Class Access:
- `apermission` (Boolean, true -> read only; false -> read and write)

db/schema.rb

```ruby
ActiveRecord::Schema.define(version: 20131012175724) do
  create_table "accesses", force: true do |t|
    t.boolean  "apermission"
    t.integer   "note_id"
    t.integer   "user_id"
    t.datetime  "created_at"
    t.datetime  "updated_at"
  end

  add_index "accesses", ["note_id"], name: "index_accesses_on_note_id"
  add_index "accesses", ["user_id"], name: "index_accesses_on_user_id"

  create_table "notes", force: true do |t|
    t.text     "text"
    t.integer  "user_id"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  add_index "notes", ["user_id"], name: "index_notes_on_user_id"

  create_table "users", force: true do |t|
    t.string    "email"
    t.string    "password_hash"
    t.string    "password_salt"
    t.datetime  "created_at"
    t.datetime  "updated_at"
    t.string    "username"
  end
end
```

app/models/access.rb

```ruby
class Access < ActiveRecord::Base
  belongs_to :note
  belongs_to :user
  attr_accessible :apermission, :user, :note
  validates_presence_of :note_id, on => :create
  validates_presence_of :user_id, on => :create
end
```

app/models/user.rb

```ruby
class User < ActiveRecord::Base
  attr_accessible :email, :username, :password, :password_confirmation
  has_many :notes, dependent: :destroy
  has_many :accesses, dependent => :destroy

  attr_accessor :password
  before_save :encrypt_password

  validates_confirmation_of :password
  validates :email, :format => { :with => /\A([^\s]+)\z/... }

  def self.search(search)
    if search
      find(:all, :conditions => ['username LIKE ?', "#{search}%"])
    else
```

---

```
```
find(:all)
end
end

# confirms existence of user
def self.authenticate(email, password)
  user = find_by_email(email)
  if user && user.password_hash == BCrypt::Engine.hash_secret(password, user.password_salt)
    user
  else
    nil
  end
end

# encrypts the password using BCrypt
def encrypt_password
  if password.present?
    self.password_salt = BCrypt::Engine.generate_salt
    self.password_hash = BCrypt::Engine.hash_secret(password, password_salt)
  end
end

# checks if a user has any type of access to a specific note
def has_access(note)
  has = Access.where("user_id = ? AND note_id= ?", self.id, note.id).first
  if has
    return true
  else
    return false
  end
end

# checks if user has read and write access to a specific note
def has_read_and_write_access(note)
  has = Access.where("user_id = ? AND note_id= ?", self.id, note.id).first
  if has
    if has.apermission == false
      return true
    end
  else
    return false
  end
end

app/controllers/accesses_controller.rb
class AccessesController < ApplicationController
  # POST /Accesss
  # POST /Accesss.json
  def create
    # stores the url to the page initiating the request
    session[:return_to] ||= request.referer
    # searches the database for the particular user to add
    collaborator = User.find_by_username(params[:access][:user])
    # checks if user is found or user is current_user
    if collaborator == nil
      session[:error] = 'NO user found with the specified Username'
      redirect_to session.delete(:return_to), notice: 'Access NOT created.'
      return
    elsif collaborator.id == current_user.id
      session[:error] = 'Cannot Give Access for yourself since you are the author of the note'
      redirect_to session.delete(:return_to), notice: 'Access NOT created.'
      return
  end
end
# find note to be shared
shared_note = Note.find(params[:note])
# find if the node has been shared in the past with the same user
@access = Access.where("user_id = ? AND note_id= ?", collaborator.id, shared_note.id).first

# if note is already shared with this user, update permission
if @access
  if params[:read_only]
    @access.apermission = params[:read_only]
  else
    @access.apermission = true # setting default value to read only
  end
  session[:error] = nil
else
  # create new access
  @access = Access.new
  if params[:read_only]
    @access.apermission = params[:read_only]
  else
    @access.apermission = true # setting default value to read only
  end
  @access.note = shared_note
  @access.user = collaborator
  session[:error] = @access.errors.full_messages.to_sentence
end

if @access.save
  redirect_to session.delete(:return_to), notice: 'Access was successfully granted.'
else
  redirect_to session.delete(:return_to), notice: 'Access NOT created.'
end
end

private
  # Use callbacks to share common setup or constraints between actions.
def set_Access
    @access = Access.find(params[:id])
  end

  # Never trust parameters from the scary internet, only allow the white list through.
def Access_params
    params.require(:Access).permit(:text, :user_id)
  end
end
```
ActiveRecord::Schema.define(version: 2013101310856) do

create_table "collections", force: true do |t|
  t.integer "user_id"
  t.datetime "created_at"
  t.datetime "updated_at"
  t.string "name"
end

create_table "followships", force: true do |t|
  t.integer "follower_id"
  t.integer "followed_id"
  t.datetime "created_at"
  t.datetime "updated_at"
end

create_table "notes", force: true do |t|
  t.text "content", limit: 255
  t.datetime "created_at"
  t.datetime "updated_at"
  t.integer "collection_id"
end

create_table "permissions", force: true do |t|
  t.integer "collection_id"
  t.integer "user_id"
  t.datetime "created_at"
  t.datetime "updated_at"
end

create_table "users", force: true do |t|
  t.string "name"
  t.string "username"
  t.datetime "created_at"
  t.datetime "updated_at"
  t.string "password_digest"
  t.string "remember_token"
end
end
```

```
app/controllers/permissions_controller.rb
#
# Controller for Permissions.
#
# Contains the following public actions:
#
#     create - action used for creating a permission
#     destroy - action used for deleting a permission
```
class PermissionsController < ApplicationController
  # Make sure user is signed in.
  before_action :signed_in_user
  
  def create
    user = User.find(params[:permission][:user_id])
    collectionID = params[:permission][:collection_id]
    # Redifine variables for AJAX/re-rendering
    getViewData(collectionID)

    # Verify the user
    if not current_user == @owner
      flash[:error] = "You can't edit permissions for a collection you don't own."
      redirect_to root_url
    else
      user.permit!(collectionID)
    end
  end

  ... 

  private
    # Retrieve data for view and verification.
    def getViewData(collectionID)
      @collection = Collection.find(collectionID)
      @permissions = @collection.permissions
      @owner = User.find(@collection.user_id)
      @followers = @owner.followers
      @followed = @owner.followed_users
    end
  end

app/models/user.rb

# User model.

# Ensures a user has a name, username and password.
# Sets relationships between a user and collections.
# Handles password encryption for the database.

# Code adapted from http://ruby.railstutorial.org/chapters/a-demo-app
# Username regex from: http://net.tutsplus.com/tutorials/other/8-regular-expressions-you-should-know/

class User < ActiveRecord::Base
  # Associate a user with many 'collections'
  # If a user is deleted, the collection should be deleted as well.
  has_many :collections, dependent: :destroy

  # Associate a user with a set of users that are followed 
  # Note: (followship = mix of relationship and follow...):
  has_many :followships, foreign_key: "follower_id", dependent: :destroy 
  has_many :followed_users, through: :followships, source: :followed

  # Associate a user with a set of users that follow the user 
  has_many :reverse_followships, foreign_key: "followed_id",
            class_name: "Followship",
            dependent: :destroy 
  has_many :followers, through: :reverse_followships, source: :follower
# Associate a user with many permissions.

```ruby
has_many :permissions, dependent: :destroy
```

```ruby
before_save { self.username = username.downcase }
before_create :create_remember_token
```

# Make sure name is entered, not too long,
# and has appropriate characters.

```ruby
validNameRegEx = /\A[a-z ,.'-]+\z/i
validates :name, presence: true,
           format: { with: validNameRegEx },
           length: { maximum: 70 }
```

... # Creates a random secure token
```ruby
def User.new_remember_token
  SecureRandom.urlsafe_base64
end
```

# Uses a hashing algorithm to encrypt a given token.
```ruby
def User.encrypt(token)
  Digest::SHA1.hexdigest(token.to_s)
end
```

# Searches users by the given name
#
# Parameters
# search - the query
def self.search(search)
  if search
    where 'name LIKE ?', "#{search}%"
  else
    scoped
  end
end
end
```

# Checks if the current user is "following" another user
#
# Parameters
# other_user - the other user
def following?(other_user)
  followships.find_by(followed_id: other_user.id)
end
```

# Set the user to "follow" another user
#
# Parameters
# other_user - the other user
def follow!(other_user)
  followships.create!(followed_id: other_user.id)
end
```

# Stops the user from following another user
#
# Parameters
# other_user - the other user
def unfollow!(other_user)
  followships.find_by(followed_id: other_user.id).destroy!
end
```

# Gives the user instance permissions to a collection
#
# Parameters
# collection_id - the id of the collection
def permit!(collection_id)
permissions.create!(user_id: self.id, collection_id: collection_id)
end

# Checks if the user instance has permissions to a collection
#
# Parameters
# collection_id - the id of the collection
def hasPermit?(collection_id)
  permissions.find_by(user_id: self.id, collection_id: collection_id)
end

# Removes permissions from the user instance for a collection
#
# Parameters
# collection_id - the id of the collection
def unpermit!(collection_id)
  permissions.find_by(user_id: self.id, collection_id: collection_id).destroy!
end

private
  # Creates a token to save remember user sessions.
  def create_remember_token
    # Add token to user as an attribute so that it is saved to
    # the database as well.
    self.remember_token = User.encrypt(User.new_remember_token)
  end
end
ActiveRecord::Schema.define(version: 20131010173206) do

create_table "folder_access_levels", force: true do |t|
  t.string "name"
  t.datetime "created_at"
  t.datetime "updated_at"
end

create_table "folders", force: true do |t|
  t.string "name"
  t.datetime "created_at"
  t.datetime "updated_at"
  t.integer "created_by"
end

create_table "folders_notes", force: true do |t|
  t.integer "folder_id"
  t.integer "note_id"
end

create_table "notes", force: true do |t|
  t.text "content"
  t.datetime "created_at"
  t.datetime "updated_at"
  t.string "xcoord"
  t.string "ycoord"
  t.string "height"
  t.string "width"
  t.string "title"
  t.integer "folder_id"
end

create_table "notifications", force: true do |t|

end
t.string  "title"
t.string  "content"
t.integer  "sequence_number"
t.integer  "user_id"
t.datetime  "created_at"
t.datetime  "updated_at"
end

create_table  "sessions", force: true do |t|
t.integer  "user_id"
t.string  "session_id"
t.datetime  "created_at"
t.datetime  "updated_at"
end

create_table  "shared_folder_folder_access_levels", force: true do |t|
t.integer  "shared_folder_id"
t.integer  "folder_access_level_id"
t.datetime  "created_at"
t.datetime  "updated_at"
end

create_table  "shared_folders", force: true do |t|
t.integer  "folder_id"
t.integer  "user_id"
t.datetime  "created_at"
t.datetime  "updated_at"
end

create_table  "users", force: true do |t|
t.string  "username"
t.string  "password"
t.datetime  "created_at"
t.datetime  "updated_at"
end
end

app/controllers/shared_folders_controller

class SharedFoldersController < ApplicationController
  #POST /shared_folders/create
  #creates a new shared folder relationship
  def create
    share_params = folder_share_params
    #find the specified folder to share
    folder = Folder.includes(:users).where({id:share_params[:share_folder_id]}).first

    respond_to do |format|
      #only if the folder exists and the owner of the folder is trying to share
      #do we allow the share to proceed
      if folder != nil and folder.owner.id == @loggedin_user.id
        new_user = User.where({id:share_params[:share_user_id]}).first
        if !folder.users.exists?({id:share_params[:share_user_id]})
          folder.users.push(new_user)
    end
      if folder.save
        #add read-only or read-and-write permissions to the shared folder
        shared_folder = SharedFolder.where({user_id:new_user.id,folder_id:folder.id}).first
        puts shared_folder.folder_access_level
        if share_params[:folder_access_level].to_s == "1"
          shared_folder.folder_access_level = FolderAccessLevel.where({id:1}).first
        else
          shared_folder.folder_access_level = FolderAccessLevel.where({id:2}).first
        end
      end
    end
  end
end
if shared_folder.save
  success_message = "Shared folder: " + share_params[:share_folder_id].to_s()
  + "with: " + share_params[:share_user_id].to_s()
  flash[:notice]=success_message

  #publish a notification
  notification_title = "New Shared Folder"
  notification_msg = "User: " + @loggedin_user.username.to_s() + " shared the folder: " + folder.name.to_s() + " with you!"
  publish_notification notification_title, notification_msg, new_user

  format.html { render :partial=>"folders/folder_share_current_user", :locals=>{:folder=>folder, :user=>new_user} }
  format.json { render json: {status: success_message} }

else
  flash[:notice]="Error, could not share folder!"
  format.html { redirect_to folders_url }
  format.json { render json: {status: "Could not share!"} }
end

else
  flash[:notice]="Error, could not share folder!"
  format.html { redirect_to folders_url }
  format.json { render json: {status: "Could not share!"} }
end

end

private
def folder_share_params
  params.permit(:share_folder_id, :share_user_id, :folder_access_level)
end

end

app/controllers/application_controller
require 'digest/sha1'
class ApplicationController < ActionController::Base
  before_action :set_current_session, :set_loggedin_user

  private

  #Attempts to log the user in using the :username and :password attributes specified in the
  #input param_args hash
  #@param param_args - hash that contains the :username and :password of the user that should be
  #logged in
  #returns true if the login succeeded, false otherwise
  def login param_args
    #create a new session
    @session = Session.new()

    #compute the password hash
    password = Digest::SHA1.hexdigest(param_args[:password])

    #check if a user with the specified username and password hash exists in the DB
    if User.exists?({username: param_args[:username], password: password})
      #get the user record
      @user = user.where({username: param_args[:username], password: password}).first

      #create a session hash that we can use to ID this user and their successful login
      #initialize the session hash
      @session.user = @user
      @session.save
    end
  end

  #if the current user is an admin, allow them to access the admin panel
  #else, redirect to the main page
  def admin_panel
    if @loggedin_user.is_admin
      render :partial=>"admin_panel"
    else
      redirect_to main_page_url
    end
  end
end
session_hash = Digest::SHA1.hexdigest(param_args[:username].to_s + param_args[:password].to_s)

#link session to user and save session
session_params_local = {session_id => session_hash, :user_id => @user.id}
@session = Session.new(session_params_local)

if @session.save
    #put session_hash that can be used to ID this user in the session hash table so that
    #the set_loggedin_user method can access the session_hash and lookup the user
    session[session[:user_session_id]] = session_hash;
    return true
else
    return false
end
end

#publishes a notification to the intended user.
#param title - string representing the title of the notification
#param content - string representing content of notification
#param intended_user_id - integer representing the id of the user who the notification is for
def publish_notification title, content, intended_user_id
    #increment sequence number of notification
    max_val = Notification.where({user_id: intended_user_id}).maximum(:sequence_number);
    seq_num = max_val
    if seq_num == nil
        seq_num = 0
    else
        seq_num = seq_num + 1
    end
    #create and save new notification
    new_notification = Notification.create({title: title, content: content,
                                          user_id: intended_user_id, sequence_number: seq_num });
end

def publish_notification_many title, content, users
    users.each do |user|
        publish_notification title, content, user.id
    end
end

#looks in the session hash for a valid session_hash representing a user's successful login
#it then sets an instance variable called @loggedin_user that is a user object representing the currently
#logged in user

def set_loggedin_user
    @loggedin_user = nil
    if Session.exists?({session_id: session[:user_session_id]})
        user_session = Session.where({session_id: session[:user_session_id]}).first
        @loggedin_user = user_session.user
    end
end

def set_current_session
    #looks in the session hash for a valid session_hash representing a user's successful login
    #it then uses this hash to look up the corresponding session record that
    #it then sets an instance variable called @current_session that is a session object
    #representing the currently
ActiveRecord::Schema.define(version: 20131013225458) do
  create_table "notes", force: true do |
    t.string "title"
    t.text "content"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.integer "user_id"
  end

  create_table "users", force: true do |
    t.string "name"
    t.string "email"
    t.string "password"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.string "salt"
  end

  create_table "users_notes", force: true do |
    t.integer "user_id"
    t.integer "note_id"
    t.boolean "read_write"
    t.datetime "created_at"
    t.datetime "updated_at"
  end
end