МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**“ТВЕРСКОЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

**(ФГБОУ ВО «ТвГУ»)**

Математический факультет

Кафедра Компьютерной безопасности и

математических методов управления

**ОТЧЕТ**

по производственной практике

студента 4 курса 45 группы

Коханчика Антона Дмитриевича

Специальность 10.05.01 Компьютерная безопасность

Специализация «Математические методы защиты информации»

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_.\_\_.20\_\_

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_.\_\_.20\_\_

Руководитель ООП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_.\_\_.20\_\_

Руководитель практики от предприятия

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_.\_\_.20\_\_

# Содержание

[Содержание 2](#_Toc487140363)

[Введение 4](#_Toc487140364)

[Теоретическое описание задачи 5](#_Toc487140365)

[Функциональные требования 5](#_Toc487140366)

[Технические требования 6](#_Toc487140367)

[Требования к интерфейсу 6](#_Toc487140368)

[Этапы решения задачи 7](#_Toc487140369)

[Описание моделей и их связей. 8](#_Toc487140370)

[Заполнение базы данных тестовыми значениями 9](#_Toc487140371)

[Валидация входящих данных 9](#_Toc487140372)

[Реализация аутентификации для нескольких пользователей с разными ролями 9](#_Toc487140373)

[Описание работы контроллеров сервера, соответствующих моделям, обработка AJAX на стороне сервера. 10](#_Toc487140374)

[Приведения кода к виду, приведённому в руководстве для оформления 10](#_Toc487140375)

[Заключение 11](#_Toc487140376)

[Ресурсы: 12](#_Toc487140377)

[Приложение 1 13](#_Toc487140378)

[app\models\event.rb 13](#_Toc487140379)

[db\migrate\20170621102430\_create\_events.rb 13](#_Toc487140380)

[app\models\order.rb 13](#_Toc487140381)

[db\migrate\20170621103510\_create\_orders.rb 13](#_Toc487140382)

[app\models\room.rb 14](#_Toc487140383)

[db\migrate\20170621103254\_create\_rooms.rb 14](#_Toc487140384)

[db\migrate\20170621112347\_create\_join\_table\_rooms\_events.rb 14](#_Toc487140385)

[Приложение 2 15](#_Toc487140386)

[db\seeds.rb 15](#_Toc487140387)

[Приложение 3 17](#_Toc487140388)

[app\models\concerns\date\_validators.rb 17](#_Toc487140389)

[Приложение 4 19](#_Toc487140390)

[app\models\user.rb 19](#_Toc487140391)

[db\migrate\20170622084335\_devise\_create\_users.rb 19](#_Toc487140392)

[Приложение 5 20](#_Toc487140393)

[app\controllers\home\_controller.rb 20](#_Toc487140394)

[app\controllers\users\_controller.rb 20](#_Toc487140395)

[app\controllers\application\_controller.rb 20](#_Toc487140396)

[app\controllers\order\_controller.rb 20](#_Toc487140397)

# Введение

Целью прохождения производственной практики является реализация полученных теоретических знаний, умений и навыков, а также получение представления о практической деятельности реальной организации.

В период с 12 июня 2017 г. по 9 июля 2017 г. я проходил производственную практику в Студии дизайна «OCBS» под руководством ее арт-директора Ткачёва Павла Сергеевича.

Студия «OCBS» специализируется на работе в области брендинга и графического дизайна, предлагая комплексные решения для бизнеса: от нейминга, создания логотипа и разработки фирменного стиля, до дизайна, полиграфии, создания сайта и упаковки.

Основным направлением деятельности Студии является создание и продвижение интернет-сайтов. Среди заказчиков компании присутствуют такие известные организации, как: Тверская Академическая Областная Филармония, Высшая школа предпринимательства, Университет Синергия и проч.

# Теоретическое описание задачи

Перед командой студентов, проходящих практику в Студии, была поставлена задача разработки сайта, предоставляющего возможность бронирования комнат для проведения мероприятий для организации-заказчика.

Продукт призван предоставлять сотрудникам организации актуальную информацию о занятости/свободности комнат, а также давать возможность забронировать их для проведения собственных мероприятий.

Цель разработки: оптимизировать процесс согласования дат и времени проведения мероприятий сотрудниками организации-заказчика, а также создать единое окно администрирования процессов бронирования для ее Службы технической поддержки.

Согласно предоставленному описанию проекта, разрабатываемый продукт должен удовлетворять ряду требований.

## Функциональные требования

1. Доступ к базе данных событий для всех пользователей организации через сеть Интернет;
2. Возможность разграничения прав пользователей;
3. Возможность создания, редактирования и удаления таких компонентов приложения, как Комнаты, События и Брони в соответствии с заданной ролью пользователя.

## Технические требования

При разработке необходимо использовать веб-фреймворк Ruby on Rails [1].

В качестве Системы Управления Базами Данных предпочтительно использование PostgreSQL, однако также возможно использование иных SQL-подобных решений, поддерживающих подходы миграции данных и структуры базы данных через описание состояний. Для реализации аутентификации пользователей в приложении необходимо использовать гем Devise [4].

## Требования к интерфейсу

Основным разделом приложения является страница броней, которая существует для каждой отслеживаемой комнаты и которая выполнена в виде календаря. Каждая из таких страниц должна давать представление о проводимых в комнате мероприятиях, а также датах и времени, когда выбранная комната свободна/занята. Детали реализации интерфейса оставлены на усмотрение разработчиков.

# Этапы решения задачи

Весь процесс разработки приложения был разделен руководителем на 3 составляющих:

1. Backend
2. Frontend
3. Testing

В ходе прохождения практики, мне была назначена роль разработчика серверной стороны архитектуры клиент-сервер (backend), но из-за тесной связи этих двух составляющих, я иногда писал и frontend-составляющие сервиса.

Мои задачи:

* описание моделей и их связей
* заполнение базы данных тестовыми значениями
* валидация входящих данных
* реализация аутентификации для нескольких пользователей с разными ролями
* описание работы контроллеров сервера, соответствующих моделям, обработка AJAX на стороне сервера.
* Дополнительно было получено задание приведения кода к виду, приведённому в руководстве для оформления кода ruby [5]

# Описание моделей и их связей.

В приложении было реализовано несколько моделей:

* Пользователь:
имеет почту, пароль, роль (администратор или рядовой пользователь)
* Событие:
создаётся пользователем, может проходить в нескольких комнатах и неоднократно за время своего проведения
имеет дату-время начала, дату окончания, название, описание
* Комната:
в ней могут проходить события посредством бронирования
имеет время начала и окончания работы, название, описание
* Бронь:
связывает комнату и событие, отличается от события продолжительностью: не дольше времени события и не дольше времени работы комнаты
имеет дату-время начала и окончания

Ruby on Rails уже реализован шаблон проектирования ActiveRecord, позволяющий работать с моделями сущностей, а не с таблицами базы данных, что упрощает разработку.

Код в приложении 1.

# Заполнение базы данных тестовыми значениями

Тут ничего особенно сложного – простая программа (т.н. rake-task), записывающая в БД случайные данные, здесь нам очень помогает гем Faker, в котором имеется множество различных видов случайных значений: имена, фамилии, email, пароли, даты, персонажи, … [6]

Код в приложении 2

# Валидация входящих данных

В ActiveRecord уже присутствуют стандартные проверки на присутствие (не пустоту) полей и их уникальность, однако в данном проекте нужно проверять дату-время по ряду требований:

* Время начала (события, брони, работы комнаты) не может быть позже времени окончания, а также длительность не меньше 15 минут
* Время брони должна попадать во время события и работы комнаты
* Брони не должны пересекаться по времени в одной комнате

Код в приложении 3

# Реализация аутентификации для нескольких пользователей с разными ролями

Здесь всю работу отдаём гему Devise [4]. Нужно лишь указать в модели users, что она управляется гемом, а также настроить элементы страниц, чтобы они показывались или не показывались пользователям с разными ролями

Код в приложении 4

# Описание работы контроллеров сервера, соответствующих моделям, обработка AJAX на стороне сервера.

Контроллер – часть архитектуры, отвечающая за обработку запросов от пользователя, контроллер получает запрос, берёт данные из БД и генерирует ответ (чаще всего в виде html-страницы, но не всегда)

Здесь описываются переменные, которые будут переданы во view-файл (который позже станет частью html-страницы), а также действия по сохранению полученных сервером данных из запроса в БД (или даже удаление записей из БД)

Коды некоторых контроллеров в приложении 5

# Приведения кода к виду, приведённому в руководстве для оформления

Здесь использовался гем Rubocop [7], он сканирует код в приложении и указывает на места, в которых требования не соблюдены, также его работу можно настроить, чтобы он игнорировал некоторые файлы или требования.

# Заключение

Производственная практика – один из важнейших элементов программы обучения. Он позволяет оказаться в рабочей среде реальной организации и получить колоссальный практический опыт.

Во время прохождения производственной практики мне удалось познакомиться с реальным процессом работы над проектом. Были затронуты все основные этапы: от обсуждения основных требований к приложению и составления технического задания, до непосредственной разработки и тестирования.

Я углубил свои знания по Ruby on Rails, Javascript, HTML, познакомился с новыми ruby-gem -ами, получил опыт в решении различного рода задач (в том числе задач в неизвестной сфере). Результатом стало работающее web-приложение, которое, возможно, вскоре окажется на сервере и появится на сайте заказчика.

# Ресурсы:

1. <http://rubyonrails.org/> - официальный сайт Rails
2. <http://rusrails.ru/> - руководство по Rails
3. <https://rubygems.org/> - документации большинства gem-ов
4. <https://github.com/plataformatec/devise/> - gem Devise
5. <https://github.com/arbox/ruby-style-guide/blob/master/README-ruRU.md> - руководство по оформлению кода Ruby
6. <https://github.com/stympy/faker> - gem Faker
7. <https://github.com/bbatsov/rubocop> - gem Rubocop

# Приложение 1

## app\models\event.rb

class Event < ApplicationRecord

 include DateValidators

 include ActiveModel::Validations

 has\_and\_belongs\_to\_many :rooms

 belongs\_to :user

 has\_many :orders, dependent: :destroy

 # проверка на присутствие

 validates :title, :begin\_datetime, :end\_datetime, :user\_id, presence: true

 # проверка на integer и not\_null

 validates :user\_id, numericality: { only\_integer: true }

 validates\_with EventValidator

end

## db\migrate\20170621102430\_create\_events.rb

class CreateEvents < ActiveRecord::Migration[5.1]

 def change

 create\_table :events do |t|

 t.string :title, null: false

 t.datetime :begin\_datetime, null: false

 t.datetime :end\_datetime, null: false

 t.text :description

 # пользователь, создавший событие

 t.integer :user\_id, null: false

 t.timestamps

 end

 end

end

## app\models\order.rb

class Order < ApplicationRecord

 include DateValidators

 include ActiveModel::Validations

 belongs\_to :room

 belongs\_to :event

 # проверка на присутствие

 validates :begin\_datetime, :end\_datetime, :room\_id, :event\_id, presence: true

 # проверка на integer и not\_null

 validates :event\_id, :room\_id, numericality: { only\_integer: true }

 validates\_with OrderValidator

end

## db\migrate\20170621103510\_create\_orders.rb

class CreateOrders < ActiveRecord::Migration[5.1]

 def change

 create\_table :orders do |t|

 t.datetime :begin\_datetime, null: false

 t.datetime :end\_datetime, null: false

 t.integer :room\_id, null: false

 t.integer :event\_id, null: false

 t.timestamps

 end

 end

end

## app\models\room.rb

class Room < ApplicationRecord

 include DateValidators

 include ActiveModel::Validations

 has\_many :orders, dependent: :destroy

 has\_and\_belongs\_to\_many :events

 # проверка на присутствие

 validates :title, :begin\_work\_time, :end\_work\_time, presence: true

 validates\_with RoomValidator

end

## db\migrate\20170621103254\_create\_rooms.rb

class CreateRooms < ActiveRecord::Migration[5.1]

 def change

 create\_table :rooms do |t|

 t.string :title, null: false

 t.time :begin\_work\_time, null: false

 t.time :end\_work\_time, null: false

 t.text :description

 t.timestamps

 end

 end

end

## db\migrate\20170621112347\_create\_join\_table\_rooms\_events.rb

class CreateJoinTableRoomsEvents < ActiveRecord::Migration[5.1]

 def change

 create\_join\_table :rooms, :events do |t|

 t.index :room\_id

 t.index :event\_id

 end

 end

end

# Приложение 2

## db\seeds.rb

users\_number = 8

events\_number = 12

rooms\_number = 6

datetime = DateTime.now.change(hour: 9)

# Пользователи

users\_number.times do

 email\_tmp = Faker::Internet.unique.email

 User.create(

 email: email\_tmp,

 password: email\_tmp[0, 2] + '12345'

 )

end

User.create(

 email: 'admin@example.com',

 password: 'password',

 role: 'admin'

)

User.create(

 email: 'admin@admins.net',

 password: 'admin2',

 role: 'admin'

)

puts 'users created'

# События

events\_number.times do |i|

 title\_tmp = Faker::Book.title

 Event.create(

 title: title\_tmp,

 description: title\_tmp + title\_tmp + title\_tmp,

 begin\_datetime: datetime + i.minutes,

 end\_datetime: datetime + 10.days,

 user\_id: rand(0..users\_number - 1)

 )

end

Event.create(

 title: 'My\_event',

 description: 'la - la - la',

 begin\_datetime: datetime - 10.hours,

 end\_datetime: datetime + 1.days,

 user\_id: rand(0..users\_number - 1)

)

Event.create(

 title: 'My\_event\_3',

 description: 'la - la - la',

 begin\_datetime: datetime - 2.days,

 end\_datetime: datetime - 1.days,

 user\_id: rand(0..users\_number - 1)

)

puts 'events created'

# Комнаты

rooms\_number.times do |i|

 Room.create(

 title: 'room\_' + (i + rand(1..6) \* 10).to\_s,

 begin\_work\_time: Time.new(2000, 1, 1, 11, 0, 0, '+03:00'),

 end\_work\_time: Time.new(2000, 1, 1, 21, 0, 0, '+03:00'),

 description: 'this is a room'

 )

end

Event.all.each do |e|

 Room.all.each do |r|

 e.rooms << r if rand(0..4).zero?

 end

end

puts 'rooms created'

# Брони

Room.all.each do |r|

 r.events.each\_with\_index do |e, i|

 b = e.begin\_datetime.change(hour: 9, min: 0)

 o = Order.new(

 begin\_datetime: b + i.hours,

 end\_datetime: b + (i + 1).hours,

 room\_id: r[:id],

 event\_id: e[:id]

 )

 # Если по какой-то причине бронь не состоялась, удаляем связь комнаты с событием

 e.rooms.destroy(r) unless o.save

 end

end

puts 'orders created'

puts 'success'

# Приложение 3

## app\models\concerns\date\_validators.rb

module DateValidators

 # Проверяет валидность даты/времени (конец раньше начала, пересечение, выход из промежутка...)

 extend ActiveSupport::Concern

 class EventValidator < ActiveModel::Validator

 def validate(record)

 if record.begin\_datetime.blank? || record.end\_datetime.blank?

 record.errors[:text] << 'Неправильная дата'

 return

 end

 if (record.end\_datetime - record.begin\_datetime).to\_i / 60 < 15

 record.errors[:text] << 'Длительность < 15 минут'

 end

 end

 end

 class RoomValidator < ActiveModel::Validator

 def validate(record)

 if record.begin\_work\_time.blank? || record.end\_work\_time.blank?

 record.errors[:text] << 'Неправильная дата'

 return

 end

 if (record.end\_work\_time - record.begin\_work\_time).to\_i / 60 < 15

 record.errors[:text] << 'Длительность < 15 минут'

 end

 if (record.end\_work\_time - record.begin\_work\_time).to\_i / 3600 > 24

 record.errors[:text] << 'Длительность > 1 дня'

 end

 end

 end

 class OrderValidator < ActiveModel::Validator

 def validate(record)

 if record.begin\_datetime.blank? || record.end\_datetime.blank?

 record.errors[:text] << 'Неправильная дата'

 return

 end

 if (record.end\_datetime - record.begin\_datetime).to\_i / 60 < 15

 record.errors[:text] << 'Длительность < 15 минут'

 end

 if record.event.blank?

 record.errors[:text] << 'Нет события'

 return

 end

 if record.end\_datetime > record.event.end\_datetime || record.begin\_datetime < record.event.begin\_datetime

 record.errors[:text] << 'Мимо события'

 end

 if record.room.blank?

 record.errors[:text] << 'Нет комнаты'

 return

 end

 if record.end\_datetime.min + 60 \* record.end\_datetime.hour > record.room.end\_work\_time.min + 60 \* record.room.end\_work\_time.hour ||

 record.begin\_datetime.min + 60 \* record.begin\_datetime.hour < record.room.begin\_work\_time.min + 60 \* record.room.begin\_work\_time.hour

 record.errors[:text] << 'Мимо комнаты'

 end

 record.room.orders.each do |order|

 if (record.begin\_datetime < order.begin\_datetime && record.end\_datetime > order.begin\_datetime) ||

 (record.begin\_datetime < order.end\_datetime && record.end\_datetime > order.end\_datetime) ||

 (record.begin\_datetime >= order.begin\_datetime && record.end\_datetime <= order.end\_datetime)

 record.errors[:text] << 'Пересечение событий'

 end

 end

 end

 end

end

# Приложение 4

## app\models\user.rb

class User < ApplicationRecord

 # Include default devise modules. Others available are:

 # :confirmable, :lockable, :timeoutable and :omniauthable

 devise :database\_authenticatable

 has\_many :events, dependent: :destroy

 # проверка на присутствие

 validates :email, :password, presence: true

 # на всякий случай, проверка уникальности email, хотя добавлять пользователей не будем

 validates :email, uniqueness: true

end

## db\migrate\20170622084335\_devise\_create\_users.rb

class DeviseCreateUsers < ActiveRecord::Migration[5.1]

 def change

 create\_table :users do |t|

 ## Database authenticatable

 t.string :email, null: false, default: ''

 t.string :encrypted\_password, null: false, default: ''

 t.string :role, null: false, default: 'user'

 t.timestamps null: false

 end

 add\_index :users, :email, unique: true

 end

end

# Приложение 5

## app\controllers\home\_controller.rb

class HomeController < ApplicationController

 def index

 @events\_top = Event.where('end\_datetime > ?', DateTime.now).order(:begin\_datetime).limit(10)

 end

end

## app\controllers\users\_controller.rb

class UsersController < ApplicationController

 def show

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

 @events = Event.all

 end

end

## app\controllers\application\_controller.rb

class ApplicationController < ActionController::Base

 protect\_from\_forgery with: :exception

 # если вошёл, остаться на этой же странице, вышел – на главную

 def after\_sign\_in\_path\_for(resource\_or\_scope)

 request.referrer

 end

 def after\_sign\_out\_path\_for(resource\_or\_scope)

 root\_path

 end

end

## app\controllers\order\_controller.rb

class OrdersController < ApplicationController

 def index

 @orders = Order.where(room\_id: params[:room\_id])

 end

 def show

 @order = Order.find(params[:id])

 end

 def new

 @order = Order.new

 @events = current\_user.role == 'admin' ? Event.all : current\_user.events

 @rooms = Room.all

 @room\_id = params[:room\_id]

 @event\_id = params[:event\_id]

 end

 def create

 @order = Order.new(order\_params)

 if @order.save

 unless @order.event.rooms.exists?(@order.room.id)

 @order.event.rooms << @order.room

 end

 redirect\_to session.delete(:return\_to)

 else

 flash[:notice] = @order.errors['text'].last

 puts(flash[:notice])

 redirect\_to action: 'new', room\_id: @order.room\_id, event\_id: @order.event\_id

 end

 end

 def edit

 @order = Order.find(params[:id])

 @events = current\_user.role == 'admin' ? Event.all : current\_user.events

 @rooms = Room.all

 end

 def update

 @order = Order.find(params[:id])

 @events = current\_user.role == 'admin' ? Event.all : current\_user.events

 @rooms = Room.all

 if @order.update(order\_params)

 redirect\_to @order

 else

 render 'edit'

 end

 end

 def destroy

 @order = Order.find(params[:id])

 room\_id = @order.room\_id

 event\_id = @order.event\_id

 @order.destroy

 if Order.where(room\_id: room\_id, event\_id: event\_id).empty?

 Room.find(room\_id).events.destroy(event\_id)

 end

 redirect\_to session.delete(:return\_to)

 end

 def order\_params

 params.require(:order).permit(:room\_id, :begin\_datetime, :end\_datetime, :event\_id)

 end

 def assist

 respond\_to do |format|

 format.html do

 render partial: 'orders/assist',

 locals: {

 room: Room.find(params[:room\_id]),

 event: Event.find(params[:event\_id])

 }

 end

 end

 end

end