end
end
```

478:F:\git\coin\exchange\peatio-master\spec\features\two_factor_auth_spec.rb

```
require 'spec_helper'

describe '2-step verification' do
```

```
  let!(:identity) { create :identity }
  let!(:member) { create :member, email: identity.email }

  it 'allows user to set it up and disable it' do
    pending

    signin identity

    # enable
    within '#two_factor_auth' do
      click_on t('private.settings.index.two_factor_auth.enable')
    end

    secret = page.find('#two_factor_otp_secret').value
    fill_in 'two_factor_otp', with: ROTP::TOTP.new(secret).now
    click_on t('private.two_factors.new.submit')
    expect(page).to have_content t('private.two_factors.create.notice')

    # signin again
    signout
    signin identity, otp: ROTP::TOTP.new(secret).now

    # disable
    within '#two_factor_auth' do
      click_link t('private.settings.index.two_factor_auth.disable')
    end

    fill_in 'two_factor_otp', with: ROTP::TOTP.new(secret).now
    click_on t('private.two_factors.edit.submit')
    expect(page).to have_content t('private.two_factors.destroy.notice')

    signout
    signin identity
    check_signin
  end
end
```

479:F:\git\coin\exchange\peatio-master\spec\features\withdraw_spec.rb

```
require 'spec_helper'

describe 'withdraw' do
  let!(:identity) { create :identity }
```

```
let!(:member) { create :verified_member, email: identity.email}

let(:radio_label) do
  "#{member.name} @ #{identity.email}"
end

before do
  Withdraw.any_instance.stubs(:examine).returns(true)
  CoinRPC.any_instance.stubs(:validateaddress).returns({isvalid: true, ismine: false})

  btc_account = member.get_account(:btc)
  btc_account.update_attributes balance: 1000
  cny_account = member.get_account(:cny)
  #cny_account.update_attributes balance: 0

  @label = 'common address'
  @bank = 'bc'
  @btc_addr = create :btc_fund_source, extra: @label, uid: '1btcaddress', member: member
  @cny_addr = create :cny_fund_source, extra: @bank, uid: '1234566890', member: member
end

it 'allows user to add a BTC withdraw address, withdraw BTC' do
  pending

  login identity

  expect(page).to have_content identity.email

  visit new_withdraws_satoshi_path
  expect(page).to have_text("1000.0")

  # submit withdraw request
  submit_satoshi_withdraw_request 600

  form = find('.simple_form')
  expect(form).to have_text('600.0')
  expect(form).to have_text('0.0')

  click_on t('actions.confirm')

  expect(current_path).to eq(new_withdraws_satoshi_path)
  expect(page).to have_text(I18n.t('private.withdraws.satoshis.update.notice'))
```

```
    expect(page).to have_text("400.0")
  end

  it 'prevents withdraws that the account has no sufficient balance' do
    pending
    current_user = Member.find_by_email identity.email
    create :two_factor_sms, member: current_user

    login identity

    visit new_withdraws_bank_path

    submit_bank_withdraw_request 800
    expect(current_path).to eq(withdraws_banks_path)
    expect(page).to
have_text(I18n.t('activerecord.errors.models.withdraws/bank.attributes.sum.poor'))
  end

  private

  def submit_bank_withdraw_request amount
    select 'Bank of China', from: 'withdraw_fund_extra'
    select @bank, from: 'withdraw_fund_uid'
    fill_in 'withdraw_sum', with: amount
    click_on I18n.t 'actions.submit'
  end

  def submit_satoshi_withdraw_request amount
    select @label, from: 'withdraw_fund_uid'
    fill_in 'withdraw_fund_extra', with: @label
    fill_in 'withdraw_sum', with: amount
    click_on I18n.t 'actions.submit'
  end
end

480:F:\git\coin\exchange\peatio-master\spec\helpers\private\assets_helper_spec.rb
require 'spec_helper'

# Specs in this file have access to a helper object that includes
# the Private::AssetsHelper. For example:
#
# describe Private::AssetsHelper do
#   describe "string concat" do
#     it "concats two strings with spaces" do
#       expect(helper.concat_strings("this","that")).to eq("this that")
#     end
#   end
# end
describe Private::AssetsHelper do
  pending "add some examples to (or delete) #{__FILE__}"
end

481:F:\git\coin\exchange\peatio-master\spec\helpers\two_factor_helper_spec.rb
require 'spec_helper'

describe TwoFactorHelper do

  describe '#two_factor_locked?' do
    context 'empty session' do
      subject { helper.two_factor_locked? }

      it { should be_true }
    end

    context 'locked' do
      subject { helper.two_factor_locked? }
      before {
        session[:two_factor_locked] = false
      }

      it { should be_true }
    end

    context 'unlock without unlocked_at' do
      subject { helper.two_factor_locked?(expired_at: 5.minutes) }
      before {
        session[:two_factor_unlock] = true
      }

      it { should be_true }
    end

    context 'unlock and expired' do
      subject { helper.two_factor_locked?(expired_at: 5.minutes) }
      before {
        session[:two_factor_unlock] = true
        session[:two_factor_unlock_at] = 10.minutes.ago
      }

      it { should be_true }
    end

    context 'unlock and not expired' do
      subject { helper.two_factor_locked?(expired_at: 10.minutes) }
      before {
        session[:two_factor_unlock] = true
        session[:two_factor_unlock_at] = 5.minutes.ago
      }

      it { should_not be_true }
    end
  end

end

482:F:\git\coin\exchange\peatio-master\spec\lib\doorkeeper\access_token_spec.rb
require 'spec_helper'

describe Doorkeeper::AccessToken do

  let(:app) { Doorkeeper::Application.create!(name: 'test', uid: 'foo', secret: 'bar', redirect_uri:
'http://test.host/oauth/callback') }
  let(:member) { create(:member) }

  subject! { Doorkeeper::AccessToken.create!(application_id: app.id, resource_owner_id:
member.id, scopes: 'identity', expires_in: 1.week) }

  context "creation" do
    it "should generate corresponding api token" do
      lambda {
        Doorkeeper::AccessToken.create!(application_id: app.id, resource_owner_id: member.id,
scopes: 'identity', expires_in: 1.week)
      }.should change(APIToken, :count).by(1)
    end

    it "should prevent app requesting all scopes" do
```

```
      lambda {
        Doorkeeper::AccessToken.create!(application_id: app.id, resource_owner_id: member.id,
scopes: 'all', expires_in: 1.week)
      }.should raise_error
    end

    it "should set token" do
      subject.token.should == APIToken.last.to_oauth_token
    end

    it "should setup api token correctly" do
      api_token = APIToken.last
      api_token.label.should == app.name
      api_token.scopes.should == %w(identity)
      api_token.expire_at.should_not be_nil
    end

    it "should link api token" do
      APIToken.last.oauth_access_token.should == subject
    end
  end

  context "revoke" do
    it "should revoke access token and destroy corresponding api token" do
      subject.revoke
      subject.should be_revoked
      APIToken.find_by_id(subject.api_token.id).should be_nil
    end
  end

  context "deletion" do
    it "should soft delete record" do
      subject.destroy
      Doorkeeper::AccessToken.find_by_id(subject.id).should be_nil
      Doorkeeper::AccessToken.with_deleted.find_by_id(subject.id).should == subject
    end
  end

end
```

483:F:\git\coin\exchange\peatio-master\spec\mailers\deposit_mailer_spec.rb

```
require "spec_helper"
```

```
describe DepositMailer do

  describe "accepted" do
    let(:deposit) { create :deposit }
    let(:mail) {
      deposit.submit!
      deposit.accept!
      DepositMailer.accepted(deposit.id)
    }

    it { expect(mail).not_to be_nil }
    it { expect(mail.subject).to match "Your deposit has been credited into your account" }
  end

end

484:F:\git\coin\exchange\peatio-master\spec\mailers\member_mailer_spec.rb
require "spec_helper"

describe MemberMailer do
  describe "notify_signin" do
    let(:member) { create :member }
    let(:mail) { MemberMailer.notify_signin(member.id) }

    it "renders the headers" do
      mail.subject.should eq("[PEATIO] You have just signed in")
      mail.to.should eq([member.email])
      mail.from.should eq([ENV['SYSTEM_MAIL_FROM']])
    end

    it "renders the body" do
      mail.body.encoded.should match("signed in")
    end
  end

end

485:F:\git\coin\exchange\peatio-master\spec\mailers\withdraw_mailer_spec.rb
require "spec_helper"

describe WithdrawMailer do
  describe "withdraw_state" do
    let(:withdraw) { create :satoshi_withdraw }
    let(:mail) do
      withdraw.cancel!
      WithdrawMailer.withdraw_state(withdraw.id)
    end

    it "renders the headers" do
      mail.subject.should eq("[Peatio] Your withdraw state update")
      mail.to.should eq([withdraw.member.email])
      mail.from.should eq([ENV['SYSTEM_MAIL_FROM']])
    end

    it "renders the body" do
      mail.body.encoded.should match("canceled")
    end
  end

  describe "submitted" do
    let(:withdraw) { create :satoshi_withdraw }
    let(:mail) do
      withdraw.submit!
      WithdrawMailer.submitted(withdraw.id)
    end

    it "renders the headers" do
      mail.subject.should eq("[Peatio] Your withdraw state update")
      mail.to.should eq([withdraw.member.email])
      mail.from.should eq([ENV['SYSTEM_MAIL_FROM']])
    end

    it "renders the body" do
      mail.body.encoded.should match("submitted")
    end
  end

  describe "done" do
    let(:withdraw) { create :satoshi_withdraw }
    let(:mail) do
      withdraw.submit!
      withdraw.accept!
      withdraw.process!
```

```
      withdraw.succeed!
      WithdrawMailer.done(withdraw.id)
    end

    it "renders the headers" do
      mail.subject.should eq("[Peatio] Your withdraw state update")
      mail.to.should eq([withdraw.member.email])
      mail.from.should eq([ENV['SYSTEM_MAIL_FROM']])
    end

    it "renders the body" do
      mail.body.encoded.should match("complete")
    end
  end
end
```

486:F:\git\coin\exchange\peatio-master\spec\models\account_spec.rb

```
require 'spec_helper'

describe Account do
  subject { create(:account, locked: "10.0".to_d, balance: "10.0") }

  it { expect(subject.amount).to be_d '20' }
  it { expect(subject.sub_funds("1.0".to_d).balance).to eql "9.0".to_d }
  it { expect(subject.plus_funds("1.0".to_d).balance).to eql "11.0".to_d }
  it { expect(subject.unlock_funds("1.0".to_d).locked).to eql "9.0".to_d }
  it { expect(subject.unlock_funds("1.0".to_d).balance).to eql "11.0".to_d }
  it { expect(subject.lock_funds("1.0".to_d).locked).to eql "11.0".to_d }
  it { expect(subject.lock_funds("1.0".to_d).balance).to eql "9.0".to_d }

  it { expect(subject.unlock_and_sub_funds('1.0'.to_d, locked: '1.0'.to_d).balance).to be_d '10' }
  it { expect(subject.unlock_and_sub_funds('1.0'.to_d, locked: '1.0'.to_d).locked).to be_d '9' }

  it { expect(subject.sub_funds("0.1".to_d).balance).to eql "9.9".to_d }
  it { expect(subject.plus_funds("0.1".to_d).balance).to eql "10.1".to_d }
  it { expect(subject.unlock_funds("0.1".to_d).locked).to eql "9.9".to_d }
  it { expect(subject.unlock_funds("0.1".to_d).balance).to eql "10.1".to_d }
  it { expect(subject.lock_funds("0.1".to_d).locked).to eql "10.1".to_d }
  it { expect(subject.lock_funds("0.1".to_d).balance).to eql "9.9".to_d }

  it { expect(subject.unlock_and_sub_funds('0.1'.to_d, locked: '1.0'.to_d).balance).to be_d '10.9' }
  it { expect(subject.unlock_and_sub_funds('0.1'.to_d, locked: '1.0'.to_d).locked).to be_d '9' }
it { expect(subject.sub_funds("10.0".to_d).balance).to eql "0.0".to_d }
it { expect(subject.plus_funds("10.0".to_d).balance).to eql "20.0".to_d }
it { expect(subject.unlock_funds("10.0".to_d).locked).to eql "0.0".to_d }
it { expect(subject.unlock_funds("10.0".to_d).balance).to eql "20.0".to_d }
it { expect(subject.lock_funds("10.0".to_d).locked).to eql "20.0".to_d }
it { expect(subject.lock_funds("10.0".to_d).balance).to eql "0.0".to_d }

it { expect{subject.sub_funds("11.0".to_d)}.to raise_error }
it { expect{subject.lock_funds("11.0".to_d)}.to raise_error }
it { expect{subject.unlock_funds("11.0".to_d)}.to raise_error }

it { expect{subject.unlock_and_sub_funds('1.1'.to_d, locked: '1.0'.to_d)}.to raise_error }

it { expect{subject.sub_funds("-1.0".to_d)}.to raise_error }
it { expect{subject.plus_funds("-1.0".to_d)}.to raise_error }
it { expect{subject.lock_funds("-1.0".to_d)}.to raise_error }
it { expect{subject.unlock_funds("-1.0".to_d)}.to raise_error }
it { expect{subject.sub_funds("0".to_d)}.to raise_error }
it { expect{subject.plus_funds("0".to_d)}.to raise_error }
it { expect{subject.lock_funds("0".to_d)}.to raise_error }
it { expect{subject.unlock_funds("0".to_d)}.to raise_error }

it "expect to set reason" do
  subject.plus_funds("1.0".to_d)
  expect(subject.last_version.reason.to_sym).to eql Account::UNKNOWN
end

it "expect to set ref" do
  ref = stub(:id => 1)

  subject.plus_funds("1.0".to_d, ref: ref)

  expect(subject.last_version.modifiable_id).to eql 1
  expect(subject.last_version.modifiable_type).to eql Mocha::Mock.name
end

describe "double operation" do
  let(:strike_volume) { "10.0".to_d }
  let(:account) { create(:account) }

  it "expect double operation funds" do
```

```
      expect do
        account.plus_funds(strike_volume, reason: Account::STRIKE_ADD)
        account.sub_funds(strike_volume, reason: Account::STRIKE_FEE)
      end.to_not change{account.balance}
    end

    it "expect double operation funds to add versions" do
      expect do
        account.plus_funds(strike_volume, reason: Account::STRIKE_ADD)
        account.sub_funds(strike_volume, reason: Account::STRIKE_FEE)
      end.to change{account.reload.versions.size}.from(0).to(2)
    end
  end

  describe "#payment_address" do
    it { expect(subject.payment_address).not_to be_nil }
    it { expect(subject.payment_address).to be_is_a(PaymentAddress) }
  end

  describe "#versions" do
    let(:account) { create(:account) }

    context 'when account add funds' do
      subject { account.plus_funds("10".to_d, reason: Account::WITHDRAW).last_version }

      it { expect(subject.reason.withdraw?).to be_true }
      it { expect(subject.locked).to be_d "0" }
      it { expect(subject.balance).to be_d "10" }
      it { expect(subject.amount).to be_d "110" }
      it { expect(subject.fee).to be_d "0" }
      it { expect(subject.fun).to eq 'plus_funds' }
    end

    context 'when account add funds with fee' do
      subject { account.plus_funds("10".to_d, fee: '1'.to_d, reason:
Account::WITHDRAW).last_version }

      it { expect(subject.reason.withdraw?).to be_true }
      it { expect(subject.locked).to be_d "0" }
      it { expect(subject.balance).to be_d "10" }
      it { expect(subject.amount).to be_d "110" }
      it { expect(subject.fee).to be_d "1" }
      it { expect(subject.fun).to eq 'plus_funds' }
    end

    context 'when account sub funds' do
      subject { account.sub_funds("10".to_d, reason: Account::WITHDRAW).last_version }
      it { expect(subject.reason.withdraw?).to be_true }
      it { expect(subject.locked).to be_d "0" }
      it { expect(subject.balance).to be_d "-10" }
      it { expect(subject.amount).to be_d "90" }
      it { expect(subject.fee).to be_d "0" }
      it { expect(subject.fun).to eq 'sub_funds' }
    end

    context 'when account sub funds with fee' do
      subject { account.sub_funds("10".to_d, fee: '1'.to_d, reason:
Account::WITHDRAW).last_version }
      it { expect(subject.reason.withdraw?).to be_true }
      it { expect(subject.locked).to be_d "0" }
      it { expect(subject.balance).to be_d "-10" }
      it { expect(subject.amount).to be_d "90" }
      it { expect(subject.fee).to be_d "1" }
      it { expect(subject.fun).to eq 'sub_funds' }
    end

    context 'when account lock funds' do
      subject { account.lock_funds("10".to_d, reason: Account::WITHDRAW).last_version }
      it { expect(subject.reason.withdraw?).to be_true }
      it { expect(subject.locked).to be_d "10" }
      it { expect(subject.balance).to be_d "-10" }
      it { expect(subject.amount).to be_d "100.0" }
    end

    context 'when account unlock funds' do
      let(:account) { create(:account, locked: "10".to_d) }
      subject { account.unlock_funds("10".to_d, reason: Account::WITHDRAW).last_version }
      it { expect(subject.reason.withdraw?).to be_true }
      it { expect(subject.locked).to be_d "-10" }
      it { expect(subject.balance).to be_d "10" }
      it { expect(subject.amount).to be_d "110" }
    end

    context 'when account unlock and sub funds' do
```

```
      let(:account) { create(:account, balance: '10'.to_d, locked: "10".to_d) }
      subject { account.unlock_and_sub_funds("10".to_d, locked: "10".to_d, reason:
Account::WITHDRAW).last_version }
      it { expect(subject.reason.withdraw?).to be_true }
      it { expect(subject.locked).to be_d "-10" }
      it { expect(subject.balance).to be_d "0" }
      it { expect(subject.amount).to be_d "10.0" }
      it { expect(subject.fee).to be_d "0" }
      it { expect(subject.fun).to eq 'unlock_and_sub_funds' }
    end

    context 'when account unlock and sub funds with fee' do
      let(:account) { create(:account, balance: '10'.to_d, locked: "10".to_d) }
      subject { account.unlock_and_sub_funds("10".to_d, fee: '1'.to_d, locked: "10".to_d, reason:
Account::WITHDRAW).last_version }
      it { expect(subject.reason.withdraw?).to be_true }
      it { expect(subject.locked).to be_d "-10" }
      it { expect(subject.balance).to be_d "0" }
      it { expect(subject.amount).to be_d "10.0" }
      it { expect(subject.fee).to be_d "1" }
      it { expect(subject.fun).to eq 'unlock_and_sub_funds' }
    end
  end

  describe "#examine" do
    let(:member) { create(:member) }
    let(:account) { create(:account, locked: "0.0".to_d, balance: "0.0") }

    context "account without any account versions" do
      it "returns true" do
        expect(account.examine).to be_true
      end

      it "returns false when account changed without versions" do
        account.stubs(:member).returns(member)
        account.update_attribute(:balance, 5000.to_d)
        expect(account.examine).to be_false
      end
    end

    context "account with account versions" do
      before do
        account.plus_funds("100.0".to_d)
        account.sub_funds("1.0".to_d)
        account.plus_funds("12.0".to_d)
        account.lock_funds("12.0".to_d)
        account.unlock_funds("1.0".to_d)
        account.lock_funds("1.0".to_d)
        account.lock_funds("1.0".to_d)
      end

      it "returns true" do
        expect(account.examine).to be_true
      end

      it "returns false when account balance doesn't match versions" do
        account.stubs(:member).returns(member)
        account.update_attribute(:balance, 5000.to_d)
        expect(account.examine).to be_false
      end

      it "returns false when account versions were changed" do
        account.versions.load.sample.update_attribute(:amount, 50.to_d)
        expect(account.examine).to be_false
      end
    end
  end

  describe "#change_balance_and_locked" do
    it "should update balance and locked funds in memory" do
      subject.change_balance_and_locked "-10".to_d, "10".to_d
      subject.balance.should be_d('0')
      subject.locked.should be_d('20')
    end

    it "should update balance and locked funds in db" do
      subject.change_balance_and_locked "-10".to_d, "10".to_d
      subject.reload
      subject.balance.should be_d('0')
      subject.locked.should be_d('20')
    end
  end

  describe "after callback" do
```

```
    it "should create account version associated to account change" do
      expect {
        subject.unlock_and_sub_funds('1.0'.to_d, locked: '2.0'.to_d)
      }.to change(AccountVersion, :count).by(1)

      v = AccountVersion.last

      v.member_id.should == subject.member_id
      v.account.should   == subject
      v.fun.should       == 'unlock_and_sub_funds'
      v.reason.should    == 'unknown'
      v.amount.should    == subject.amount
      v.balance.should   == '1.0'.to_d
      v.locked.should    == '-2.0'.to_d
    end

    it "should retry the whole transaction on stale object error" do
      # `unlock_and_sub_funds('5.0'.to_d, locked: '8.0'.to_d, fee: ZERO)`
      ActiveRecord::Base.connection.execute "update accounts set balance = balance + 3, locked =
locked - 8 where id = #{subject.id}"

      expect {
        expect {
          ActiveRecord::Base.transaction do
            create(:order_ask) # any other statements should be executed
            subject.unlock_and_sub_funds('1.0'.to_d, locked: '2.0'.to_d)
          end
        }.to change(OrderAsk, :count).by(1)
      }.to change(AccountVersion, :count).by(1)

      v = AccountVersion.last
      v.amount.should  == '14.0'.to_d
      v.balance.should == '1.0'.to_d
      v.locked.should  == '-2.0'.to_d
    end
  end

  describe "concurrent lock_funds" do
    it "should raise error on the second lock_funds" do
      account1 = Account.find subject.id
      account2 = Account.find subject.id
      subject.reload.balance.should == BigDecimal.new('10')

      expect do
        ActiveRecord::Base.transaction do
          account1.lock_funds 8, reason: Account::ORDER_SUBMIT
        end
        ActiveRecord::Base.transaction do
          account2.lock_funds 8, reason: Account::ORDER_SUBMIT
        end
      end.to raise_error(ActiveRecord::RecordInvalid)

      subject.reload.balance.should == BigDecimal.new('2')
    end
  end

  describe ".enabled" do
    let!(:account1) { create(:account, currency: Currency.first.code)}
    let!(:account2) { create(:account, currency: Currency.last.code)}
    let!(:account3) { create(:account, currency: Currency.all[1].code)}
    before do
      Currency.stubs(:ids).returns([Currency.first.id, Currency.last.id])
    end

    it "should only return the accoutns with currency enabled" do
      Account.enabled.to_a.should == [account1, account2]
    end

  end

end
```

487:F:\git\coin\exchange\peatio-master\spec\models\account_version_spec.rb

```
require 'spec_helper'

describe AccountVersion do

  let(:member)  { create(:member) }
  let(:account) { member.get_account(:btc) }

  before { account.update_attributes(locked: '10.0'.to_d, balance: '10.0'.to_d) }

  context "#optimistically_lock_account_and_save!" do
```

```
    # mock AccountVersion attributes of
    # `unlock_and_sub_funds('5.0'.to_d, locked: '8.0'.to_d, fee: ZERO)`
    let(:attrs) do
      { account_id: account.id,
        fun: :unlock_and_sub_funds,
        fee: Account::ZERO,
        reason: Account::UNKNOWN,
        amount: '15.0'.to_d,
        currency: account.currency,
        member_id: account.member_id,
        locked: '-8.0'.to_d,
        balance: '3.0'.to_d }
    end

    it "should require account id" do
      attrs.delete :account_id
      expect {
        AccountVersion.optimistically_lock_account_and_create!('13.0'.to_d, '2.0'.to_d, attrs)
      }.to raise_error(ActiveRecord::ActiveRecordError)
    end

    it "should save record if associated account is fresh" do
      expect {
        # `unlock_and_sub_funds('5.0'.to_d, locked: '8.0'.to_d, fee: ZERO)`
        ActiveRecord::Base.connection.execute "update accounts set balance = balance + 3, locked
= locked - 8 where id = #{account.id}"
        AccountVersion.optimistically_lock_account_and_create!('13.0'.to_d, '2.0'.to_d, attrs)
      }.to change(AccountVersion, :count).by(1)
    end

    it "should raise StaleObjectError if associated account is stale" do
      account_in_another_thread = Account.find account.id
      account_in_another_thread.plus_funds('2.0'.to_d)

      expect {
        # `unlock_and_sub_funds('5.0'.to_d, locked: '8.0'.to_d, fee: ZERO)`
        ActiveRecord::Base.connection.execute "update accounts set balance = balance + 3, locked
= locked - 8 where id = #{account.id}"
        AccountVersion.optimistically_lock_account_and_create!('13.0'.to_d, '2.0'.to_d, attrs)
      }.to raise_error(ActiveRecord::StaleObjectError)

      expect {
        AccountVersion.optimistically_lock_account_and_create!('15.0'.to_d, '2.0'.to_d, attrs)
      }.to change(AccountVersion, :count).by(1)
    end

    it "should save associated modifiable record" do
      attrs_with_modifiable = attrs.merge(modifiable_id: 1, modifiable_type: 'OrderAsk')

      expect {
        AccountVersion.optimistically_lock_account_and_create!('10.0'.to_d, '10.0'.to_d,
attrs_with_modifiable)
      }.to change(AccountVersion, :count).by(1)
    end
  end

end

488:F:\git\coin\exchange\peatio-master\spec\models\amqp_config_spec.rb
require 'spec_helper'

module Worker
  class Test
  end
end

describe AMQPConfig do

  let(:config) do
    Hashie::Mash.new({
      connect:   { host: '127.0.0.1' },
      exchange:  { testx:  { name: 'testx', type: 'fanout' },
                   testd:  { name: 'testd', type: 'direct' },
                   topicx: { name: 'topicx', type: 'topic' } },
      queue:     { testq: { name: 'testq', durable: true } },
      binding:   {
        test:    { queue: 'testq', exchange: 'testx' },
        testd:   { queue: 'testq', exchange: 'testd' },
        topic:   { queue: 'testq', exchange: 'topicx', topics: 'test.a,test.b' },
        default: { queue: 'testq' }
      }
    })
  end
  before do
    AMQPConfig.stubs(:data).returns(config)
  end

  it "should tell client how to connect" do
    AMQPConfig.connect.should == {'host' => '127.0.0.1'}
  end

  it "should return queue settings" do
    AMQPConfig.queue(:testq).should == ['testq', {durable: true}]
  end

  it "should return exchange settings" do
    AMQPConfig.exchange(:testx).should == ['fanout', 'testx']
  end

  it "should return binding queue" do
    AMQPConfig.binding_queue(:test).should == ['testq', {durable: true}]
  end

  it "should return binding exchange" do
    AMQPConfig.binding_exchange(:test).should == ['fanout', 'testx']
  end

  it "should set exchange to nil when binding use default exchange" do
    AMQPConfig.binding_exchange(:default).should be_nil
  end

  it "should find binding worker" do
    AMQPConfig.binding_worker(:test).should be_instance_of(Worker::Test)
  end

  it "should return queue name of binding" do
    AMQPConfig.routing_key(:testd).should == 'testq'
  end

  it "should return topics to subscribe" do
    AMQPConfig.topics(:topic).should == ['test.a', 'test.b']
  end

end
```

489:F:\git\coin\exchange\peatio-master\spec\models\amqp_queue_spec.rb

```ruby
require 'spec_helper'

describe AMQPQueue do
  let(:config) do
    Hashie::Mash.new({
      connect:   { host: '127.0.0.1' },
      exchange:  { testx: { name: 'testx', type: 'fanout' } },
      queue:     { testq: { name: 'testq', durable: true },
                   testd: { name: 'testd'} },
      binding:   {
        test:    { queue: 'testq', exchange: 'testx' },
        testd:   { queue: 'testd' },
        default: { queue: 'testq' }
      }
    })
  end

  let(:default_exchange) { stub('default_exchange') }
  let(:channel) { stub('channel', default_exchange: default_exchange) }

  before do
    AMQPConfig.stubs(:data).returns(config)

    AMQPQueue.unstub(:publish)
    AMQPQueue.stubs(:exchanges).returns({default: default_exchange})
    AMQPQueue.stubs(:channel).returns(channel)
  end

  it "should instantiate exchange use exchange config" do
    channel.expects(:fanout).with('testx')
    AMQPQueue.exchange(:testx)
  end

  it "should publish message on selected exchange" do
    exchange = mock('test exchange')
    channel.expects(:fanout).with('testx').returns(exchange)
    exchange.expects(:publish).with(JSON.dump(data: 'hello'), {})
    AMQPQueue.publish(:testx, data: 'hello')
  end

  it "should publish message on default exchange" do
```

```
    default_exchange.expects(:publish).with(JSON.dump(data: 'hello', locale: I18n.locale),
routing_key: 'testd')
    AMQPQueue.enqueue(:testd, data: 'hello')
  end

end

490:F:\git\coin\exchange\peatio-master\spec\models\api_token_spec.rb
require 'spec_helper'

describe APIToken do

  let(:token) { create(:api_token, scopes: '') }

  it "should generate keys before validation on create" do
    token.access_key.size.should == 40
    token.secret_key.size.should == 40
  end

  it "should not change keys on update" do
    access_key = token.access_key
    secret_key = token.secret_key

    token.member_id = 999
    token.save && token.reload

    token.access_key.should == access_key
    token.secret_key.should == secret_key
  end

  it "should allow ip if ip filters is not set" do
    token.allow_ip?('127.0.0.1').should == true
    token.allow_ip?('127.0.0.2').should == true
  end

  it "should allow ip if ip is in ip whitelist" do
    token.trusted_ip_list = %w(127.0.0.1)
    token.allow_ip?('127.0.0.1').should == true
    token.allow_ip?('127.0.0.2').should == false
  end

  it "should tranlsate comma seperated whitelist to trusted ip list" do
```

```
    token.ip_whitelist = "127.0.0.1, 127.0.0.2,127.0.0.3"
    token.trusted_ip_list = %w(127.0.0.1 127.0.0.2 127.0.0.3)
  end

  it "should return empty array if no scopes given" do
    token.scopes.should be_empty
  end

  it "should return scopes array" do
    token.scopes = 'foo bar'
    token.scopes.should == %w(foo bar)
  end

  it "should return false if out of scope" do
    token.in_scopes?(%w(foo)).should be_false
  end

  it "should return true if in scope" do
    token.scopes = 'foo'
    token.in_scopes?(%w(foo)).should be_true
  end

  it "should return true if token has all scopes" do
    token.scopes = 'all'
    token.in_scopes?(%w(foo)).should be_true
    token.in_scopes?(%w(bar)).should be_true
  end

  it "should return true if api require no scope" do
    token.in_scopes?(nil).should be_true
    token.in_scopes?([]).should be_true
  end

  it "should destroy itself only" do
    token.destroy
    APIToken.find_by_id(token).should be_nil
  end

  it "should destroy dependent oauth access token" do
    app =Doorkeeper::Application.create!(name: 'test', uid: 'foo', secret: 'bar', redirect_uri:
'http://test.host/oauth/callback')
    access_token = Doorkeeper::AccessToken.create!(application_id: app.id, resource_owner_id:
```

```
create(:member).id, scopes: 'profile', expires_in: 1.week)

    token.update_attributes oauth_access_token_id: access_token.id
    token.destroy

    Doorkeeper::AccessToken.find_by_id(access_token).should be_nil
  end

end
```

491:F:\git\coin\exchange\peatio-master\spec\models\audit\transfer_audit_log_spec.rb

```
require 'spec_helper'

module Audit
  describe TransferAuditLog do
    describe ".audit!" do
      let(:deposit) { create(:deposit) }
      let(:member) { create(:member) }

      subject { TransferAuditLog.audit!(deposit, member) }

      before do
        deposit.stubs(:aasm_state_was).returns('submitted')
        deposit.stubs(:aasm_state).returns('accepted')
      end

      it "should create the TransferAuditLog record" do
        expect { subject }.to change{ TransferAuditLog.count }.by(1)
      end

      its(:operator) { should == member }
      its(:auditable) { should == deposit }
      its(:source_state) { should == 'submitted' }
      its(:target_state) { should == 'accepted' }

    end
  end

end
```

492:F:\git\coin\exchange\peatio-master\spec\models\bank_spec.rb

```
require 'spec_helper'
```

```
describe Bank do
  context '#with_currency' do
    it { expect(Bank.with_currency(:cny)).not_to be_empty }
  end

  context '#currency_obj' do
    subject { Bank.with_currency(:cny).first }
    its(:currency_obj) { should be_present }
  end
end
```

493:F:\git\coin\exchange\peatio-master\spec\models\comment_spec.rb

```
require 'spec_helper'

describe Comment do
  describe "#send_notification" do
    let!(:author) { create(:member, email: 'terry@apple.com') }
    let!(:admin)  { create(:member) }
    let!(:ticket) { create(:ticket, author: author) }
    let(:mailer) { mock() }
    before { mailer.stubs(:deliver) }
    after { comment.send(:send_notification) }

    context "admin reply the ticket" do
      let!(:comment) { create(:comment, author: admin, ticket: ticket)}
      it "should notify the author" do
        CommentMailer.expects(:user_notification).with(comment.id).returns(mailer)
      end
    end

    context "author reply the ticket" do
      let!(:comment) { create(:comment, author: author, ticket: ticket)}

      it "should not notify the admin" do
        CommentMailer.expects(:admin_notification).with(comment.id).returns(mailer)
      end

    end
  end
end
```

494:F:\git\coin\exchange\peatio-master\spec\models\deposit_channel_spec.rb

```ruby
require 'spec_helper'

describe DepositChannel do

  context "#sort" do
    let(:dc1) { DepositChannel.new }
    let(:dc2) { DepositChannel.new }

    it "sort DepositChannel" do
      dc1.stubs(:sort_order).returns 1
      dc2.stubs(:sort_order).returns 2
      expect([dc2, dc1].sort.first.sort_order).to eq(1)
    end
  end

end
```

495:F:\git\coin\exchange\peatio-master\spec\models\deposit_spec.rb

```ruby
require 'spec_helper'

describe Deposit do
  let(:deposit ) { create(:deposit, amount: 100.to_d) }

  it 'should compute fee' do
    expect(deposit.fee).to eql 0.to_d
    expect(deposit.amount).to eql 100.to_d
  end

  context 'when deposit fee 10%' do
    let(:deposit) { create(:deposit, amount: 100.to_d) }

    before do
      Deposit.any_instance.stubs(:calc_fee).returns([90, 10])
    end

    it 'should compute fee' do
      expect(deposit.fee).to eql 10.to_d
      expect(deposit.amount).to eql 90.to_d
    end
  end
end
```

```
496:F:\git\coin\exchange\peatio-master\spec\models\documnet_spec.rb
require 'spec_helper'

describe Document do
  describe "locale specific title setters & getters" do
    it 'sets the title in respective locales' do
      I18n.locale = :en
      d = Document.new
      d.en_title = 'Good morning!'
      d.zh_cn_title = ''

      d.save

      expect(Document.with_translations('en').last.en_title).to eq('Good morning!')
      expect(Document.with_translations('zh-CN').last.zh_cn_title).to eq('')
      expect(I18n.locale).to eq(:en)
    end
  end

  describe "locale specific body setters" do
    it 'sets the body in respective locales' do
      d = Document.new
      d.en_body = 'Good morning!'
      d.zh_cn_body = ''

      d.save

      expect(Document.with_translations('en').last.en_body).to eq('Good morning!')
      expect(Document.with_translations('zh-CN').last.zh_cn_body).to eq('')
    end
  end
end

497:F:\git\coin\exchange\peatio-master\spec\models\fund_source_spec.rb
require 'spec_helper'

describe FundSource do

  context '#label' do
    context 'for btc' do
      let(:fund_source) { build(:btc_fund_source) }
      subject { fund_source }

      its(:label) { should eq("#{fund_source.uid} (bitcoin)") }
    end

    context 'bank' do
      let(:fund_source) { build(:cny_fund_source) }
      subject { fund_source }

      its(:label) { should eq('Bank of China#****1234') }
    end
  end

end
```

498:F:\git\coin\exchange\peatio-master\spec\models\global_spec.rb

```
require 'spec_helper'

describe Global do
 let(:global) { Global['btccny'] }
end
```

499:F:\git\coin\exchange\peatio-master\spec\models\identity_spec.rb

```
require 'spec_helper'

describe Identity do
 it { should allow_value("pas1Word").for(:password) }
 it { should allow_value("pas1Wo@d").for(:password) }
 it { should allow_value("pas1Wo_d").for(:password) }
 it { should allow_value("123456").for(:password) }
 it { should_not allow_value("pwd").for(:password) }

 it "should unify email" do
   create(:identity, email: 'foo@example.com')
   build(:identity, email: 'Foo@example.com').should_not be_valid
 end

end
```

500:F:\git\coin\exchange\peatio-master\spec\models\id_document_spec.rb

```
require 'spec_helper'
```

```
describe IdDocument do
  let(:member) { create(:member) }
  subject { member.id_document }

  it { should be_valid }

  context 'aasm_state' do
    describe 'default state' do
      its(:aasm_state) { should eq('unverified') }
    end

    describe 'submit' do
      before do
        subject.submit
      end

      its(:aasm_state) { should eq('verifying') }
    end

    describe 'verified' do
      before do
        subject.submit
        subject.approve
      end

      its(:aasm_state) { should eq('verified') }
    end

    describe 'reject' do
      before do
        subject.submit
        subject.reject
      end

      its(:aasm_state) { should eq('unverified') }
    end
  end
end
```

501:F:\git\coin\exchange\peatio-master\spec\models\market_spec.rb

```
require 'spec_helper'
```

```
describe Market do

  context 'visible market' do
    # it { expect(Market.orig_all.count).to eq(2) }
    it { expect(Market.all.count).to eq(1) }
  end

  context 'markets hash' do
    it "should list all markets info" do
      Market.to_hash.should == {:btccny=>{:name=>"BTC/CNY", :base_unit=>"btc",
:quote_unit=>"cny"}}
    end
  end

  context 'market attributes' do
    subject { Market.find('btccny') }

    its(:id)         { should == 'btccny' }
    its(:name)       { should == 'BTC/CNY' }
    its(:base_unit)  { should == 'btc' }
    its(:quote_unit) { should == 'cny' }
    its(:visible)    { should be_true }
  end

  context 'enumerize' do
    subject { Market.enumerize }

    it { should be_has_key :btccny }
    it { should be_has_key :ptsbtc }
  end

  context 'shortcut of global access' do
    subject { Market.find('btccny') }

    its(:bids)   { should_not be_nil }
    its(:asks)   { should_not be_nil }
    its(:trades) { should_not be_nil }
    its(:ticker) { should_not be_nil }
  end

end
```

502:F:\git\coin\exchange\peatio-master\spec\models\matching\engine_spec.rb

```
require 'spec_helper'

describe Matching::Engine do

  let(:market) { Market.find('btccny') }
  let(:price)  { 10.to_d }
  let(:volume) { 5.to_d }
  let(:ask)    { Matching.mock_limit_order(type: :ask, price: price, volume: volume)}
  let(:bid)    { Matching.mock_limit_order(type: :bid, price: price, volume: volume)}

  let(:orderbook) { Matching::OrderBookManager.new('btccny', broadcast: false) }
  subject         { Matching::Engine.new(market, mode: :run) }
  before          { subject.stubs(:orderbook).returns(orderbook) }

  context "submit market order" do
    let!(:bid)  { Matching.mock_limit_order(type: :bid, price: '0.1'.to_d, volume: '0.1'.to_d) }
    let!(:ask1) { Matching.mock_limit_order(type: :ask, price: '1.0'.to_d, volume: '1.0'.to_d) }
    let!(:ask2) { Matching.mock_limit_order(type: :ask, price: '2.0'.to_d, volume: '1.0'.to_d) }
    let!(:ask3) { Matching.mock_limit_order(type: :ask, price: '3.0'.to_d, volume: '1.0'.to_d) }

    it "should fill the market order completely" do
      mo = Matching.mock_market_order(type: :bid, locked: '6.0'.to_d, volume: '2.4'.to_d)

      AMQPQueue.expects(:enqueue).with(:trade_executor, {market_id: market.id, ask_id: ask1.id,
bid_id: mo.id, strike_price: ask1.price, volume: ask1.volume, funds: '1.0'.to_d}, anything)
      AMQPQueue.expects(:enqueue).with(:trade_executor, {market_id: market.id, ask_id: ask2.id,
bid_id: mo.id, strike_price: ask2.price, volume: ask2.volume, funds: '2.0'.to_d}, anything)
      AMQPQueue.expects(:enqueue).with(:trade_executor, {market_id: market.id, ask_id: ask3.id,
bid_id: mo.id, strike_price: ask3.price, volume: '0.4'.to_d, funds: '1.2'.to_d}, anything)

      subject.submit bid
      subject.submit ask1
      subject.submit ask2
      subject.submit ask3
      subject.submit mo

      subject.ask_orders.limit_orders.should have(1).price_level
      subject.ask_orders.limit_orders.values.first.should == [ask3]
      ask3.volume.should == '0.6'.to_d
      subject.bid_orders.market_orders.should be_empty
    end

    it "should fill the market order partially and cancel it" do
      mo = Matching.mock_market_order(type: :bid, locked: '6.0'.to_d, volume: '2.4'.to_d)

      AMQPQueue.expects(:enqueue).with(:trade_executor, {market_id: market.id, ask_id: ask1.id,
bid_id: mo.id, strike_price: ask1.price, volume: ask1.volume, funds: '1.0'.to_d}, anything)
      AMQPQueue.expects(:enqueue).with(:trade_executor, {market_id: market.id, ask_id: ask2.id,
bid_id: mo.id, strike_price: ask2.price, volume: ask2.volume, funds: '2.0'.to_d}, anything)
      AMQPQueue.expects(:enqueue).with(:order_processor, has_entries(action: 'cancel', order:
has_entry(id: mo.id)), anything)

      subject.submit bid
      subject.submit ask1
      subject.submit ask2
      subject.submit mo

      subject.ask_orders.limit_orders.should be_empty
      subject.bid_orders.market_orders.should be_empty
    end

    it "should partially fill then cancel the market order if locked funds run out" do
      mo = Matching.mock_market_order(type: :bid, locked: '2.5'.to_d, volume: '2'.to_d)

      AMQPQueue.expects(:enqueue).with(:trade_executor, {market_id: market.id, ask_id: ask1.id,
bid_id: mo.id, strike_price: ask1.price, volume: ask1.volume, funds: '1.0'.to_d}, anything)
      AMQPQueue.expects(:enqueue).with(:trade_executor, {market_id: market.id, ask_id: ask2.id,
bid_id: mo.id, strike_price: ask2.price, volume: '0.75'.to_d, funds: '1.5'.to_d}, anything)

      subject.submit bid
      subject.submit ask1
      subject.submit ask2
      subject.submit ask3
      subject.submit mo

      subject.ask_orders.limit_orders.should have(2).price_level
      ask2.volume.should == '0.25'.to_d
      ask3.volume.should == '1.0'.to_d

      subject.bid_orders.market_orders.should be_empty
    end
```

```
  end

  context "submit limit order" do
    context "fully match incoming order" do
      it "should execute trade" do
        AMQPQueue.expects(:enqueue)
        .with(:trade_executor, {market_id: market.id, ask_id: ask.id, bid_id: bid.id, strike_price: price,
volume: volume, funds: '50.0'.to_d}, anything)

        subject.submit(ask)
        subject.submit(bid)

        subject.ask_orders.limit_orders.should be_empty
        subject.bid_orders.limit_orders.should be_empty
      end
    end

    context "partial match incoming order" do
      let(:ask) { Matching.mock_limit_order(type: :ask, price: price, volume: 3.to_d)}

      it "should execute trade" do
        AMQPQueue.expects(:enqueue)
          .with(:trade_executor, {market_id: market.id, ask_id: ask.id, bid_id: bid.id, strike_price: price,
volume: 3.to_d, funds: '30.0'.to_d}, anything)

        subject.submit(ask)
        subject.submit(bid)

        subject.ask_orders.limit_orders.should be_empty
        subject.bid_orders.limit_orders.should_not be_empty

        AMQPQueue.expects(:enqueue)
          .with(:order_processor, {action: 'cancel', order: bid.attributes}, anything)
        subject.cancel(bid)
        subject.bid_orders.limit_orders.should be_empty
      end
    end

    context "match order with many counter orders" do
      let(:bid)    { Matching.mock_limit_order(type: :bid, price: price, volume: 10.to_d)}

      let(:asks) do
```

```
        [nil,nil,nil].map do
          Matching.mock_limit_order(type: :ask, price: price, volume: 3.to_d)
        end
      end

      it "should execute trade" do
        AMQPQueue.expects(:enqueue).times(asks.size)

        asks.each {|ask| subject.submit(ask) }
        subject.submit(bid)

        subject.ask_orders.limit_orders.should be_empty
        subject.bid_orders.limit_orders.should_not be_empty
      end
    end

    context "fully match order after some cancellatons" do
      let(:bid)      { Matching.mock_limit_order(type: :bid, price: price,   volume: 10.to_d)}
      let(:low_ask)  { Matching.mock_limit_order(type: :ask, price: price-1, volume: 3.to_d) }
      let(:high_ask) { Matching.mock_limit_order(type: :ask, price: price,   volume: 3.to_d) }

      it "should match bid with high ask" do
        subject.submit(low_ask) # low ask enters first
        subject.submit(high_ask)
        subject.cancel(low_ask) # but it's cancelled

        AMQPQueue.expects(:enqueue)
        .with(:trade_executor, {market_id: market.id, ask_id: high_ask.id, bid_id: bid.id, strike_price:
high_ask.price, volume: high_ask.volume, funds: '30.0'.to_d}, anything)
        subject.submit(bid)

        subject.ask_orders.limit_orders.should be_empty
        subject.bid_orders.limit_orders.should_not be_empty
      end
    end
  end

  context "#cancel" do
    it "should cancel order" do
      subject.submit(ask)
      subject.cancel(ask)
      subject.ask_orders.limit_orders.should be_empty
      subject.submit(bid)
      subject.cancel(bid)
      subject.bid_orders.limit_orders.should be_empty
    end
  end

  context "float number edge cases" do
    it "should add up used funds to locked funds" do
      order = create(:order_bid, price: '3662.05', volume: '0.62')
      bid  = Matching.mock_limit_order(order.to_matching_attributes)

      ask1 = Matching.mock_limit_order(type: :ask, price: '3658.28'.to_d, volume: '0.0129'.to_d)
      ask2 = Matching.mock_limit_order(type: :ask, price: '3661.72'.to_d, volume: '0.26'.to_d)
      ask3 = Matching.mock_limit_order(type: :ask, price: '3659.00'.to_d, volume: '0.2945'.to_d)
      ask4 = Matching.mock_limit_order(type: :ask, price: '3661.68'.to_d, volume: '0.0526'.to_d)

      used_funds = 0
      subject.stubs(:publish).with do |order, counter_order, trade|
        price, volume, funds = trade
        used_funds += funds
      end

      subject.submit bid
      subject.submit ask1
      subject.submit ask2
      subject.submit ask3
      subject.submit ask4

      used_funds.should ==  order.compute_locked
    end
  end

  context "dryrun" do
    subject { Matching::Engine.new(market, mode: :dryrun) }

    it "should not publish matched trades" do
      AMQPQueue.expects(:enqueue).never

      subject.submit(ask)
      subject.submit(bid)
```

```
      subject.ask_orders.limit_orders.should be_empty
      subject.bid_orders.limit_orders.should be_empty

      subject.queue.should have(1).trade
      subject.queue.first.should == [:trade_executor, {market_id: market.id, ask_id: ask.id, bid_id:
bid.id, strike_price: price, volume: volume, funds: '50.0'.to_d}, {persistent: false}]
    end
  end

end

503:F:\git\coin\exchange\peatio-master\spec\models\matching\executor_spec.rb
require 'spec_helper'

describe Matching::Executor do

  let(:alice)  { who_is_billionaire }
  let(:bob)    { who_is_billionaire }
  let(:market) { Market.find('btccny') }
  let(:price)  { 10.to_d }
  let(:volume) { 5.to_d }

  subject {
    Matching::Executor.new(
      market_id:    market.id,
      ask_id:       ask.id,
      bid_id:       bid.id,
      strike_price: price.to_s('F'),
      volume:       volume.to_s('F'),
      funds:        (price*volume).to_s('F')
    )
  }

  context "invalid volume" do
    let(:ask) { ::Matching::LimitOrder.new create(:order_ask, price: price, volume: volume, member:
alice).to_matching_attributes }
    let(:bid) { ::Matching::LimitOrder.new create(:order_bid, price: price, volume: 3.to_d, member:
bob).to_matching_attributes }

    it "should raise error" do
      expect { subject.execute! }.to raise_error(Matching::TradeExecutionError)
    end
  end

  context "invalid price" do
    let(:ask) { ::Matching::LimitOrder.new create(:order_ask, price: price, volume: volume, member:
alice).to_matching_attributes }
    let(:bid) { ::Matching::LimitOrder.new create(:order_bid, price: price-1, volume: volume, member:
bob).to_matching_attributes }

    it "should raise error" do
      expect { subject.execute! }.to raise_error(Matching::TradeExecutionError)
    end
  end

  context "full execution" do
    let(:ask) { ::Matching::LimitOrder.new create(:order_ask, price: price, volume: volume, member:
alice).to_matching_attributes }
    let(:bid) { ::Matching::LimitOrder.new create(:order_bid, price: price, volume: volume, member:
bob).to_matching_attributes }

    it "should create trade" do
      expect {
        trade = subject.execute!

        trade.trend.should  == 'up'
        trade.price.should  == price
        trade.volume.should == volume
        trade.ask_id.should == ask.id
        trade.bid_id.should == bid.id
      }.to change(Trade, :count).by(1)
    end

    it "should set trend to down" do
      market.expects(:latest_price).returns(11.to_d)
      trade = subject.execute!

      trade.trend.should == 'down'
    end

    it "should set trade used funds" do
      market.expects(:latest_price).returns(11.to_d)
      trade = subject.execute!
      trade.funds.should == price*volume
    end

    it "should increase order's trades count" do
      subject.execute!
      Order.find(ask.id).trades_count.should == 1
      Order.find(bid.id).trades_count.should == 1
    end

    it "should mark both orders as done" do
      subject.execute!

      Order.find(ask.id).state.should == Order::DONE
      Order.find(bid.id).state.should == Order::DONE
    end

    it "should publish trade through amqp" do
      AMQPQueue.expects(:publish)
      subject.execute!
    end
  end

  context "partial ask execution" do
    let(:ask) { create(:order_ask, price: price, volume: 7.to_d, member: alice) }
    let(:bid) { create(:order_bid, price: price, volume: 5.to_d, member: bob) }

    it "should set bid to done only" do
      subject.execute!

      ask.reload.state.should_not == Order::DONE
      bid.reload.state.should == Order::DONE
    end
  end

  context "partial bid execution" do
    let(:ask) { create(:order_ask, price: price, volume: 5.to_d, member: alice) }
    let(:bid) { create(:order_bid, price: price, volume: 7.to_d, member: bob) }

    it "should set ask to done only" do
      subject.execute!

      ask.reload.state.should == Order::DONE
      bid.reload.state.should_not == Order::DONE
    end
  end

  context "partially filled market order whose locked fund run out" do
    let(:ask) { create(:order_ask, price: '2.0'.to_d, volume: '3.0'.to_d, member: alice) }
    let(:bid) { create(:order_bid, price: nil, ord_type: 'market', volume: '2.0'.to_d, locked: '3.0'.to_d,
member: bob) }

    it "should cancel the market order" do
      executor = Matching::Executor.new(
        market_id:    market.id,
        ask_id:       ask.id,
        bid_id:       bid.id,
        strike_price: '2.0',
        volume:       '1.5',
        funds:        '3.0'
      )
      executor.execute!

      bid.reload.state.should == Order::CANCEL
    end
  end

  context "unlock not used funds" do
    let(:ask) { create(:order_ask, price: price-1, volume: 7.to_d, member: alice) }
    let(:bid) { create(:order_bid, price: price, volume: volume, member: bob) }

    subject {
      Matching::Executor.new(
        market_id:    market.id,
        ask_id:       ask.id,
        bid_id:       bid.id,
        strike_price: price-1, # so bid order only used (price-1)*volume
        volume:       volume.to_s('F'),
        funds:        ((price-1)*volume).to_s('F')
      )
    }

    it "should unlock funds not used by bid order" do
      locked_before = bid.hold_account.reload.locked

      subject.execute!
      locked_after = bid.hold_account.reload.locked

      locked_after.should == locked_before - (price*volume)
    end

    it "should save unused amount in order locked attribute" do
      subject.execute!
      bid.reload.locked.should == price*volume - (price-1)*volume
    end
  end

  context "execution fail" do
    let(:ask) { ::Matching::LimitOrder.new create(:order_ask, price: price, volume: volume, member:
alice).to_matching_attributes }
    let(:bid) { ::Matching::LimitOrder.new create(:order_bid, price: price, volume: volume, member:
bob).to_matching_attributes }

    it "should not create trade" do
      # set locked funds to 0 so strike will fail
      alice.get_account(:btc).update_attributes(locked: ::Trade::ZERO)

      expect do
        expect { subject.execute! }.to raise_error(Account::LockedError)
      end.not_to change(Trade, :count)
    end
  end

end

504:F:\git\coin\exchange\peatio-master\spec\models\matching\limit_order_spec.rb
require 'spec_helper'

describe Matching::LimitOrder do

  context "initialize" do
    it "should throw invalid order error for empty attributes" do
      expect {
        Matching::LimitOrder.new({type: '', price: '', volume: ''})
      }.to raise_error(Matching::InvalidOrderError)
    end

    it "should initialize market" do
```

```
        Matching.mock_limit_order(type: :bid).market.should be_instance_of(Market)
      end
    end

    context "crossed?" do
      it "should cross at lower or equal price for bid order" do
        order = Matching.mock_limit_order(type: :bid, price: '10.0'.to_d)
        order.crossed?('9.0'.to_d).should be_true
        order.crossed?('10.0'.to_d).should be_true
        order.crossed?('11.0'.to_d).should be_false
      end

      it "should cross at higher or equal price for ask order" do
        order = Matching.mock_limit_order(type: :ask, price: '10.0'.to_d)
        order.crossed?('9.0'.to_d).should be_false
        order.crossed?('10.0'.to_d).should be_true
        order.crossed?('11.0'.to_d).should be_true
      end
    end
  end

  505:F:\git\coin\exchange\peatio-master\spec\models\matching\market_order_spec.rb
  require 'spec_helper'

  describe Matching::MarketOrder do

    context "initialize" do
      it "should not allow price attribute" do
        expect { Matching.mock_market_order(type: :ask, price: '1.0'.to_d) }.to raise_error
      end

      it "should only accept positive sum limit" do
        expect { Matching.mock_market_order(type: :bid, locked: '0.0'.to_d) }.to raise_error
      end
    end

    context "#fill" do
      subject { Matching.mock_market_order(type: :bid, locked: '10.0'.to_d, volume: '2.0'.to_d) }

      it "should raise not enough volume error" do
        expect { subject.fill('1.0'.to_d, '3.0'.to_d, '3.0'.to_d) }.to
  raise_error(Matching::NotEnoughVolume)
```

```
    end

    it "should raise sum limit reached error" do
      expect { subject.fill('11.0'.to_d, '1.0'.to_d, '11.0'.to_d) }.to
raise_error(Matching::ExceedSumLimit)
    end

    it "should also decrease volume and sum limit" do
      subject.fill '6.0'.to_d, '1.0'.to_d, '6.0'.to_d
      subject.volume.should == '1.0'.to_d
      subject.locked.should == '4.0'.to_d
    end
  end

end
```

506:F:\git\coin\exchange\peatio-master\spec\models\matching\order_book_manager_spec.rb

```
require 'spec_helper'

describe Matching::OrderBookManager do

  context ".build_order" do
    it "should build limit order" do
      order = ::Matching::OrderBookManager.build_order id: 1, market: 'btccny', ord_type: 'limit',
type: 'ask', price: '1.0', volume: '1.0', timestamp: 12345
      order.should be_instance_of(::Matching::LimitOrder)
    end
  end

end
```

507:F:\git\coin\exchange\peatio-master\spec\models\matching\order_book_spec.rb

```
require 'spec_helper'

describe Matching::OrderBook do

  context "#find" do
    subject { Matching::OrderBook.new('btccny', :ask) }

    it "should find specific order" do
      o1 = Matching.mock_limit_order(type: :ask, price: '1.0'.to_d)
      o2 = Matching.mock_limit_order(type: :ask, price: '1.0'.to_d)
      subject.add o1
      subject.add o2

      subject.find(o1.dup).object_id.should == o1.object_id
      subject.find(o2.dup).object_id.should == o2.object_id
    end
  end

  context "#add" do
    subject { Matching::OrderBook.new('btccny', :ask) }

    it "should reject invalid order whose volume is zero" do
      expect {
        subject.add Matching.mock_limit_order(type: :ask, volume: '0.0'.to_d)
      }.to raise_error(::Matching::InvalidOrderError)
    end

    it "should add market order" do
      subject.add Matching.mock_limit_order(type: :ask)

      o1 = Matching.mock_market_order(type: :ask)
      o2 = Matching.mock_market_order(type: :ask)
      o3 = Matching.mock_market_order(type: :ask)
      subject.add o1
      subject.add o2
      subject.add o3

      subject.market_orders.should == [o1, o2, o3]
    end

    it "should create price level for order with new price" do
      order = Matching.mock_limit_order(type: :ask)
      subject.add order
      subject.limit_orders.keys.first.should == order.price
      subject.limit_orders.values.first.should == [order]
    end

    it "should add order with same price to same price level" do
      o1 = Matching.mock_limit_order(type: :ask)
      o2 = Matching.mock_limit_order(type: :ask, price: o1.price)
      subject.add o1
      subject.add o2
      subject.limit_orders.keys.should have(1).price_level
      subject.limit_orders.values.first.should == [o1, o2]
    end

    it "should broadcast add event" do
      order = Matching.mock_limit_order(type: :ask)

      AMQPQueue.expects(:enqueue).with(:slave_book, {action: 'new', market: 'btccny', side: :ask},
{persistent: false})
      AMQPQueue.expects(:enqueue).with(:slave_book, {action: 'add', order: order.attributes},
{persistent: false})
      subject.add order
    end

    it "should not broadcast add event" do
      order = Matching.mock_limit_order(type: :ask)

      AMQPQueue.expects(:enqueue).with(:slave_book, {action: 'add', order: order.attributes},
{persistent: false}).never
      Matching::OrderBook.new('btccny', :ask, broadcast: false).add order
    end
  end

  context "#remove" do
    subject { Matching::OrderBook.new('btccny', :ask) }

    it "should remove market order" do
      subject.add Matching.mock_limit_order(type: :ask)
      order = Matching.mock_market_order(type: :ask)
      subject.add order
      subject.remove order
      subject.market_orders.should be_empty
    end

    it "should remove limit order" do
      o1 = Matching.mock_limit_order(type: :ask, price: '1.0'.to_d)
      o2 = Matching.mock_limit_order(type: :ask, price: '1.0'.to_d)
      subject.add o1
      subject.add o2
      subject.remove o1.dup # dup so it's not the same object, but has same id
    subject.limit_orders.values.first.should have(1).order
  end

  it "should remove price level if its only limit order removed" do
    order = Matching.mock_limit_order(type: :ask)
    subject.add order
    subject.remove order.dup
    subject.limit_orders.should be_empty
  end

  it "should return nil if order is not found" do
    order = Matching.mock_limit_order(type: :ask)
    subject.remove(order).should be_nil
  end

  it "should return order in book" do
    o1 = Matching.mock_limit_order(type: :ask, price: '1.0'.to_d)
    o2 = o1.dup
    o1.volume = '12345'.to_d
    subject.add o1
    o = subject.remove o2
    o.volume.should == '12345'.to_d
  end
end

context "#best_limit_price" do
  it "should return highest bid price" do
    book = Matching::OrderBook.new('btccny', :bid)
    o1   = Matching.mock_limit_order(type: :bid, price: '1.0'.to_d)
    o2   = Matching.mock_limit_order(type: :bid, price: '2.0'.to_d)
    book.add o1
    book.add o2

    book.best_limit_price.should == o2.price
  end

  it "should return lowest ask price" do
    book = Matching::OrderBook.new('btccny', :ask)
    o1   = Matching.mock_limit_order(type: :ask, price: '1.0'.to_d)
    o2   = Matching.mock_limit_order(type: :ask, price: '2.0'.to_d)
    book.add o1
    book.add o2
    book.best_limit_price.should == o1.price
  end

  it "should return nil if there's no limit order" do
    book = Matching::OrderBook.new('btccny', :ask)
    book.best_limit_price.should be_nil
  end
end

context "#top" do
  it "should return market order if there's any market order" do
    book = Matching::OrderBook.new('btccny', :ask)
    o1 = Matching.mock_limit_order(type: :ask)
    o2 = Matching.mock_market_order(type: :ask)
    book.add o1
    book.add o2

    book.top.should == o2
  end

  it "should return nil for empty book" do
    book = Matching::OrderBook.new('btccny', :ask)
    book.top.should be_nil
  end

  it "should find ask order with lowest price" do
    book = Matching::OrderBook.new('btccny', :ask)
    o1 = Matching.mock_limit_order(type: :ask, price: '1.0'.to_d)
    o2 = Matching.mock_limit_order(type: :ask, price: '2.0'.to_d)
    book.add o1
    book.add o2

    book.top.should == o1
  end

  it "should find bid order with highest price" do
    book = Matching::OrderBook.new('btccny', :bid)
    o1 = Matching.mock_limit_order(type: :bid, price: '1.0'.to_d)
    o2 = Matching.mock_limit_order(type: :bid, price: '2.0'.to_d)
    book.add o1
    book.add o2
    book.top.should == o2
  end

  it "should favor earlier order if orders have same price" do
    book = Matching::OrderBook.new('btccny', :ask)
    o1 = Matching.mock_limit_order(type: :ask, price: '1.0'.to_d)
    o2 = Matching.mock_limit_order(type: :ask, price: '1.0'.to_d)
    book.add o1
    book.add o2

    book.top.should == o1
  end
end

context "#fill_top" do
  subject { Matching::OrderBook.new('btccny', :ask) }

  it "should raise error if there is no top order" do
    expect { subject.fill_top '1.0'.to_d, '1.0'.to_d, '1.0'.to_d }.to raise_error
  end

  it "should complete fill the top market order" do
    subject.add Matching.mock_limit_order(type: :ask, volume: '1.0'.to_d)
    subject.add Matching.mock_market_order(type: :ask, volume: '1.0'.to_d)
    subject.fill_top '1.0'.to_d, '1.0'.to_d, '1.0'.to_d
    subject.market_orders.should be_empty
    subject.limit_orders.should have(1).order
  end

  it "should partial fill the top market order" do
    subject.add Matching.mock_limit_order(type: :ask, volume: '1.0'.to_d)
    subject.add Matching.mock_market_order(type: :ask, volume: '1.0'.to_d)
    subject.fill_top '1.0'.to_d, '0.6'.to_d, '0.6'.to_d
    subject.market_orders.first.volume.should == '0.4'.to_d
    subject.limit_orders.should have(1).order
  end

  it "should remove the price level if top order is the only order in level" do
    subject.add Matching.mock_limit_order(type: :ask, volume: '1.0'.to_d)
    subject.fill_top '1.0'.to_d, '1.0'.to_d, '1.0'.to_d
    subject.limit_orders.should be_empty
    end

    it "should remove order from level" do
      subject.add Matching.mock_limit_order(type: :ask, volume: '1.0'.to_d)
      subject.add Matching.mock_limit_order(type: :ask, volume: '1.0'.to_d)
      subject.fill_top '1.0'.to_d, '1.0'.to_d, '1.0'.to_d
      subject.limit_orders.values.first.should have(1).order
    end

    it "should fill top order with volume" do
      subject.add Matching.mock_limit_order(type: :ask, volume: '2.0'.to_d)
      subject.fill_top '1.0'.to_d, '0.5'.to_d, '0.5'.to_d
      subject.top.volume.should == '1.5'.to_d
    end
  end

end
```

508:F:\git\coin\exchange\peatio-master\spec\models\matching\price_level_spec.rb

```
require 'spec_helper'

describe Matching::PriceLevel do

  subject  { Matching::PriceLevel.new('1.0'.to_d) }
  let(:o1) { Matching.mock_limit_order(type: :ask) }
  let(:o2) { Matching.mock_limit_order(type: :ask) }
  let(:o3) { Matching.mock_limit_order(type: :ask) }

  before do
    subject.add o1
    subject.add o2
    subject.add o3
  end

  it "should remove order" do
    subject.remove o2
    subject.orders.should == [o1, o3]
  end

  it "should find order by id" do
    subject.find(o1.id).should == o1
    subject.find(o2.id).should == o2
  end
end

509:F:\git\coin\exchange\peatio-master\spec\models\member_spec.rb
require 'spec_helper'

describe Member do
  let(:member) { build(:member) }
  subject { member }

  describe 'sn' do
    subject(:member) { create(:member) }
    it { expect(member.sn).to_not be_nil }
    it { expect(member.sn).to_not be_empty }
    it { expect(member.sn).to match /^PEA.*TIO$/ }
  end

  describe 'before_create' do
    it "should unify email" do
      create(:identity, email: 'foo@example.com')
      build(:identity, email: 'Foo@example.com').should_not be_valid
    end

    it 'creates accounts for the member' do
      expect {
        member.save!
      }.to change(member.accounts, :count).by(Currency.codes.size)

      Currency.codes.each do |code|
        expect(Account.with_currency(code).where(member_id: member.id).count).to eq 1
      end
    end
  end

  describe 'build id_document before create' do
    it 'create id_document for the member' do
      member.save
      expect(member.reload.id_document).to_not be_blank
    end
  end

  describe 'send activation after create' do
    let(:auth_hash) {
      {
        'provider' => 'identity',
        'info' => { 'email' => 'foobar@peatio.dev' }
      }
    }

    it 'create activation' do
      expect {
        Member.from_auth(auth_hash)
      }.to change(Token::Activation, :count).by(1)
    end
  end

  describe '#send_password_changed_notification' do
    let(:member) { create :member }

    before do
      member.send_password_changed_notification
      @mail = ActionMailer::Base.deliveries.last
    end

    it { expect(ActionMailer::Base.deliveries).not_to be_empty }
    it { expect(@mail.subject).to match "Your password changed" }
  end

  describe '#trades' do
    subject { create(:member) }

    it "should find all trades belong to user" do
      ask = create(:order_ask, member: member)
      bid = create(:order_bid, member: member)
      t1 = create(:trade, ask: ask)
      t2 = create(:trade, bid: bid)
      member.trades.order('id').should == [t1, t2]
    end
  end

  describe ".current" do
    let(:member) { create(:member) }
    before do
      Thread.current[:user] = member
    end

    after do
      Thread.current[:user] = nil
    end

    specify { Member.current.should == member }
  end

  describe ".current=" do
    let(:member) { create(:member) }
    before { Member.current = member }
    after { Member.current = nil }
    specify { Thread.current[:user].should == member }
  end

  describe "#unread_messages" do
    let!(:user) { create(:member) }

    let!(:ticket) { create(:ticket, author: user) }
    let!(:comment) { create(:comment, ticket: ticket) }

    before { ReadMark.delete_all }

    specify { user.unread_comments.count.should == 1 }

  end

  describe "#identity" do
    it "should not raise but return nil when authentication is not found" do
      member = create(:member)
      expect(member.identity).to be_nil
    end
  end

  describe 'Member.search' do
    before do
      create(:member)
      create(:member)
      create(:member)
    end
```

```
describe 'search without any condition' do
  subject { Member.search(field: nil, term: nil) }

  it { expect(subject.count).to eq(3) }
end

describe 'search by email' do
  let(:member) { create(:member) }
  subject { Member.search(field: 'email', term: member.email) }

  it { expect(subject.count).to eq(1) }
  it { expect(subject).to be_include(member) }
end

describe 'search by phone number' do
  let(:member) { create(:member) }
  subject { Member.search(field: 'phone_number', term: member.phone_number) }

  it { expect(subject.count).to eq(1) }
  it { expect(subject).to be_include(member) }
end

describe 'search by name' do
  let(:member) { create(:verified_member) }
  subject { Member.search(field: 'name', term: member.name) }

  it { expect(subject.count).to eq(1) }
  it { expect(subject).to be_include(member) }
end

describe 'search by wallet address' do
  let(:fund_source) { create(:btc_fund_source) }
  let(:member) { fund_source.member }
  subject { Member.search(field: 'wallet_address', term: fund_source.uid) }

  it { expect(subject.count).to eq(1) }
  it { expect(subject).to be_include(member) }
end

describe 'search by deposit address' do
  let(:payment_address) { create(:btc_payment_address) }
  let(:member) { payment_address.account.member }
```

```
      subject { Member.search(field: 'wallet_address', term: payment_address.address) }

      it { expect(subject.count).to eq(1) }
      it { expect(subject).to be_include(member) }
    end
  end

  describe "#create_auth_for_identity" do
    let(:identity) { create(:identity) }
    let(:member) { create(:member, email: identity.email) }

    it "should create the authentication" do
      expect do
        member.create_auth_for_identity(identity)
      end.to change(Identity, :count).by(1)
    end
  end

  describe "#remove_auth" do
    let!(:identity) { create(:identity) }
    let!(:member) { create(:member, email: identity.email) }
    let!(:weibo_auth) { create(:authentication, provider: 'weibo', member_id: member.id)}
    let!(:identity_auth) { create(:authentication, provider: 'identity', member_id: member.id, uid:
identity.id)}

    context "third party" do
      it "should delete the weibo auth" do
        expect do
          expect do
            member.remove_auth('weibo')
          end.not_to change(Identity, :count)
        end.to change(Authentication, :count).by(-1)
        member.auth('weibo').should be_nil
      end
    end

    context "identity" do
      it "should delete the ideneity auth and the identity" do
        expect do
          expect do
            member.remove_auth('identity')
          end.to change(Identity, :count).by(-1)
      end.to change(Authentication, :count).by(-1)
      member.auth('identity').should be_nil
    end
  end
end

describe "#locate_email" do
  context "Email is blank" do
    let!(:member) { create(:member, email: nil) }
    let(:auth) {
      {'info' => { 'email' => nil}}
    }

    it "should return nil" do
      Member.count.should == 1
      Member.send(:locate_email, auth).should be_nil
    end
  end

  context "Emails is exist and can find member" do
    let(:email) { 'fuck@chinese.gov' }
    let!(:member) { create(:member, email: email) }
    let(:auth) {
      { 'provider' => 'weibo', 'uid' => 'hehe', 'info' => { 'email' => email} }
    }

    it "should return the user and create the auth" do
      expect do
        Member.send(:locate_email, auth).should == member
      end.to change(Authentication, :count).by(1)
    end
  end

  context "Email is exist but can not find member" do
    let(:email) { 'fuck@chinese.gov' }
    let!(:member) { create(:member, email: email) }

    let(:auth) {
      { 'provider' => 'weibo', 'uid' => 'hehe', 'info' => { 'email' => email + 'veryhard'} }
    }

    it "should not create auth and return nil" do
```

```
        expect do
          Member.send(:locate_email, auth).should be_nil
        end.not_to change(Authentication, :count)
      end
    end
  end


end

510:F:\git\coin\exchange\peatio-master\spec\models\order_ask_spec.rb
require 'spec_helper'

describe OrderAsk do

  subject { create(:order_ask) }

  its(:compute_locked) { should == subject.volume }

  context "compute locked for market order" do
    let(:price_levels) do
      [ ['202'.to_d, '10.0'.to_d],
        ['201'.to_d, '10.0'.to_d],
        ['200'.to_d, '10.0'.to_d],
        ['100'.to_d, '10.0'.to_d] ]
    end

    before do
      global = Global.new('btccny')
      global.stubs(:asks).returns(price_levels)
      Global.stubs(:[]).returns(global)
    end

    it "should require a little" do
      OrderBid.new(volume: '5'.to_d, ord_type: 'market').compute_locked.should == '1010'.to_d *
OrderBid::LOCKING_BUFFER_FACTOR
    end

    it "should raise error if volume is too large" do
      expect { OrderBid.new(volume: '30'.to_d, ord_type: 'market').compute_locked }.not_to
raise_error
      expect { OrderBid.new(volume: '31'.to_d, ord_type: 'market').compute_locked }.to raise_error
    end
  end

end

511:F:\git\coin\exchange\peatio-master\spec\models\order_bid_spec.rb
require 'spec_helper'

describe OrderBid do

  subject { create(:order_bid) }

  its(:compute_locked) { should == subject.volume*subject.price }

  context "compute locked for market order" do
    let(:price_levels) do
      [ ['100'.to_d, '10.0'.to_d],
        ['101'.to_d, '10.0'.to_d],
        ['102'.to_d, '10.0'.to_d],
        ['200'.to_d, '10.0'.to_d] ]
    end

    before do
      global = Global.new('btccny')
      global.stubs(:asks).returns(price_levels)
      Global.stubs(:[]).returns(global)
    end

    it "should require a little" do
      OrderBid.new(volume: '5'.to_d, ord_type: 'market').compute_locked.should == '500'.to_d *
OrderBid::LOCKING_BUFFER_FACTOR
    end

    it "should require more" do
      OrderBid.new(volume: '25'.to_d, ord_type: 'market').compute_locked.should == '2520'.to_d *
OrderBid::LOCKING_BUFFER_FACTOR
    end

    it "should raise error if the market is not deep enough" do
      expect { OrderBid.new(volume: '50'.to_d, ord_type: 'market').compute_locked }.to raise_error
    end
    it "should raise error if volume is too large" do
      expect { OrderBid.new(volume: '30'.to_d, ord_type: 'market').compute_locked }.not_to
raise_error
      expect { OrderBid.new(volume: '31'.to_d, ord_type: 'market').compute_locked }.to raise_error
    end
  end

end

512:F:\git\coin\exchange\peatio-master\spec\models\order_spec.rb
require 'spec_helper'

describe Order, 'validations' do
  it { should validate_presence_of(:ord_type) }
  it { should validate_presence_of(:volume) }
  it { should validate_presence_of(:origin_volume) }
  it { should validate_presence_of(:locked) }
  it { should validate_presence_of(:origin_locked) }

  context "limit order" do
    it "should make sure price is present" do
      order = Order.new(currency: 'btccny', price: nil, ord_type: 'limit')
      order.should_not be_valid
      order.errors[:price].should == ["is not a number"]
    end

    it "should make sure price is greater than zero" do
      order = Order.new(currency: 'btccny', price: '0.0'.to_d, ord_type: 'limit')
      order.should_not be_valid
      order.errors[:price].should == ["must be greater than 0"]
    end
  end

  context "market order" do
    it "should make sure price is not present" do
      order = Order.new(currency: 'btccny', price: '0.0'.to_d, ord_type: 'market')
      order.should_not be_valid
      order.errors[:price].should == ['must not be present']
    end
  end
end
```

```
describe Order, "#fix_number_precision" do
  let(:order_bid) { create(:order_bid, currency: 'btccny', price: '12.326'.to_d, volume:
'123.123456789') }
  let(:order_ask) { create(:order_ask, currency: 'btccny', price: '12.326'.to_d, volume:
'123.123456789') }
  it { expect(order_bid.price).to be_d '12.32' }
  it { expect(order_bid.volume).to be_d '123.1234' }
  it { expect(order_bid.origin_volume).to be_d '123.1234' }
  it { expect(order_ask.price).to be_d '12.32' }
  it { expect(order_ask.volume).to be_d '123.1234' }
  it { expect(order_ask.origin_volume).to be_d '123.1234' }
end

describe Order, "#done" do
  let(:ask_fee) { '0.003'.to_d }
  let(:bid_fee) { '0.001'.to_d }
  let(:order) { order_bid }
  let(:order_bid) { create(:order_bid, price: "1.2".to_d, volume: "10.0".to_d) }
  let(:order_ask) { create(:order_ask, price: "1.2".to_d, volume: "10.0".to_d) }
  let(:hold_account) { create(:account, member_id: 1, locked: "100.0".to_d, balance: "0.0".to_d) }
  let(:expect_account) { create(:account, member_id: 2, locked: "0.0".to_d, balance: "0.0".to_d) }

  before do
    order_bid.stubs(:hold_account).returns(hold_account)
    order_bid.stubs(:expect_account).returns(expect_account)
    order_ask.stubs(:hold_account).returns(hold_account)
    order_ask.stubs(:expect_account).returns(expect_account)
    OrderBid.any_instance.stubs(:fee).returns(bid_fee)
    OrderAsk.any_instance.stubs(:fee).returns(ask_fee)
  end

  def mock_trade(volume, price)
    build(:trade, volume: volume, price: price, id: rand(10))
  end

  shared_examples "trade done" do
    before do
      hold_account.reload
      expect_account.reload
    end

    it "order_bid done" do
      trade = mock_trade(strike_volume, strike_price)

      hold_account.expects(:unlock_and_sub_funds).with(
        strike_volume * strike_price, locked: strike_volume * strike_price,
        reason: Account::STRIKE_SUB, ref: trade)

      expect_account.expects(:plus_funds).with(
        strike_volume - strike_volume * bid_fee,
        has_entries(:reason => Account::STRIKE_ADD, :ref => trade))

      order_bid.strike(trade)
    end

    it "order_ask done" do
      trade = mock_trade(strike_volume, strike_price)
      hold_account.expects(:unlock_and_sub_funds).with(
        strike_volume, locked: strike_volume,
        reason: Account::STRIKE_SUB, ref: trade)

      expect_account.expects(:plus_funds).with(
        strike_volume * strike_price - strike_volume * strike_price * ask_fee,
        has_entries(:reason => Account::STRIKE_ADD, :ref => trade))

      order_ask.strike(trade)
    end
  end

  describe Order do
    describe "#state" do
      it "should be keep wait state" do
        expect do
          order.strike(mock_trade("5.0", "0.8"))
        end.to_not change{ order.state }.by(Order::WAIT)
      end

      it "should be change to done state" do
        expect do
          order.strike(mock_trade("10.0", "1.2"))
        end.to change{ order.state }.from(Order::WAIT).to(Order::DONE)
      end
    end
```

```
describe "#volume" do
  it "should be change volume" do
    expect do
      order.strike(mock_trade("4.0", "1.2"))
    end.to change{ order.volume }.from("10.0".to_d).to("6.0".to_d)
  end

  it "should be don't change origin volume" do
    expect do
      order.strike(mock_trade("4.0", "1.2"))
    end.to_not change{ order.origin_volume }.by("10.0".to_d)
  end
end

describe "#trades_count" do
  it "should increase trades count" do
    expect do
      order.strike(mock_trade("4.0", "1.2"))
    end.to change{ order.trades_count }.from(0).to(1)
  end
end

describe "#done" do
  context "trade done volume 5.0 with price 0.8" do
    let(:strike_price) { "0.8".to_d }
    let(:strike_volume) { "5.0".to_d }
    it_behaves_like "trade done"
  end

  context "trade done volume 3.1 with price 0.7" do
    let(:strike_price) { "0.7".to_d }
    let(:strike_volume) { "3.1".to_d }
    it_behaves_like "trade done"
  end

  context "trade done volume 10.0 with price 0.8" do
    let(:strike_price)  { "0.8".to_d }
    let(:strike_volume) { "10.0".to_d }

    it "should unlock not used funds" do
      trade = mock_trade(strike_volume, strike_price)
```

```
        hold_account.expects(:unlock_and_sub_funds).with(
          strike_volume * strike_price, locked: strike_volume * strike_price,
          reason: Account::STRIKE_SUB, ref: trade)

        expect_account.expects(:plus_funds).with(
          strike_volume - strike_volume * bid_fee,
          has_entries(:reason => Account::STRIKE_ADD, :ref => trade))

        hold_account.expects(:unlock_funds).with(
          strike_volume * (order.price - strike_price),
          reason: Account::ORDER_FULLFILLED, ref: trade)

        order_bid.strike(trade)
      end
    end
  end
 end
end

describe Order, "#head" do
  let(:currency) { :btccny }

  describe OrderAsk do
    it "price priority" do
      foo = create(:order_ask, price: "1.0".to_d, created_at: 2.second.ago)
      create(:order_ask, price: "1.1".to_d, created_at: 1.second.ago)
      expect(OrderAsk.head(currency)).to eql foo
    end

    it "time priority" do
      foo = create(:order_ask, price: "1.0".to_d, created_at: 2.second.ago)
      create(:order_ask, price: "1.0".to_d, created_at: 1.second.ago)
      expect(OrderAsk.head(currency)).to eql foo
    end
  end

  describe OrderBid do
    it "price priority" do
      foo = create(:order_bid, price: "1.1".to_d, created_at: 2.second.ago)
      create(:order_bid, price: "1.0".to_d, created_at: 1.second.ago)
      expect(OrderBid.head(currency)).to eql foo
    end
    it "time priority" do
      foo = create(:order_bid, price: "1.0".to_d, created_at: 2.second.ago)
      create(:order_bid, price: "1.0".to_d, created_at: 1.second.ago)
      expect(OrderBid.head(currency)).to eql foo
    end
  end
end

describe Order, "#kind" do
  it "should be ask for ask order" do
    OrderAsk.new.kind.should == 'ask'
  end

  it "should be bid for bid order" do
    OrderBid.new.kind.should == 'bid'
  end
end

describe Order, "related accounts" do
  let(:alice)  { who_is_billionaire }
  let(:bob)    { who_is_billionaire }

  context OrderAsk do
    it "should hold btc and expect cny" do
      ask = create(:order_ask, member: alice)
      ask.hold_account.should == alice.get_account(:btc)
      ask.expect_account.should == alice.get_account(:cny)
    end
  end

  context OrderBid do
    it "should hold cny and expect btc" do
      bid = create(:order_bid, member: bob)
      bid.hold_account.should == bob.get_account(:cny)
      bid.expect_account.should == bob.get_account(:btc)
    end
  end
end

describe Order, "#avg_price" do
  it "should be zero if not filled yet" do
```

```
    OrderAsk.new(locked: '1.0', origin_locked: '1.0', volume: '1.0', origin_volume: '1.0',
funds_received: '0').avg_price.should == '0'.to_d
    OrderBid.new(locked: '1.0', origin_locked: '1.0', volume: '1.0', origin_volume: '1.0',
funds_received: '0').avg_price.should == '0'.to_d
  end

  it "should calculate average price of bid order" do
    OrderBid.new(currency: 'btccny', locked: '10.0', origin_locked: '20.0', volume: '1.0',
origin_volume: '3.0', funds_received: '2.0').avg_price.should == '5'.to_d
  end

  it "should calculate average price of ask order" do
    OrderAsk.new(currency: 'btccny', locked: '1.0', origin_locked: '2.0', volume: '1.0', origin_volume:
'2.0', funds_received: '10.0').avg_price.should == '10'.to_d
  end
end

describe Order, "#estimate_required_funds" do
  let(:price_levels) do
    [ ['1.0'.to_d, '10.0'.to_d],
      ['2.0'.to_d, '20.0'.to_d],
      ['3.0'.to_d, '30.0'.to_d] ]
  end

  before do
    global = Global.new('btccny')
    global.stubs(:asks).returns(price_levels)
    Global.stubs(:[]).returns(global)
  end
end

describe Order, "#strike" do
  it "should raise error if order has been cancelled" do
    order = Order.new(state: Order::CANCEL)
    expect { order.strike(mock('trade')) }.to raise_error
  end
end

513:F:\git\coin\exchange\peatio-master\spec\models\partial_tree_spec.rb
require 'spec_helper'

describe PartialTree do
```

```
  pending "add some examples to (or delete) #{__FILE__}"
end

514:F:\git\coin\exchange\peatio-master\spec\models\payment_address_spec.rb
require 'spec_helper'

describe PaymentAddress do

  context ".create" do
    before do
      PaymentAddress.any_instance.stubs(:id).returns(1)
    end

    it "generate address after commit" do
      AMQPQueue.expects(:enqueue)
        .with(:deposit_coin_address,
              {payment_address_id: 1, currency: 'btc'},
              {persistent: true})

      PaymentAddress.create currency: :btc
    end
  end

end

515:F:\git\coin\exchange\peatio-master\spec\models\payment_transaction_spec.rb
require 'spec_helper'

describe PaymentTransaction do
  it "expect state transfer" do
    tx = create(:payment_transaction, deposit: create(:deposit))
    tx.stubs(:refresh_confirmations)

    tx.stubs(:min_confirm?).returns(false)
    tx.stubs(:max_confirm?).returns(false)

    expect(tx.unconfirm?).to be_true
    expect(tx.check).to be_false
    expect(tx.check).to be_false
    expect(tx.check).to be_false
    expect(tx.unconfirm?).to be_true
    tx.stubs(:min_confirm?).returns(true)
    tx.stubs(:max_confirm?).returns(false)

    expect(tx.check).to be_true
    expect(tx.confirming?).to be_true

    tx.stubs(:min_confirm?).returns(false)
    tx.stubs(:max_confirm?).returns(true)

    expect(tx.check).to be_true
    expect(tx.confirmed?).to be_true
    expect(tx.check).to be_true
  end

end
```

516:F:\git\coin\exchange\peatio-master\spec\models\proof_spec.rb

```
require 'spec_helper'

describe Proof do
  describe '#asset_sum' do
    it 'aggregates address balances' do
      proof = Proof.new(addresses: [
        {"address"=>"1HjfnJpQmANtuW7yr1ggeDfyfe1kDK7rxx", "balance"=>1},
        {"address"=>"1HjfnJpQmANtuW7yr1ggeDfyfe1kDK7rm3", "balance"=>2.00005},
        {"address"=>"1dice97ECuByXAvqXpaYzSaQuPVvrtmz6", "balance"=>5.84489237}
      ])

      expect(proof.asset_sum).to eq(8.84494237)
    end
  end
end
```

517:F:\git\coin\exchange\peatio-master\spec\models\ticket_spec.rb

```
require 'spec_helper'

describe Ticket do
  describe "Validation" do
    context "Both title and content is empty" do
      subject { Ticket.new }
      it { should_not be_valid }
    end
    context "Title is empty" do
      subject { Ticket.new(content: 'xman is here') }
      it { should be_valid }
    end

    context "Content is empty" do
      subject { Ticket.new(title: 'xman is here') }
      it { should be_valid }
    end

  end

  describe "#title_for_display" do
    let(:text) { 'alsadkjf aslkdjf aslkdjfla skdjf alsdkjf dlsakjf lasdkjf sadkfasdf xx' }
    context "title is present" do
      let(:ticket) { create(:ticket, title: text)}
      subject{ ticket }
      its(:title_for_display) { should == "alsadkjf aslkdjf aslkdjfla skdjf alsdkjf dlsakjf lasdkjf ..." }
    end

    context "title is blank" do
      let(:ticket) { create(:ticket, content: text) }
      subject{ ticket }
      its(:title_for_display) { should == "alsadkjf aslkdjf aslkdjfla skdjf alsdkjf dlsakjf lasdkjf ..." }
    end
  end

  describe "#send_notification" do
    let(:ticket) { create(:ticket) }
    let(:mailer) { mock() }
    before do
      mailer.stubs(:deliver)
      ticket
    end

    after do
      ticket.send(:send_notification)
    end

    it "should notify the admin" do
      TicketMailer.expects(:admin_notification).with(ticket.id).returns(mailer)
    end
  end
end

518:F:\git\coin\exchange\peatio-master\spec\models\token\activation_spec.rb
require 'spec_helper'

describe Token::Activation do
  let(:member) { create :member }
  let(:activation) { create :activation, member: member }

  describe '#confirm!' do
    before { activation.confirm! }

    it { expect(member).to be_activated }
  end

  describe 'send_token after creation' do
    let(:mail) { ActionMailer::Base.deliveries.last }

    before { activation }

    it { expect(mail.subject).to match('Account Activation') }
  end

end

519:F:\git\coin\exchange\peatio-master\spec\models\token\reset_password_spec.rb
require 'spec_helper'

describe Token::ResetPassword do
  let(:member) { create :member }
  let(:token) { Token::ResetPassword.new email: member.email }

  describe 'create' do
    it {
      expect {
        token.save
      }.to change(Token::ResetPassword, :count).by(1)
    }
    it { expect(token).not_to be_is_used }
  end
  describe 're-create token within 30 minutes' do
    before { token.save }

    it {
      expect {
        Timecop.travel(29.minutes.from_now)
        expect(token.reload).not_to be_expired

        new_token = Token::ResetPassword.create email: member.email
        expect(new_token).not_to be_valid
      }.not_to change(Token::ResetPassword, :count)
    }

  end

  describe 're-create token after 30 minutes' do
    before { token.save }

    it {
      expect {
        Timecop.travel(31.minutes.from_now)
        expect(token.reload).to be_expired

        new_token = Token::ResetPassword.create email: member.email
        expect(new_token).not_to be_expired
        expect(new_token).not_to eq(token)
      }.to change(Token::ResetPassword, :count).by(1)
    }
  end
end

520:F:\git\coin\exchange\peatio-master\spec\models\trade_spec.rb
require 'spec_helper'

describe Trade, ".latest_price" do
  context "no trade" do
    it { expect(Trade.latest_price(:btccny)).to be_d "0.0" }
  end

  context "add one trade" do
    let!(:trade) { create(:trade, currency: :btccny) }
    it { expect(Trade.latest_price(:btccny)).to eq(trade.price) }
  end
end

describe Trade, ".collect_side" do
  let(:member) { create(:member) }
  let(:ask)    { create(:order_ask, member: member) }
  let(:bid)    { create(:order_bid, member: member) }

  let!(:trades) {[
    create(:trade, ask: ask, created_at: 2.days.ago),
    create(:trade, bid: bid, created_at: 1.day.ago)
  ]}

  it "should add side attribute on trades" do
    results = Trade.for_member(ask.currency, member)
    results.should have(2).trades
    results.find {|t| t.id == trades.first.id }.side.should == 'ask'
    results.find {|t| t.id == trades.last.id  }.side.should == 'bid'
  end

  it "should sort trades in reverse creation order" do
    Trade.for_member(ask.currency, member, order: 'id desc').first.should == trades.last
  end

  it "should return 1 trade" do
    results = Trade.for_member(ask.currency, member, limit: 1)
    results.should have(1).trade
  end

  it "should return trades from specified time" do
    results = Trade.for_member(ask.currency, member, time_to: 30.hours.ago)
    results.should have(1).trade
    results.first.should == trades.first
  end
end

describe Trade, "#for_notify" do
  let(:order_ask) { create(:order_ask) }
  let(:order_bid) { create(:order_bid) }
  let(:trade) { create(:trade, ask: order_ask, bid: order_bid) }
  subject(:notify) { trade.for_notify('ask') }

  it { expect(notify).not_to be_blank }
  it { expect(notify[:kind]).not_to be_blank }
  it { expect(notify[:at]).not_to be_blank }
  it { expect(notify[:price]).not_to be_blank }
  it { expect(notify[:volume]).not_to be_blank }

  it "should use side as kind" do
    trade.side = 'ask'
    trade.for_notify[:kind].should == 'ask'
  end

end
```

521:F:\git\coin\exchange\peatio-master\spec\models\two_factor\app_spec.rb

```
require 'spec_helper'

describe TwoFactor::App do
  let(:member) { create :member }
  let(:app) { member.app_two_factor  }

  describe "generate code" do
    subject { app }

    its(:otp_secret) { should_not be_blank }
  end

  describe '#refresh' do
    context 'inactivated' do
      it {
        orig_otp_secret = app.otp_secret.dup
        app.refresh!
        expect(app.otp_secret).not_to eq(orig_otp_secret)
      }
    end

    context 'activated' do
      subject { create :two_factor_app, activated: true }

      it {
        orig_otp_secret = subject.otp_secret.dup
        subject.refresh!
        expect(subject.otp_secret).to eq(orig_otp_secret)
      }
    end
  end

  describe 'uniq validate' do
    let(:member) { create :member }

    it "reject duplicate creation" do
      duplicate = TwoFactor.new app.attributes
      expect(duplicate).not_to be_valid
    end
  end

  describe 'self.fetch_by_type' do
    it "return nil for wrong type" do
      expect(TwoFactor.by_type(:foobar)).to be_nil
    end

    it "create new one by type" do
      expect {
        expect(app).not_to be_nil
      }.to change(TwoFactor::App, :count).by(1)
    end

    it "retrieve exist one instead of creating" do
      two_factor = member.app_two_factor
      expect(member.app_two_factor).to eq(two_factor)
    end
  end

  describe '#active!' do
    subject { member.app_two_factor }
    before { subject.active! }

    its(:activated?) { should be_true }
  end

  describe '#deactive!' do
    subject { create :two_factor_app, activated: true }
    before { subject.deactive! }
```

```
    its(:activated?) { should_not be_true }
  end


  describe '.activated' do
    before { create :member, :app_two_factor_activated }

    it "should has activated" do
      expect(TwoFactor.activated?).to be_true
    end
  end

  describe 'send_notification_mail' do
    let(:mail) { ActionMailer::Base.deliveries.last }

    describe "activated" do
      before { app.active! }

      it { expect(mail.subject).to match('Google authenticator activated') }
    end

    describe "deactived" do
      let(:member) { create :member, :app_two_factor_activated }
      before { app.deactive! }

      it { expect(mail.subject).to match('Google authenticator deactivated') }
    end
  end

end

522:F:\git\coin\exchange\peatio-master\spec\models\two_factor\sms_spec.rb
require 'spec_helper'

describe TwoFactor::Sms do
  let(:member) { create :member }
  let(:two_factor) { member.sms_two_factor }

  describe "generate code" do
    subject { two_factor }
    it "should generate 6 random digits" do
      subject.otp_secret.should =~ /^\d{6}$/
    end
  end

  describe "#refresh" do
    subject { two_factor }

    its(:otp_secret) { should_not be_blank }
    it {
      orig_otp_secret = two_factor.otp_secret.dup
      two_factor.refresh!
      expect(two_factor.otp_secret).not_to eq(orig_otp_secret)
    }
  end

  describe "#phone_number and #country" do
    describe "assigns phone_number and country" do
      subject {
        two_factor.phone_number = '123-1234-1234'
        two_factor.country = 'CN'
        two_factor
      }

      its(:phone_number) { should_not be_blank }
      its(:country) { should_not be_blank }
    end

    describe "invalid phone_number on send code phase" do
      subject {
        two_factor.send_code_phase = true
        two_factor.phone_number = '0412789194'
        two_factor
      }

      it { should_not be_valid }
    end

    describe "valid phone_number with country on send code phase" do
      subject {
        two_factor.send_code_phase = true
        two_factor.phone_number = '0412789194'
        two_factor.country = 'AU'
        two_factor
      }

      it { should be_valid }
    end
  end

  describe "#update member's phone_number when send_otp" do
    subject {
      two_factor.phone_number = '123-1234-1234'
      two_factor.send_otp
      two_factor
    }

    it { expect(member.phone_number).not_to be_blank }
  end

  describe '#verify?' do
    describe 'invalid code' do
      subject {
        two_factor.otp = 'foobar'
        two_factor
      }

      it { should_not be_verify }
    end

    describe 'verify succeed' do
      subject {
        two_factor.otp = two_factor.otp_secret
        two_factor
      }

      it { should be_verify }
    end
  end

  describe '#sms_message' do
    its(:sms_message) { should_not be_blank }
  end
```

```
  describe '#activated' do
    let(:member) { create :member }
    subject {
      two_factor = member.sms_two_factor
      two_factor.deactive!
      two_factor
    }

    it { expect(subject).not_to be_activated }
    it { expect(member.sms_two_factor).not_to be_activated }
  end
end


523:F:\git\coin\exchange\peatio-master\spec\models\withdraw_spec.rb
require 'spec_helper'

describe Withdraw do

  context '#fix_precision' do
    it "should round down to max precision" do
      withdraw = create(:satoshi_withdraw, sum: '0.123456789')
      withdraw.sum.should == '0.12345678'.to_d
    end
  end

  context 'fund source' do
    it "should strip trailing spaces in fund_uid" do
      fund_source = create(:btc_fund_source, uid: 'test   ')
      @withdraw = create(:satoshi_withdraw, fund_source_id: fund_source.id)
      @withdraw.fund_uid.should == 'test'
    end
  end

  context 'bank withdraw' do
    describe "#audit!" do
      subject { create(:bank_withdraw) }
      before  { subject.submit! }

      it "should accept withdraw with clean history" do
        subject.audit!
        subject.should be_accepted
    end

    it "should mark withdraw with suspicious history" do
      subject.account.versions.delete_all
      subject.audit!
      subject.should be_suspect
    end

    it "should approve quick withdraw directly" do
      subject.update_attributes sum: 5
      subject.audit!
      subject.should be_processing
    end
  end
end

context 'coin withdraw' do
  describe '#audit!' do
    subject { create(:satoshi_withdraw) }

    before do
      subject.submit!
    end

    it "should be rejected if address is invalid" do
      CoinRPC.stubs(:[]).returns(mock('rpc', validateaddress: {isvalid: false}))
      subject.audit!
      subject.should be_rejected
    end

    it "should be rejected if address belongs to hot wallet" do
      CoinRPC.stubs(:[]).returns(mock('rpc', validateaddress: {isvalid: true, ismine: true}))
      subject.audit!
      subject.should be_rejected
    end

    it "should accept withdraw with clean history" do
      CoinRPC.stubs(:[]).returns(mock('rpc', validateaddress: {isvalid: true}))
      subject.audit!
      subject.should be_accepted
    end
    it "should mark withdraw with suspicious history" do
      CoinRPC.stubs(:[]).returns(mock('rpc', validateaddress: {isvalid: true}))
      subject.account.versions.delete_all
      subject.audit!
      subject.should be_suspect
    end

    it "should approve quick withdraw directly" do
      CoinRPC.stubs(:[]).returns(mock('rpc', validateaddress: {isvalid: true}))
      subject.update_attributes sum: '0.099'
      subject.audit!
      subject.should be_processing
    end
  end

  describe 'sn' do
    before do
      Timecop.freeze(Time.local(2013,10,7,18,18,18))
      @withdraw = create(:satoshi_withdraw, id: 1)
    end

    after do
      Timecop.return
    end

    it "generate right sn" do
      expect(@withdraw.sn).to eq('13100718180001')
    end

    it 'alias withdraw_id to sn' do
      expect(@withdraw.withdraw_id).to eq('13100718180001')
    end
  end

  describe 'account id assignment' do
    subject { build :satoshi_withdraw, account_id: 999 }

    it "don't accept account id from outside" do
      subject.save
      expect(subject.account_id).to eq(subject.member.get_account(subject.currency).id)
    end
  end
```

```
  end

  context 'Worker::WithdrawCoin#process' do
    subject { create(:satoshi_withdraw) }
    before do
      @rpc = mock()
      @rpc.stubs(getbalance: 50000, sendtoaddress: '12345', settxfee: true )
      @broken_rpc = mock()
      @broken_rpc.stubs(getbalance: 5)

      subject.submit
      subject.accept
      subject.process
      subject.save!
    end

    it 'transitions to :almost_done after calling rpc but getting Exception' do
      CoinRPC.stubs(:[]).returns(@broken_rpc)

      lambda { Worker::WithdrawCoin.new.process({id: subject.id}, {}, {}) }.should
raise_error(Account::BalanceError)

      expect(subject.reload.almost_done?).to be_true
    end

    it 'transitions to :done after calling rpc' do
      CoinRPC.stubs(:[]).returns(@rpc)

      expect { Worker::WithdrawCoin.new.process({id: subject.id}, {}, {}) }.to
change{subject.account.reload.amount}.by(-subject.sum)

      subject.reload
      expect(subject.done?).to be_true
      expect(subject.txid).to eq('12345')
    end

    it 'does not send coins again if previous attempt failed' do
      CoinRPC.stubs(:[]).returns(@broken_rpc)
      begin Worker::WithdrawCoin.new.process({id: subject.id}, {}, {}); rescue; end
      CoinRPC.stubs(:[]).returns(mock())

      expect { Worker::WithdrawCoin.new.process({id: subject.id}, {}, {}) }.to_not
```

```
change{subject.account.reload.amount}
      expect(subject.reload.almost_done?).to be_true
    end
  end

  context 'aasm_state' do
    subject { create(:bank_withdraw, sum: 1000) }

    before do
      subject.stubs(:send_withdraw_confirm_email)
    end

    it 'initializes with state :submitting' do
      expect(subject.submitting?).to be_true
    end

    it 'transitions to :submitted after calling #submit!' do
      subject.submit!

      expect(subject.submitted?).to be_true
      expect(subject.sum).to eq subject.account.locked
      expect(subject.sum).to eq subject.account_versions.last.locked
    end

    it 'transitions to :rejected after calling #reject!' do
      subject.submit!
      subject.accept!
      subject.reject!

      expect(subject.rejected?).to be_true
    end

    context :process do
      before do
        subject.submit!
        subject.accept!
      end

      it 'transitions to :processing after calling #process! when withdrawing fiat currency' do
        subject.stubs(:coin?).returns(false)

        subject.process!
        expect(subject.processing?).to be_true
      end

      it 'transitions to :failed after calling #fail! when withdrawing fiat currency' do
        subject.stubs(:coin?).returns(false)

        subject.process!

        expect { subject.fail! }.to_not change{subject.account.amount}

        expect(subject.failed?).to be_true
      end

      it 'transitions to :processing after calling #process!' do
        subject.expects(:send_coins!)

        subject.process!

        expect(subject.processing?).to be_true
      end
    end

    context :cancel do
      it 'transitions to :canceled after calling #cancel!' do
        subject.cancel!

        expect(subject.canceled?).to be_true
        expect(subject.account.locked).to eq 0
      end

      it 'transitions from :submitted to :canceled after calling #cancel!' do
        subject.submit!
        subject.cancel!

        expect(subject.canceled?).to be_true
        expect(subject.account.locked).to eq 0
      end

      it 'transitions from :accepted to :canceled after calling #cancel!' do
        subject.submit!
        subject.accept!
```

```
        subject.cancel!

        expect(subject.canceled?).to be_true
        expect(subject.account.locked).to eq 0
      end
    end
  end

  context "#quick?" do
    subject(:withdraw) { build(:satoshi_withdraw) }

    it "returns false if currency doesn't set quick withdraw max" do
      withdraw.should_not be_quick
    end

    it "returns false if exceeds quick withdraw amount" do
      withdraw.currency_obj.stubs(:quick_withdraw_max).returns(withdraw.sum-1)
      withdraw.should_not be_quick
    end

    it "returns true" do
      withdraw.currency_obj.stubs(:quick_withdraw_max).returns(withdraw.sum+1)
      withdraw.should be_quick
    end
  end

end
```

```
524:F:\git\coin\exchange\peatio-master\spec\models\worker\deposit_coin_spec.rb
require 'spec_helper'

describe Worker::DepositCoin do

  subject { Worker::DepositCoin.new }

  context "sendmany transaction" do
    let(:raw) do
      {:amount=>0.2,
       :confirmations=>39,
       :blockhash=>
        "0000000000d744827317b3f679c52d0090243a13153c6082e0e65cb83fa1193d",
       :blockindex=>1,
       :blocktime=>1412317163,
       :txid=>"1a33b61174e5c52c189af4169b6919d059a0024ee6526326961fe6dd8af2e260",
       :walletconflicts=>[],
       :time=>1412317158,
       :timereceived=>1412317158,
       :details=>
        [{"account"=>"payment",
          "address"=>"mov9LqpntN18cuyzUDBoaS8vPY8pF421Y3",
          "category"=>"receive",
          "amount"=>0.1},
         {"account"=>"payment",
          "address"=>"mqRtfJSdgrbbgMPasq4j3br1G4h3AoJ4hE",
          "category"=>"receive",
          "amount"=>0.1}],
       :hex=> ''}
    end

    let(:payload) do
      {'txid' => '1a33b61174e5c52c189af4169b6919d059a0024ee6526326961fe6dd8af2e260',
       'channel_key' => 'satoshi'}
    end

    before do
      create(:btc_payment_address, address: 'mov9LqpntN18cuyzUDBoaS8vPY8pF421Y3')
      create(:btc_payment_address, address: 'mqRtfJSdgrbbgMPasq4j3br1G4h3AoJ4hE')
      subject.stubs(:get_raw).returns(raw)
    end

    it "should deposit many accounts" do
      lambda {
        subject.process payload, {}, {}
      }.should change(Deposit, :count).by(2)
    end
  end

end

525:F:\git\coin\exchange\peatio-master\spec\models\worker\matching_spec.rb
require 'spec_helper'

describe Worker::Matching do
```

```
  let(:alice)  { who_is_billionaire }
  let(:bob)    { who_is_billionaire }
  let(:market) { Market.find('btccny') }

  subject { Worker::Matching.new }

  context "engines" do
    it "should get all engines" do
      subject.engines.keys.should == [market.id]
    end

    it "should started all engines" do
      subject.engines.values.map(&:mode).should == [:run]
    end
  end

  context "partial match" do
    let(:existing) { create(:order_ask, price: '4001', volume: '10.0', member: alice) }

    before do
      subject.process({action: 'submit', order: existing.to_matching_attributes}, {}, {})
    end

    it "should started engine" do
      subject.engines['btccny'].mode.should == :run
    end

    it "should match part of existing order" do
      order = create(:order_bid, price: '4001', volume: '8.0', member: bob)

      AMQPQueue.expects(:enqueue)
        .with(:slave_book, {action: 'update', order: {id: existing.id, timestamp: existing.at, type: :ask,
volume: '2.0'.to_d, price: existing.price, market: 'btccny', ord_type: 'limit'}}, anything)
      AMQPQueue.expects(:enqueue)
        .with(:trade_executor, {market_id: market.id, ask_id: existing.id, bid_id: order.id, strike_price:
'4001'.to_d, volume: '8.0'.to_d, funds: '32008'.to_d}, anything)
      subject.process({action: 'submit', order: order.to_matching_attributes}, {}, {})
    end

    it "should match part of new order" do
      order = create(:order_bid, price: '4001', volume: '12.0', member: bob)
      AMQPQueue.expects(:enqueue)
        .with(:trade_executor, {market_id: market.id, ask_id: existing.id, bid_id: order.id, strike_price:
'4001'.to_d, volume: '10.0'.to_d, funds: '40010'.to_d}, anything)
      AMQPQueue.expects(:enqueue).with(:slave_book, anything, anything).times(2)
      subject.process({action: 'submit', order: order.to_matching_attributes}, {}, {})
    end
  end

  context "complex partial match" do
    # submit  | ask price/volume | bid price/volume |
    # -----------------------------------------------
    # ask1    | 4003/3           |                  |
    # -----------------------------------------------
    # ask2    | 4002/3, 4003/3   |                  |
    # -----------------------------------------------
    # bid3    |                  | 4003/2           |
    # -----------------------------------------------
    # ask4    | 4002/3           |                  |
    # -----------------------------------------------
    # bid5    |                  |                  |
    # -----------------------------------------------
    # bid6    |                  | 4001/5           |
    # -----------------------------------------------
    let!(:ask1) { create(:order_ask, price: '4003', volume: '3.0', member: alice) }
    let!(:ask2) { create(:order_ask, price: '4002', volume: '3.0', member: alice) }
    let!(:bid3) { create(:order_bid, price: '4003', volume: '8.0', member: bob) }
    let!(:ask4) { create(:order_ask, price: '4002', volume: '5.0', member: alice) }
    let!(:bid5) { create(:order_bid, price: '4003', volume: '3.0', member: bob) }
    let!(:bid6) { create(:order_bid, price: '4001', volume: '5.0', member: bob) }

    let!(:orderbook) { Matching::OrderBookManager.new('btccny', broadcast: false) }
    let!(:engine)    { Matching::Engine.new(market, mode: :run) }

    before do
      engine.stubs(:orderbook).returns(orderbook)
      ::Matching::Engine.stubs(:new).returns(engine)
    end

    it "should create many trades" do
      AMQPQueue.expects(:enqueue)
        .with(:trade_executor, {market_id: market.id, ask_id: ask1.id, bid_id: bid3.id, strike_price:
```

```
ask1.price, volume: ask1.volume, funds: '12009'.to_d}, anything).once
      AMQPQueue.expects(:enqueue)
        .with(:trade_executor, {market_id: market.id, ask_id: ask2.id, bid_id: bid3.id, strike_price:
ask2.price, volume: ask2.volume, funds: '12006'.to_d}, anything).once
      AMQPQueue.expects(:enqueue)
        .with(:trade_executor, {market_id: market.id, ask_id: ask4.id, bid_id: bid3.id, strike_price:
bid3.price, volume: '2.0'.to_d, funds: '8006'.to_d}, anything).once
      AMQPQueue.expects(:enqueue)
        .with(:trade_executor, {market_id: market.id, ask_id: ask4.id, bid_id: bid5.id, strike_price:
ask4.price, volume: bid5.volume, funds: '12006'.to_d}, anything).once

      subject
    end
  end

  context "cancel order" do
    let(:existing) { create(:order_ask, price: '4001', volume: '10.0', member: alice) }

    before do
      subject.process({action: 'submit', order: existing.to_matching_attributes}, {}, {})
    end

    it "should cancel existing order" do
      subject.process({action: 'cancel', order: existing.to_matching_attributes}, {}, {})
      subject.engines[market.id].ask_orders.limit_orders.should be_empty
    end
  end

  context "dryrun" do
    let!(:ask) { create(:order_ask, price: '4000', volume: '3.0', member: alice) }
    let!(:bid) { create(:order_bid, price: '4001', volume: '8.0', member: bob) }

    subject { Worker::Matching.new(mode: :dryrun) }

    context "very old orders matched" do
      before do
        ask.update_column :created_at, 1.day.ago
      end

      it "should not start engine" do
        subject.engines['btccny'].mode.should == :dryrun
        subject.engines['btccny'].queue.should have(1).trade
      end
    end

    context "buffered orders matched" do
      it "should start engine" do
        subject.engines['btccny'].mode.should == :run
      end
    end
  end

end

526:F:\git\coin\exchange\peatio-master\spec\models\worker\slave_book_spec.rb
require 'spec_helper'

describe Worker::SlaveBook do

  subject { Worker::SlaveBook.new(false) }

  let(:market)   { Market.find(:btccny) }
  let(:low_ask)  { Matching.mock_limit_order(type: 'ask', price: '10.0'.to_d) }
  let(:high_ask) { Matching.mock_limit_order(type: 'ask', price: '12.0'.to_d) }
  let(:low_bid)  { Matching.mock_limit_order(type: 'bid', price: '6.0'.to_d) }
  let(:high_bid) { Matching.mock_limit_order(type: 'bid', price: '8.0'.to_d) }

  context "#get_depth" do
    before do
      subject.process({action: 'add', order: low_ask.attributes}, {}, {})
      subject.process({action: 'add', order: high_ask.attributes}, {}, {})
      subject.process({action: 'add', order: low_bid.attributes}, {}, {})
      subject.process({action: 'add', order: high_bid.attributes}, {}, {})
    end

    it "should return lowest asks" do
      subject.get_depth(market, :ask).should == [
        ['10.0'.to_d, low_ask.volume],
        ['12.0'.to_d, high_ask.volume]
      ]
    end

    it "should return highest bids" do
      subject.get_depth(market, :bid).should == [
        ['8.0'.to_d, high_bid.volume],
        ['6.0'.to_d, low_bid.volume]
      ]
    end

    it "should updated volume" do
      attrs = low_ask.attributes.merge(volume: '0.01'.to_d)
      subject.process({action: 'update', order: attrs}, {}, {})
      subject.get_depth(market, :ask).should == [
        ['10.0'.to_d, '0.01'.to_d],
        ['12.0'.to_d, high_ask.volume]
      ]
    end
  end

  context "#process" do
    it "should create new orderbook manager" do
      subject.process({action: 'add', order: low_ask.attributes}, {}, {})
      subject.process({action: 'new', market: market.id, side: 'ask'}, {}, {})
      subject.get_depth(market, :ask).should be_empty
    end

    it "should remove an empty order" do
      subject.process({action: 'add', order: low_ask.attributes}, {}, {})
      subject.get_depth(market, :ask).should_not be_empty

      # after matching, order volume could be ZERO
      attrs = low_ask.attributes.merge(volume: '0.0'.to_d)
      subject.process({action: 'remove', order: attrs}, {}, {})

      subject.get_depth(market, :ask).should be_empty
    end
  end

end

527:F:\git\coin\exchange\peatio-master\spec\observers\transfer_observer_spec.rb
require 'spec_helper'

describe TransferObserver do
  describe "#after_update" do
    let!(:member) { create(:member) }
    let!(:deposit) { create(:deposit, aasm_state: 'submitted')}
    before do
      TransferObserver.any_instance.stubs(:current_user).returns(member)
    end

    subject { deposit.update_attributes(aasm_state: 'accepted')}

    it "should create the audit log" do
      expect { subject }.to change{ Audit::TransferAuditLog.count }.by(1)

    end
  end
end

528:F:\git\coin\exchange\peatio-master\spec\routing\admin\members_spec.rb
require 'spec_helper'

describe '/admin/members' do
end

529:F:\git\coin\exchange\peatio-master\spec\routing\admin\two_factors_spec.rb
require 'spec_helper'

describe '/admin/members/1/two_factors' do
  let(:url) { '/admin/members/1/two_factors/1' }
  it { expect(delete: url).to be_routable }
end

530:F:\git\coin\exchange\peatio-master\spec\routing\trade_spec.rb
require "spec_helper"

describe "routes for trade" do

  it "routes /markets/xxxyyy to the trade controller" do
    Market.expects(:find_by_id).with('xxxyyy').returns(Market.new(id: 'xxxyyy', base_unit: 'xxx',
quote_unit: 'yyy'))
    { :get => "/markets/xxxyyy" }.should be_routable

    Market.expects(:find_by_id).with('yyyxxx').returns(nil)
    { :get => "/markets/yyyxxx" }.should_not be_routable
  end
```

```
end

531:F:\git\coin\exchange\peatio-master\spec\routing\two_factors_spec.rb
require 'spec_helper'

describe 'two_factors' do
  it { expect(get('/two_factors/sms')).to be_routable }
  it { expect(get('/two_factors')).to be_routable }
  it { expect(put('/two_factors/sms')).to be_routable }
end

532:F:\git\coin\exchange\peatio-master\spec\routing\verify\google_auths_spec.rb
require 'spec_helper'

describe 'google_auths' do
  describe 'get /verify/google_auth' do
    it { expect(get('/verify/google_auth')).to be_routable }
  end

  describe 'get /verify/google_auth/edit' do
    it { expect(get('/verify/google_auth/edit')).to be_routable }
  end

  describe 'put /verify/google_auth' do
    it { expect(put('/verify/google_auth')).to be_routable }
  end
end

533:F:\git\coin\exchange\peatio-master\spec\routing\verify\sms_auths_spec.rb
require 'spec_helper'

describe "sms_auths" do
  describe "GET /verify/sms_auth" do
    it { expect(get("/verify/sms_auth")).to be_routable }
  end

  describe "PUT /verify/sms_auth" do
    it { expect(put("/verify/sms_auth")).to be_routable }
  end
end

534:F:\git\coin\exchange\peatio-master\spec\services\coin_rpc_spec.rb
```

```
require 'spec_helper'

describe CoinRPC do
  describe '#http_post_request' do
    it 'raises custom error on connection refused' do
      Net::HTTP.any_instance.stubs(:request).raises(Errno::ECONNREFUSED)

      rpc_client = CoinRPC::BTC.new('http://127.0.0.1:18332')

      expect {
        rpc_client.http_post_request ''
      }.to raise_error(CoinRPC::ConnectionRefusedError)
    end
  end
end

535:F:\git\coin\exchange\peatio-master\spec\services\ordering_spec.rb
require 'spec_helper'

describe Ordering do
  let(:order) { create(:order_bid, volume: '1.23456789', price: '1.23456789') }
  let(:account) { create(:account, balance: 100.to_d, locked: 100.to_d) }

  describe "ordering service can submit order" do
    before do
      order.stubs(:hold_account).returns(account)
      AMQPQueue.expects(:enqueue).with(:matching, anything)
    end

    it "should return true on success" do
      Ordering.new(order).submit.should be_true
    end

    it "should set locked funds on order" do
      Ordering.new(order).submit
      order.locked.should == order.compute_locked
      order.origin_locked.should == order.compute_locked
    end

    it "should compute locked after number precision fixed" do
      Ordering.new(order).submit
      order.reload.locked.should == '1.23'.to_d * '1.2345'.to_d
    end
  end

  describe "ordering service can cancel order" do
    before do
      order.stubs(:hold_account).returns(account)
    end

    it "should soft cancel order" do
      AMQPQueue.expects(:enqueue).with(:matching, action: 'cancel', order:
order.to_matching_attributes)
      Ordering.new(order).cancel
    end

    it "should hard cancel order" do
      Ordering.new(order).cancel!
      order.reload.state.should == Order::CANCEL
      account.reload.locked.should == ('100'.to_d - order.locked)
    end
  end
end
```

536:F:\git\coin\exchange\peatio-master\spec\spec_helper.rb

```
# This file is copied to spec/ when you run 'rails generate rspec:install'
ENV["RAILS_ENV"] ||= 'test'
ENV["ADMIN"] ||= 'admin@peatio.dev'
require File.expand_path("../../config/environment", __FILE__)
require 'rspec/rails'
require 'rspec/autorun'
require 'capybara/poltergeist'

# Requires supporting ruby files with custom matchers and macros, etc,
# in spec/support/ and its subdirectories.
Dir[Rails.root.join("spec/support/**/*.rb")].each { |f| require f }

# Checks for pending migrations before tests are run.
# If you are not using ActiveRecord, you can remove this line.
ActiveRecord::Migration.check_pending! if defined?(ActiveRecord::Migration)

Capybara.register_driver :poltergeist do |app|
  Capybara::Poltergeist::Driver.new(app, {
    js_errors: false,
    debug: false,
    logger: nil,
    phantomjs_logger: nil,
    window_size: [1440, 900]
  })
end

Capybara.javascript_driver = :poltergeist

RSpec.configure do |config|
  # ## Mock Framework
  #
  # If you prefer to use mocha, flexmock or RR, uncomment the appropriate line:
  #
  config.mock_with :mocha
  # config.mock_with :flexmock
  # config.mock_with :rr

  # Remove this line if you're not using ActiveRecord or ActiveRecord fixtures
  # config.fixture_path = "#{::Rails.root}/spec/fixtures"

  # If you're not using ActiveRecord, or you'd prefer not to run each of your
  # examples within a transaction, remove the following line or assign false
  # instead of true.
  config.use_transactional_fixtures = false

  # If true, the base class of anonymous controllers will be inferred
  # automatically. This will be the default behavior in future versions of
  # rspec-rails.
  config.infer_base_class_for_anonymous_controllers = false

  # Run specs in random order to surface order dependencies. If you find an
  # order dependency and want to debug it, you can fix the order by providing
  # the seed, which is printed after each run.
  #     --seed 1234
  config.order = "random"

  config.include FactoryGirl::Syntax::Methods

  config.before(:suite) do
    DatabaseCleaner.strategy = :deletion
  end
  config.before(:each) do
    DatabaseCleaner.start

    Rails.cache.clear
    AMQPQueue.stubs(:publish)
    KlineDB.stubs(:kline).returns([])

    I18n.locale = :en
  end

  config.after(:each) do
    DatabaseCleaner.clean
  end
end
```

537:F:\git\coin\exchange\peatio-master\spec\support\api_helper.rb

```
def time_to_milliseconds(t=Time.now)
  (t.to_f*1000).to_i
end

def sign(secret_key, method, uri, params)
  req = mock('request', request_method: method.to_s.upcase, path_info: uri)
  auth = APIv2::Auth::Authenticator.new(req, params)
  APIv2::Auth::Utils.hmac_signature(secret_key, auth.payload)
end

def signed_request(method, uri, opts={})
  token = opts[:token] || create(:api_token)
  path  = uri.sub(/^\/api/, '')

  params = opts[:params] || {}
  params[:access_key] = token.access_key
  params[:tonce]      = time_to_milliseconds
  params[:signature]  = sign(token.secret_key, method, path, params)

  send method, uri, params
end

def signed_get(uri, opts={})
  signed_request :get, uri, opts
```

```
end

def signed_post(uri, opts={})
  signed_request :post, uri, opts
end

def signed_delete(uri, opts={})
  signed_request :delete, uri, opts
end
```

538:F:\git\coin\exchange\peatio-master\spec\support\cookie_helper.rb

```
def clear_cookie
  page.driver.cookies.each do |k, v|
    page.driver.remove_cookie k
  end
end
```

539:F:\git\coin\exchange\peatio-master\spec\support\deposit_helper.rb

```
def deposit admin_identity, member, amount
    login admin_identity
    click_on 'admin'

    # this part is handled by a google extension
    query = {deposit: { txid: "deposit_#{Time.now.to_i}",
              sn: member.sn,
              fund_uid: identity.email,
              fund_extra: member.name,
              amount: amount }}

    visit(new_admin_currency_deposit_path(query))

    within 'form' do
      click_on I18n.t('helpers.submit.deposit.create')
    end
end
```

540:F:\git\coin\exchange\peatio-master\spec\support\i18n_helper.rb

```
def t(key)
  I18n.t(key)
end
```

541:F:\git\coin\exchange\peatio-master\spec\support\login_helper.rb

```ruby
def login(identity, otp: nil, password: nil)
  visit root_path
  click_on I18n.t('header.signin')
  expect(current_path).to eq(signin_path)

  within 'form#new_identity' do
    fill_in 'identity_email', with: identity.email
    fill_in 'identity_password', with: (password || identity.password)
    click_on I18n.t('header.signin')
  end

  if otp
    fill_in 'two_factor_otp', with: otp
    click_on I18n.t('helpers.submit.two_factor.create')
  end
end

def signout
  click_link t('header.signout')
end

def check_signin
  expect(page).not_to have_content(I18n.t('header.signin'))
end

alias :signin :login
```

542:F:\git\coin\exchange\peatio-master\spec\support\matching_helper.rb

```ruby
def who_is_billionaire
  member = create(:member)
  member.get_account(:btc).update_attributes(
    locked: '1000000000.0'.to_d, balance: '1000000000.0'.to_d)
  member.get_account(:cny).update_attributes(
    locked: '1000000000.0'.to_d, balance: '1000000000.0'.to_d)
  member
end

def print_time(time_hash)
  msg = time_hash.map{|k,v| "#{k}: #{v}"}.join(", ")
  puts "    \u25BC #{msg}"
end
```

```ruby
module Matching

  class <<self
    @@mock_order_id = 10000

    def mock_limit_order(attrs)
      @@mock_order_id += 1
      Matching::LimitOrder.new({
        id: @@mock_order_id,
        timestamp: Time.now.to_i,
        volume: 1+rand(10),
        price:  3000+rand(3000),
        market: 'btccny'
      }.merge(attrs))
    end

    def mock_market_order(attrs)
      @@mock_order_id += 1
      Matching::MarketOrder.new({
        id: @@mock_order_id,
        timestamp: Time.now.to_i,
        volume: 1+rand(10),
        locked: 15000+rand(15000),
        market: 'btccny'
      }.merge(attrs))
    end
  end

end
```

543:F:\git\coin\exchange\peatio-master\spec\support\pusher.rb

```ruby
# stub pusher requests

class Pusher::Client
  def trigger_async(*args)
  end
end

class Pusher::Channel
  def trigger_async(*args)
  end
```

```
end

544:F:\git\coin\exchange\peatio-master\spec\support\rspec_matchers.rb
RSpec::Matchers.define :be_d do |expected|
  match do |actual|
    if expected.kind_of? BigDecimal
      actual.to_d == expected
    elsif expected.kind_of? String
      actual.to_d == expected.to_d
    else
      raise "not support type #{expected.class}"
    end
  end

  failure_message_for_should do |actual|
    "expected #{actual.to_s} would be of #{expected.to_s}"
  end
end
```