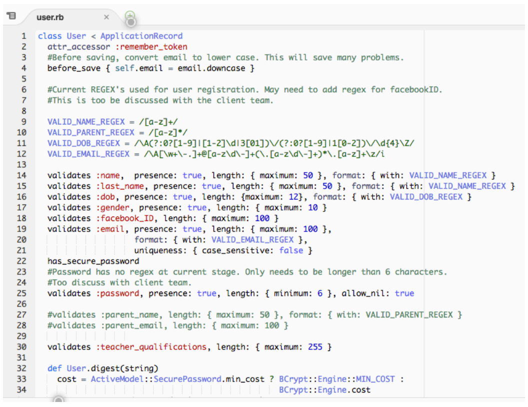
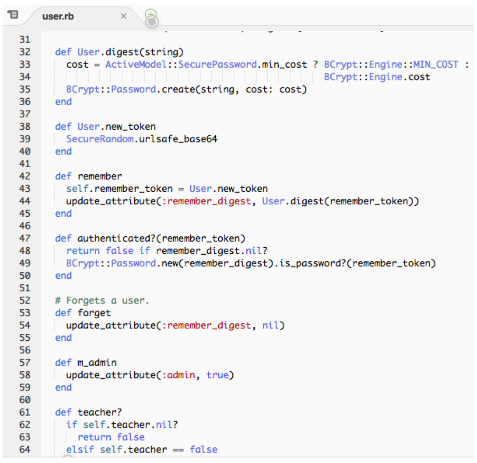
Model:



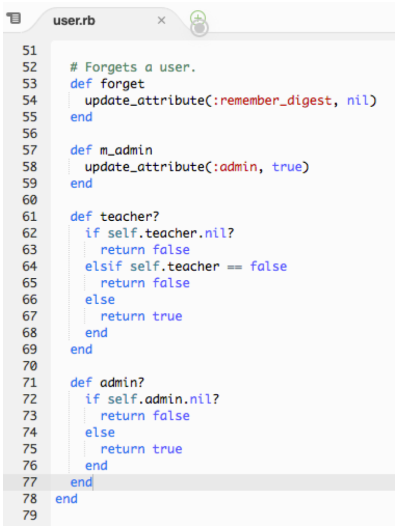
```ruby
class User < ApplicationRecord
  attr_accessor :remember_token
  #Before saving, convert email to lower case. This will save many problems.
  before_save { self.email = email.downcase }

  #Current REGEX's used for user registration. May need to add regex for facebookID.
  #This is too be discussed with the client team.

  VALID_NAME_REGEX = /[a-z]+/
  VALID_PARENT_REGEX = /[a-z]*/
  VALID_DOB_REGEX = /\A(?:0?[1-9]|[1-2]\d|3[01])\/(?:0?[1-9]|1[0-2])\/\d{4}\Z/
  VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-]+(\.[a-z\d\-]+)*\.[a-z]+\z/i

  validates :name,  presence: true, length: { maximum: 50 }, format: { with: VALID_NAME_REGEX }
  validates :last_name, presence: true, length: { maximum: 50 }, format: { with: VALID_NAME_REGEX }
  validates :dob, presence: true, length: {maximum: 12}, format: { with: VALID_DOB_REGEX }
  validates :gender, presence: true, length: { maximum: 10 }
  validates :facebook_ID, length: { maximum: 100 }
  validates :email, presence: true, length: { maximum: 100 },
                    format: { with: VALID_EMAIL_REGEX },
                    uniqueness: { case_sensitive: false }
  has_secure_password
  #Password has no regex at current stage. Only needs to be longer than 6 characters.
  #Too discuss with client team.
  validates :password, presence: true, length: { minimum: 6 }, allow_nil: true

  #validates :parent_name, length: { maximum: 50 }, format: { with: VALID_PARENT_REGEX }
  #validates :parent_email, length: { maximum: 100 }

  validates :teacher_qualifications, length: { maximum: 255 }

  def User.digest(string)
    cost = ActiveModel::SecurePassword.min_cost ? BCrypt::Engine::MIN_COST :
                                                  BCrypt::Engine.cost
```

Figure 1. Model Code part 1



```ruby

  def User.digest(string)
    cost = ActiveModel::SecurePassword.min_cost ? BCrypt::Engine::MIN_COST :
                                                  BCrypt::Engine.cost
    BCrypt::Password.create(string, cost: cost)
  end

  def User.new_token
    SecureRandom.urlsafe_base64
  end

  def remember
    self.remember_token = User.new_token
    update_attribute(:remember_digest, User.digest(remember_token))
  end

  def authenticated?(remember_token)
    return false if remember_digest.nil?
    BCrypt::Password.new(remember_digest).is_password?(remember_token)
  end

  # Forgets a user.
  def forget
    update_attribute(:remember_digest, nil)
  end

  def m_admin
    update_attribute(:admin, true)
  end

  def teacher?
    if self.teacher.nil?
      return false
    elsif self.teacher == false
```

Figure 2. Model Code part 2

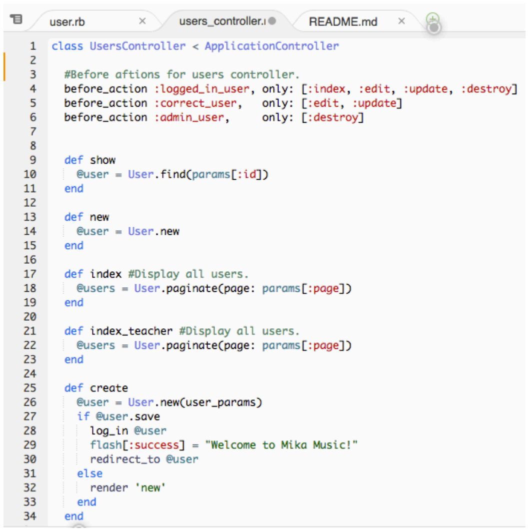


```ruby

  # Forgets a user.
  def forget
    update_attribute(:remember_digest, nil)
  end

  def m_admin
    update_attribute(:admin, true)
  end

  def teacher?
    if self.teacher.nil?
      return false
    elsif self.teacher == false
      return false
    else
      return true
    end
  end

  def admin?
    if self.admin.nil?
      return false
    else
      return true
    end
  end
end

```

Figure 3. Model Code Part 3

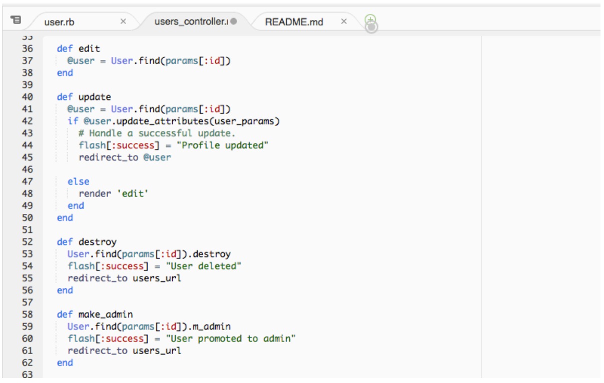
The above model was implemented to allow the controller to achieve specific actions such as determine if a user is an admin or a teacher. Validations were put in place to validate all user input from the forms however as can be seen in Figure 1, some validation was not implemented as responses from the client team were pending. For these our team wanted to confirm if passwords needed validation and if parent information should be included in the first release. The model was continuously subject to change until the very end of release 1 and thus some changes may have not been fully implemented. Required changes are to be added in release 2.

Controller:



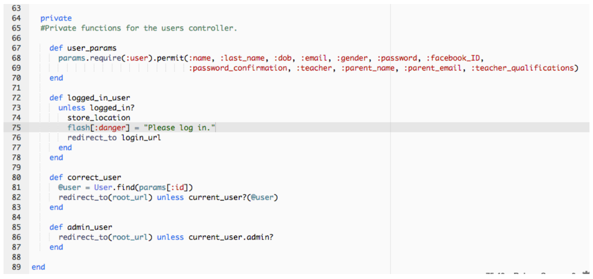
```
class UsersController < ApplicationController

  #Before aftions for users controller.
  before_action :logged_in_user, only: [:index, :edit, :update, :destroy]
  before_action :correct_user,   only: [:edit, :update]
  before_action :admin_user,     only: [:destroy]


  def show
    @user = User.find(params[:id])
  end

  def new
    @user = User.new
  end

  def index #Display all users.
    @users = User.paginate(page: params[:page])
  end

  def index_teacher #Display all users.
    @users = User.paginate(page: params[:page])
  end

  def create
    @user = User.new(user_params)
    if @user.save
      log_in @user
      flash[:success] = "Welcome to Mika Music!"
      redirect_to @user
    else
      render 'new'
    end
  end
```

Figure 4. User Controller part 1



```
  def edit
    @user = User.find(params[:id])
  end

  def update
    @user = User.find(params[:id])
    if @user.update_attributes(user_params)
      # Handle a successful update.
      flash[:success] = "Profile updated"
      redirect_to @user

    else
      render 'edit'
    end
  end

  def destroy
    User.find(params[:id]).destroy
    flash[:success] = "User deleted"
    redirect_to users_url
  end

  def make_admin
    User.find(params[:id]).m_admin
    flash[:success] = "User promoted to admin"
    redirect_to users_url
  end
```

Figure 5. User Controller part 2



```
  private
  #Private functions for the users controller.

    def user_params
      params.require(:user).permit(:name, :last_name, :dob, :email, :gender, :password, :facebook_ID,
                                   :password_confirmation, :teacher, :parent_name, :parent_email, :teacher_qualifications)
    end

    def logged_in_user
      unless logged_in?
        store_location
        flash[:danger] = "Please log in."
        redirect_to login_url
      end
    end

    def correct_user
      @user = User.find(params[:id])
      redirect_to(root_url) unless current_user?(@user)
    end

    def admin_user
      redirect_to(root_url) unless current_user.admin?
    end

end
```

Figure 6. User Controller part 3

The user controller handles the creation, viewing, updating and deletion of the users in the website. Two functions are used to display the users of the website (one for students and another for students) as seen in Figure 4. Originally it was planned for the index teacher to only return users that are teachers however after some difficulty implementing this, the index_teacher function returns all users to the page and then the view separates the teachers from all other users. This is an inefficient solution that should be changed in future sprints. The create function as seen in Figure 4, creates the user with the inputted form parameters and adds them to the database. On successful save to the database, a flash card is displayed to the user. Update works in a similar fashion.

The biggest design decision for the controllers was the choice of mas assignable user parameters (ass seen in Figure 6). This decision was based on what the client team believed the users should have the ability to assign to themselves. Originally, teachers were unable to assign themselves as teachers and it was up to the admin to go through and promote the teacher from a student to a teacher. This was later changed as reflected above and now anyone could elect themselves as a teacher. Future releases will fix this by sending a confirmation email to the admins on attempted teacher registration.