МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет «Вычислительная техника» (наименование)

Кафедра «Системы автоматизированного проектирования» (наименование)

Направление подготовки 09.03.01 «Информатика и вычислительная техника» (код и наименование)

Профиль «Системы автоматизированного проектирования» (наименование)

БАКАЛАВРСКАЯ РАБОТА

на тему

Разработка компонентов медицинской информационной системы.

Студент _____ 25.05.2017 (подпись, дата) Тупаев Р.Е. (фамилия, инициалы)

Руководитель _____ 1.06.17 (подпись, дата) Бурукина И.П. (фамилия, инициалы)

Нормоконтролёр ст.преп. каф. САПР ПГУ (должность, место работы) _____ 27.05.17 (подпись, дата) _____ (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ (подпись) _____ (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ (подпись) _____ (фамилия, инициалы)

Пенза 2017

Реферат

Пояснительная записка 70 листов, 9 рисунков, 1 таблица, 12 источников, 2 приложения.

WEB, МЕДИЦИНСКАЯ ИНФОРМАЦИОННАЯ СИСТЕМА, RUBY, RUBY ON RAILS, POSTGRESQL, NGINX, HTML5, CSS.

Объектом разработки является компонент медицинской информационной системы, предназначенный для повышения эффективности регистратуры за счет автоматизации учета, безопасного хранения данных, разграниченного доступ к информации, помощи в планировании работы врачей.

Цель работы - разработки является компонент медицинской информационной системы, основными функциями которого являются: ввод данных, разграничение доступа, поиск информации, оформление лерасписания лечения пациентов и учет исполнения.

В процессе разработки проводилось изучение принципов построения WEB-приложений, в частности технологии разработки приложений с использованием фреймворка Ruby on Rails.

В результате выполнения дипломной работы создано WEB-приложение для автомтизации работы регистратуры в санатории.

Разработанный сервис следует применять для автоматизации работы регистратуры санатория.

СОДЕРЖАНИЕ

ПЕРЧЕНЬ СОКРАЩЕНИЙ 6
ВВЕДЕНИЕ 7
ПОСТАНОВКА ЗАДАЧИ 9
АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 10
ОБЗОР ТЕХНОЛОГИЙ WEB-ПРОГРАММИРОВАНИЯ 16
Язык программирования Ruby 16
Фреймворк Ruby on Rails 17
Среда разработки RubyMine 18
SQLite3 18
Proxy-сервер 19
ПРОЕКТИРОВАНИЕ КОМПОНЕНТОВ МЕДИЦИНСКОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ 20
Описание метода разработки 20
Подготовительный этап 21
Проектирование механизма аутентификации 23
Проектирование системы управления содержимым 25
Проектирование системы разграничения доступа 26
Проектирование системы управления расписанием 30
ПРОЕКТИРОВАНИЕ БД 32
ПРОЕКТИРОВАНИЕ КОНТРОЛЛЕРОВ 34
Обработка URL 35
Организация поиска 35
Просмотр списка записей 36
Просмотр подробностей записи 37
Дизайн приложения 38

ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 39
Объект испытаний 39
Цель испытаний 39
Требования к программе 39
Методы испытания 40
Создание записи 41
Ошибки и способы их устранения 43
ФУНКЦИОНИРОВАНИЕ ПРОГРАММЫ 45
Диаграмма использования 45
Описание схемы работы системы 46
Диаграмма классов 47
Логическая модель данных 48
Физическая модель данных 48
ЗАКЛЮЧЕНИЕ 50
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 51
ПРИЛОЖЕНИЕ А. СХЕМЫ 53
ПРИЛОЖЕНИЕ Б. ЛИСТИНГИ 55
Приложение Б1. Модели 56
Приложение Б2. Контроллеры 58
Приложение Б3. Представление 64
Приложение Б4. Миграции 65
Приложение Б5. Создание БД 68

ПЕРЧЕНЬ СОКРАЩЕНИЙ

ROR – (Ruby on Rails) фреймворк для создания web-приложений на языке ruby;

MVC – (Model-View-Controller) модель-представление-контроллер;

ORM – (Object relative mapping) объектно-реляционное отображение;

IDE – (Integrated Development Environment) интегрированная среда разработки;

CRUD – (Create Read Update Delete) создать прочитать обновить удалить;

REST – (Representational State Transfer) архитектурный стиль взаимодействия компонентов распределённого приложения в сети;

СУБД – система управления базой данных

ОС – операционная система;

ПО – программное обеспечение;

ПГУ – Пензенский государственный университет;

ЕСПД – единая система программной документации;

БД – база данных.

ВВЕДЕНИЕ

Одним из важных направлений развития современного общества является его глобальная информатизация.

Автоматизация процессов, на сегодняшний день, стала насущной необходимостью в условиях жесткой конкуренции. Именно автоматизация и информтизация дает новые возможности любой организации по ускорению работы, позволяет опередить конкурентов при принятии как оперативных, так и стратегических решений.

В связи с ростом развития технологий, предприятию все больше необходима автоматизация процессов, в частности работы с документами. Причин этому много. Во-первых, информацию необходимо обрабатывать как можно быстрее и качественнее, подчас информационные потоки не менее важны, чем материальные. Во-вторых, утеря информации или ее попадание в чужие руки может обойтись весьма дорого. Можно выделить ряд проблем, общих для тех организаций, где работа с документами ведется традиционным способом:

- потеря документов;
- накапливается множество документов, назначение и источник которых неясны;
- документы и информация, содержащаяся в них, попадают в чужие руки;
- создается несколько копий одного и того же документа - на бумагу и копирование документов тратиться немало средств.

Внедрение системы электронного документооборота позволяет решить все эти проблемы, а также:

- обеспечит слаженную работу всех подразделений;

- упростит работу с документами, повысит ее эффективность;
- повысит производительность труда сотрудников;
- повысит оперативность доступа к информации;
- позволит разграничить права доступа сотрудников к информации.

Автоматизированная система управления базируется на современных программных средствах с использованием новейших информационных технологий, которые удовлетворяют самым высоким требованиям по защите информации и быстродействию системы, эффективно распределяют нагрузку в режиме коллективного доступа к данным.

ПОСТАНОВКА ЗАДАЧИ

Задачей дипломного проектирования является написание и отладка компонента медицинской информационной системы, предназначенный для автоматизации учета данных, безопасного хранения данных.

Система должна быть представлена WEB-сервисом, в котором сотрудники регистратуры санатория имеют возможность добавления, изменения и удаления информации о пациентах. Должен быть реализован удобный поиск информации. В системе должна быть реализована возможность оформления расписания лечения для пациентов, а также учет его исполнения. Система должна помогать в планировании работы врачей, кабинетов и медицинского оборудования.

В приложении должно быть предусмотрено разграничение доступа. Работник регистратуры должен иметь доступ только с записи и учету пациентов. Врачи должны иметь доступ к расписанию лечения пациентов и расписанию работы кабинетов и медицинского оборудования на редактирование, к другим разделам только на чтение. Руководство санатория должен иметь доступ ко всем разделам системы и меть возможность изменять любые данные. Доступ к любому разделу системы возможен только после аутентификации.

Систем должна быть разработана на языке программирования Ruby с использованием фреймворка Ruby on Rails. Среда исполнения интерпретатор Ruby 2.3.

АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Системы управления информацией (порталы) — обеспечивают агрегирование, управление и доставку информации через сети Internet, Intranet и Extranet. Эти технологии обеспечивают фундамент создания информационных порталов. Системы управления информацией дают возможность организациям накапливать и использовать экспертизу в распределенной корпоративной среде на основе использования бизнес-правил, контекста и метаданных. Хотя большинство доступных сегодня технологий обеспечивают, в основном, статические публикации, обеспечение большей интерактивности и средств совместной работы — дело ближайшего будущего.

Из важнейших характеристик систем управления информцией специалисты обычно выделяют следующие:

- программная платформа (система, обеспечивающая хранение и поиск документов, а также система обмена сообщениями. В настоящее время используется архитектура «клиент-сервер»);
- поддержка распределенной обработки информации;
- возможности масштабирования (набор поддерживаемых платформ; максимальное число пользователей; число уровней вложенности структур);
- открытость архитектуры и возможность интеграции с другими приложениями;
- типы документов, с которыми работает система (форматы документов; поддержка работы с составными документами и несколькими версиями документа; связи документов (один документ может быть ответом на другой или может быть порожден при исполнении предыдущего

документа); совместное использование электронных и бумажных документов);

- коллективная работа группы исполнителей над одним (или несколькими) документами;
- способ оповещения должностных лиц;
- особенности настройки продукта для нужд конкретного заказчика (например, регистрационная карточка должна содержать все необходимые реквизиты); наличие локализованного (русифицированного) интерфейса;
- средства регламентации доступа и криптозащиты;
- средства оповещения о нарушениях в регламенте прохождения документов;
- ориентация на традиционную российскую концепцию документооборота [1].

Системы управления характеризуются следующими функциями:

1) Ввод — система должна обеспечивать ввод, обработку, хранение, регистрацию данных.

2) Хранение — централизованное хранилище данных позволяет решить целый ряд проблем:

- возможность оперативного получения целостной и непротиворечивой информации;
- организация сбора, хранения и обработки информации;
- высокая надежность и управляемость системы, быстрый и подконтрольный доступ пользователей к информации;

- экономию средств за счет сосредоточения обслуживающего персонала в одном месте.

Обзор аналогов

Медицинская информационная система Медиалог

Данная система позволяет вести электронную медицинскую карту пациента, обеспечивает возможность обмена электронными медицинскими картами между разными учреждениями, содержит развитый и гибкий инструментарий для поддержки расписания приема врачей, диагностических кабинетов, мест групповых занятий. Медиалог обеспечивает учет медицинских услуг и взаиморасчеты с различными контрагентами медицинских учреждений – страховыми компаниями, предприятиями и пациентами. В системе предусмотрены механизмы для организация структурированного хранилища изображений, поиска, просмотра и редактирования изображений.

Из недостатков следует отметить слабое обеспечение безопасности и конфиденциальности данных, что в свою очередь является одним из ключевых требований к современной информационной системе. Система имеет неудобный пользовательский интерфейс, сложна в развертывании на рабочих местах и стоит при этом больших денег.

Медицинская информационная система AKSI - клиника

Данная информационная система обладает открытой сервисно-ориентированной архитектурой, быстро и легко масштабируются от одного рабочего места до сети медицинских учреждений ведомства, региона, федерального округа, страны. Система имеет удобный пользовательский интерфейс и эффективно автоматизирует деятельность поликлиник, стационаров, многопрофильных медцентров, частных клиник,

ведомственных медицинских учреждений, санаториев. AKSI – клиника имеет надежную систему безопасности и конфиденциальности данных.

Из недостатков следует отметить высокую стоимость продукта. Система позволяет хранить лишь структурированную текстовую информацию о пациенте. Невозможно изменить систему под какую – либо менее масштабную задачу.

Медицинская информационная система MedWork

Данная информационная система легка в администрировании и обучении за счет удобного пользовательского интерфейса. Систему удобно использовать в медпункте, крупной клиники, диагностическом центре, больнице из нескольких зданий. MedWork открыта для доработки и сопровождения как персоналом клиники, так и сторонними разработчиками.

Из недостатков следует отметить слабое обеспечение безопасности и конфиденциальности данных. Система не позволяет связать несколько независимых учреждений здравоохранения в единую сеть. При использовании системы возникали сбои, а также она долго реагировала на действия пользователя.

Медицинская информационная система «Санаторий» компании Медотрейд

Данная информационная система является интегрированной платформой с единой архитектурой, настраиваемым пользовательским интерфейсом и мощной поддержкой пользователей. Система имеет специально разработанную для небольших коммерческих санаторно-профилактических учреждений версию. Средство защиты информации обеспечивает надежную безопасность и конфиденциальность данных.

Из недостатков следует отметить плохую масштабируемость системы. Система не позволяет связать несколько независимых учреждений здравоохранения в единую сеть. Система позволяет хранить ограниченное количество графической информации о пациенте.

Результат сравнения продуктов

Приведенная ниже таблица составлена в результате анализа информации, представленной на официальных веб-сайтах медицинских информационных систем.

Параметр	**Медиалог**	**AKSI – клиника**	**MedWork**	**Санаторий**
Интуитивно понятный интерфейс	Нет	Да	Да	Да
Хранение графической информации о пациенте в электронной карте	Да	Нет	Да	Да
Легкость в развертывании и масштабируемости	Нет	Да	Нет	Нет
Поддержка системы (по 5 бальной шкале)	3	4	5	4
Оценка безопасности системы (по 5 бальной шкале)	3	5	4	3
Цена системы	Средняя	Высокая	Низкая	Низкая

Таблица 1 – Результаты сравнения медицинских информационных систем

Из результата сравнения становится ясно, что большинство аналогов довольно таки сложны в развертывании и не позволяют осуществить обмен информацией с другими системами. Так же видно, что чем выше цена

системы, тем надежнее в ней механизмы безопасности. Многие из систем не позволяют пользователю перестраивать их под свои определенные задачи.

ОБЗОР ТЕХНОЛОГИЙ WEB-ПРОГРАММИРОВАНИЯ

Язык программирования Ruby

Ruby (англ. *Ruby* — рубин) — динамический, рефлективный, интерпретируемый высокоуровневый язык программирования для быстрого и удобного объектно-ориентированного программирования. Язык обладает независимой от операционной системы реализацией многопоточности, строгой динамической типизацией, сборщиком мусора и многими другими возможностями. По особенностям синтаксиса он близок к языкам Perl и Eiffel, по объектно-ориентированному подходу — к Smalltalk. Также некоторые черты языка взяты из Python, Lisp [6].

К возможностям данного языка программирования можно отнести:

- Имеет лаконичный и простой синтаксис, частично разработанный под влиянием Ада, Eiffel и Python [12].
- Позволяет обрабатывать исключения в стиле Java, Python, C#.
- Позволяет переопределять операторы, которые на самом деле являются методами.
- Полностью объектно-ориентированный язык программирования. Все данные в Ruby являются объектами в понимании Smalltalk. Единственное исключение — управляющие конструкции, которые в Ruby, в отличие от Smalltalk, не являются объектами [12].
- Не поддерживает множественное наследование, но вместо него может использоваться концепция «примесей», основанная в данном языке на механизме модулей [12].
- Содержит автоматический сборщик мусора. Он работает для всех объектов Ruby, в том числе для внешних библиотек.
- Поддерживает замыкания с полной привязкой к переменным.

- Поддерживает блоки кода (код заключается в {…} или do … end).
- Целые переменные в Ruby автоматически конвертируются между типами Fixnum (32-разрядные) и Bignum (больше 32 разрядов) в зависимости от их значения, что позволяет производить целочисленные математические расчёты со сколь угодно большой точностью [8].
- Не требует предварительного объявления переменных. Язык использует простые соглашения для обозначения области видимости.
- В Ruby непосредственно в языке реализованы многие шаблоны проектирования.
- Может динамически загружать расширения, если это позволяет операционная система.
- Имеет независимую от ОС поддержку невытесняющей многопоточности.
- Перенесён на множество платформ.

Фреймворк Ruby on Rails

Для разработки был выбран фреймворк Ruby on Rails.

Ruby on Rails (RoR) — фреймворк, написанный на языке программирования Ruby, реализует архитектурный шаблон Model-View-Controller для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером баз данных. Является открытым программным обеспечением и распространяется под лицензией MIT [7].

Причины выбора:

1. Большая распространенность. Существует большое количество библиотек и готовых решений, позволяющих значительно ускорить процесс разработки. Также на stackoverflow.com можно найти ответ на практически любой вопрос, возникающий при разработке.

2. Основаннойсть на MVC. Данный шаблон проектирования позволяет разграничить объекты, отвечающие за визуальное отображение, обработку web запросов и реализацию модели данных.
3. Поддержка ORM. Каждая таблица в БД может автоматически быть преобразована в класс, а каждая запись в экземпляр класса. Данный функционал реализует стандартная библиотека ActiveRecord. Также все классы имеют встроенные функции работы с БД
4. Язык программирования Ruby. На данный момент язык считается наиболее объектно-ориентированным. Множество специфичных операторов и синтаксических конструкций позволяют сократить размер кода, а также вероятность появления ошибок в коде.

Среда разработки RubyMine

На сегодняшний день RubyMine самая распространенная IDE для разработки с использованием фреймворка ROR. На втором месте по популярности стоит связка текстовый редактор – консоль.

Причины выбора данной IDE:

1. Удобный пользовательский интерфейс;
2. Уникальные средства создания и оптимизации приложений;
3. Своевременные обновления, позволяющие IDE не отставать от быстрого развития фреймворка.

SQLite3

При разработке приложения использовалась БД SQLite3. SQLite3 – компактная встраиваемая база данных с открытым исходным кодом. Ruby on Rails работает с ней «из коробки», она быстродейственна и легко переносится с компьютера на компьютер. Так же фреймворк абстрагируется от используемой базы данных и не имеет значения какую БД ты используешь

при разработке, а какая используется в релизе. Данную базу данных было решено использовать также и в релизе.

Proxy-сервер

При размещении сайта не сервере было решено использовать два вида серверов: Backend и Frontend. Frontend-сервер получает web-запрос от сети и перенаправляет его на Backend-сервер, который в свою очередь обрабатывает его и возвращает ответ. В качестве Frontend-сервера был выбран Nginx. Отличительными особенностями которого являются легкая конфигурируемость, быстродействие и, в отличие от Apach, он не создает дополнительных нитей для каждого запроса. Вопроса в выборе Backend-сервера не стояло. Для целей создаваемого приложения подходил любой. Выбран был Puma, так как в отличие от остальных поддерживает web-сокеты «из коровки», незначительно быстрее остальных и имеет возможность распараллеливать запросы, а не выполнять поочередно (что важно только для высоконагруженных приложений).

ПРОЕКТИРОВАНИЕ КОМПОНЕНТОВ МЕДИЦИНСКОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Описание метода разработки

Проектирование системы осуществлялось методом нисходящего программирования. При этом методе проектирование осуществляется сверху вниз. Сначала пишется головной модуль, а нижележащие функции обработки данных заменяются заглушками. Далее отлаживается головной модуль и заглушки заменяются конкретными модулями для обработки данных. Этот процесс повторяется до тех пор, пока не будут отлажены модули самых низших уровней. Преимуществом такого подхода при написании программ является то, что процесс написания программы совпадает с процессом отладки. При разработке компилятора процесс компиляции был разбит на несколько подзадач, таких как обработка текста, дешифрация команд, генерация кода команд, обработка данных. Такая декомпозиция задачи на отдельные подзадачи облегчает процесс проектирования и соответствует идеологии проектирования методом нисходящего программирования [5].

Работу по созданию приложения, условно можно разбить на несколько этапов:

1. Предварительный этап;
2. Разработка архитектуры проекта;
3. Программирование;
4. Разработка дизайна приложения;
5. Тестирование;
6. Размещение;
7. Администрирование (поддержка).

Подготовительный этап

На предварительном этапе устанавливаются все необходимые программные средства и создается каркас приложения. Также подключаются все необходимые библиотеки [5].

Установить Ruby на компьютер можно разными способами, правильным считается способ, использующий RVM (Ruby Version Manager). Данная программа позволяет использовать одновременно разные версии языка программирования, легко переключаться между ними, а также загружать другие версии. Для установки менеджера версий необходимо выполнить команду:

```
\curl -sSL https://get.rvm.io | bash -s stable
```

Ruby использует несколько библиотек и их необходимо установить предварительно:

```
sudo apt-get install git-core curl zlib1g-dev build-
essential libssl-dev libreadline-dev libyaml-dev libsqlite3-dev
sqlite3 libxml2-dev libxslt1-dev libcurl4-openssl-dev python-
software-properties libffi-dev
```

Затем нужно установить сам язык Ruby с использованием команды, где цифра обозначает версию языка:

```
rvm install 2.3.0
rvm use 2.3.0 --default
ruby -v
```

После установки языка программирования необходимо установить Bundler, являющийся незаменимой частью Ruby on Rails.

```
gem install bundler
```

Ruby on Rails является библиотекой (gem), написанной на языке Ruby и устанавливается также, как и любая библиотека:

```
gem install rails -v 5.0.1
```

Приложение можно создать как при помощи мастера, так и из консоли. Для создания приложения необходимо выполнить команду «rails new», а затем создать базу данных командой «rails db:create»

Затем необходимо подключить необходимые библиотеки. Подключение библиотек в Rails-приложениях просто. Для этого необходимо прописать название библиотеки в Gemfile и вызвать команду «bundle install».

Список необходимых для данного приложения библиотек:

```
source 'https://rubygems.org'

git_source(:github) do |repo_name|
  repo_name = "#{repo_name}/#{repo_name}" unless
repo_name.include?("/")
  "https://github.com/#{repo_name}.git"
end

gem 'rails', '~> 5.0.1'
gem 'sqlite3'
gem 'puma', '~> 3.0'
gem 'sass-rails', '~> 5.0'
gem 'uglifier', '>= 1.3.0'
gem 'coffee-rails', '~> 4.2'
gem 'jquery-rails'
gem 'turbolinks', '~> 5'
gem 'jbuilder', '~> 2.5'

gem 'rolify'
gem 'cancancan'

gem 'devise'
gem 'activeadmin', github: 'activeadmin/activeadmin'
gem 'inherited_resources', github:
'activeadmin/inherited_resources'
gem 'activeadmin_addons'

group :development, :test do
  gem 'byebug', platform: :mri
end

group :development do
  gem 'better_errors'
  gem 'web-console', '>= 3.3.0'
```

```
    gem 'listen', '~> 3.0.5'
    gem 'spring'
    gem 'spring-watcher-listen', '~> 2.0.0'
  end

  gem 'tzinfo-data', platforms: [ :mingw, :mswin, :x64_mingw,
:jruby]

  # позволяет работать с изображениями
  gem 'carrierwave'
  # организует хранилище для фотографий
  gem 'fog'
  gem 'mini_magick'

  gem 'russian'

  gem 'ckeditor', github: 'galetahub/ckeditor'
```

Проектирование механизма аутентификации

Аутентификацией в разрабатываемом приложении будет заниматься библиотека «devise». В настоящий момент она практически стандартной для фреймворка Ruby on Rails и используется практически во всех приложения. Подключается библиотека через Gemfile, как было указано в предыдущем пункте.

Для установки библиотеки необходимо выполнить команду:

```
rails g active_admin:install User
```

Команда создаст конфигурационные файлы, миграцию, создающую таблицу, а также произведет настройки, для использования библиотеки «из коробки». Пользователем является любой сотрудник профилактического заведения, так как в фреймворке принято называть таблицу для аутентификации пользователи, назовем ее также. Второй причиной данного названия послужило то, что в дальнейшем будут добавлены роли и будет присутствовать одноименная роль сотрудник, что вызовет путаницу.

```
class DeviseCreateUsers < ActiveRecord::Migration[ 5.0]
  def change
    create_table :users do |t|
```

```
      ## Database authenticatable
      t.string :email,              null: false, default: ""
      t.string :encrypted_password, null: false, default: ""
      ## Recoverable
      t.string   :reset_password_token
      t.datetime :reset_password_sent_at
      ## Rememberable
      t.datetime :remember_created_at
      ## Trackable
      t.integer  :sign_in_count, default: 0, null: false
      t.datetime :current_sign_in_at
      t.datetime :last_sign_in_at
      t.string   :current_sign_in_ip
      t.string   :last_sign_in_ip

      t.string  :avatar
      t.string  :name
      t.timestamps null: false
    end
    add_index :users, :email,                unique: true
    add_index :users, :reset_password_token, unique: true
  end
end
```

Библиотека автоматически сгенерирует класс модели для созданной таблицы, а также пропишет в роутер все пути, необходимые для реализации механизма аутентификации.

В конфигурационном файле, создаваемом библиотекой, помимо закомментированных команд, демонстрирующих возможности библиотеки и способы ее настройки, указаны настройки аутентификации. Указан адрес электронной почты, от имени которой будет производится отправка сообщений (даже без изменения письма будут приходить). Указана длинна пароля и параметры токена аутентификации.

```
Devise.setup do |config|
  config.mailer_sender = 'please-change-me-at-config-
initializers-devise@example.com'
  config.case_insensitive_keys = [ :email]
  config.strip_whitespace_keys = [ :email]
  config.skip_session_storage = [ :http_auth]
  config.stretches = Rails.env.test? ? 1 : 11
```

```
  config.reconfirmable = true
  config.expire_all_remember_me_on_sign_out = truee
  config.password_length = 6..128
  config.email_regexp = /\A[^@\s]+@[^@\s]+\z/
  config.reset_password_within = 6.hours
  config.sign_out_via = :delete
end
```

Проектирование системы управления содержимым

Разрабатываемая система будет построена поверх административной панели ActiveAdmin. Данная система, совместно с «devise», реализует высокий уровень защиты данных, а также удобство использования и легкость конфигурирования под требования, предъявляемые к разрабатываемой системе. Также в библиотеке уже имеется «сносный» дизайн, реализован механизм поиска и фильтрации, требуемый по техническому заданию.

Для установки административной панели необходимо выполнить команду:

```
rails g active_admin:install User
```

Команда принимает параметром имя класса, для которого реализована аутентификация посредством библиотеки «devise».

Библиотека создаст в корне папку «admin», в которой будут лежать файлы контроллеров, которые в дальнейшем необходимо будет реализовать.

Также библиотека создает конфигурационный файл:

```
ActiveAdmin.setup do |config|
  config.site_title = "Hospital"
  config.authentication_method = :authenticate_user!
  config.current_user_method = :current_user
  config.logout_link_path = :destroy_user_session_path
  config.batch_actions = true
  config.localize_format = :long
  config.root_to = 'patients#index'
  config.register_javascript 'ckeditor/init.js'

end
```

Проектирование системы разграничения доступа

Для реализации системы разграничения доступа было решено использовать в тандеме две библиотеки: «CanCanCan» и «Rolify». Первая библиотека предоставляет возможность лаконичной проверки прав пользователя и с аналогичной простотой задавать ему права. Вторая библиотека позволяет создавать роли для пользователей и легко ассоциировать пользователя с ними. Что немаловажно, пользователь имеет возможность одновременно относиться к нескольким ролям. Примером, для нашей предметной области будет то, что главный врач одновременно и врач, и управляющий медицинским учреждением.

Для установки библиотеки необходимо выполнить команду:

```
rails g cancan:ability
```

Во время выполнения команды, генератор создаст миграцию, генерирующую таблицу, с полиморфной привязкой с модели, которая имеет определенную роль. Например, врач может быть пациентом, а пациент врачом. В разрабатываемой версии системы возможность объединения моделей не предусмотрена, но данный прием при проектировании позволит легко расширить систему. Ниже представлен сгенерированный файл миграции:

```
class RolifyCreateRoles < ActiveRecord::Migration
  def change
    create_table(:roles) do |t|
      t.string :name
      t.references :resource, :polymorphic => true

      t.timestamps
    end

    create_table(:users_roles, :id => false) do |t|
      t.references :user
      t.references :role
    end
```

```
        add_index(:roles, :name)
        add_index(:roles, [  :name, :resource_type, :resource_id
])
        add_index(:users_roles, [  :user_id, :role_id ])
      end
    end
```

Таблица отношения пользователей и ролей не будет иметь собственной модели, она создана для того, чтобы реализовать отношение многие ко многим между ролями и пользователями. Дополнительно необходимо указать в настройках модели, что к ней подключена библиотека.

Библиотека «CanCanCan» создает модель «Ability», в которой необходимо указать все возможности каждой роли.

```
    class Ability
      include CanCan::Ability

      def initialize(user)
        user ||= User.new # guest user (not logged in)
        if user.has_role? :chief_medical_officer#управление
может все
          can :manage, :all
        else
          #так как нет прямой иерархии в ролях, роли дополняют
друг друга и нужно проверить все
          if user.has_role? :doctor
            can :read, :all
            can :manage, Patient
            can :manage, Room
            can :manage, Equipment
          end
          if user.has_role? :register_office
            can :read, :all
            can :manage, Patient
          end
          if user.has_role? :employee
            can :read, :all
          end
        end
      end
    end
```

В данном коде фигурируют четыре роли: главный врач, доктор, работник регистратуры и наемный работник. Главный врач имеет доступ ко всему; врач имеет доступ к пациентам, кабинетам и оборудованию на изменение, остальные модели доступны только для чтения; сотрудник регистратуры имеет доступ на изменение к пациентам, остальное на чтение; рядовой сотрудник имеет доступ только на чтение ко всем моделям.

На момент первоначального запуска приложения не существует ни ролей, ни пользователей, способных их создать. Для решения данной проблемы в фреймворке существует возможность «посадки» данных, вызовом команды:

```
rails db:seed
```

Данная команда запускает скрипт, который занимается «популяцией» базы данных, его содержимое:

```
    Role.find_or_create_by name: :doctor                # Врач
    Role.find_or_create_by name: :chief_medical_officer #
Главный врач и все управление
    Role.find_or_create_by name: :register_office       #
Работник регистратуры
    Role.find_or_create_by name: :employee              #
Рядовой сотрудник, которому зачем-то создали аккаунт.
                                                        # Имеет
доступ только на чтение. Роль по умолчанию

    User.transaction do
      password = '123456'#первоначальный пароль, который
необходимо заменить после первого входа

      u = User.create!(email: 'doctor1@example.com', name: 'Иван
Смирнов', password: password, password_confirmation: password)
      u.add_role :doctor
      u.save
      u = User.create!(email: 'doctor2@example.com', name:
'Витор Клочков', password: password, password_confirmation:
password)
      u.add_role :doctor
      u.save
      u = User.create!(email: 'doctor3@example.com', name:
'Ринат Хабибулин', password: password, password_confirmation:
password)
```

```
      u.add_role :doctor
      u.save

      u = User.create!(email: 'register@example.com', name:
'Нина Грушина', password: password, password_confirmation:
password)
      u.add_role :register_office
      u.save
    end
    puts 'Success'
```

На данном этапе модель «User» имеет окончательный вариант. Для реализации проверки доступа, модель делегирует методы «can?» и «cannot?» модели «Ability», которая на основании конфигурации, представленной выше, определяет возможности каждого пользователя.

```
    class User < ApplicationRecord
      mount_uploader :avatar, AvatarUploader
      rolify
      devise :database_authenticatable,
             :recoverable, :rememberable, :trackable,
:validatable

      belongs_to :role
      delegate :can?, :cannot?, to: :ability
      after_create :assign_default_role
      validates_uniqueness_of :email

      def assign_default_role
        add_role :employee
      end
    end
```

После создания пользователя, ему назначается роль по умолчанию – сотрудник. Также в модели указано, что библиотека «devise» может аутентифицировать пользователя по информации из базы данных, восстанавливать пароль, если он был забыт, сохранять аутентификационные данные в куках браузера, вести статистику использования сервиса пользователем, а также проводить валидацию данных, введенных пользователем.

Для увеличения удобства использования разрабатаевомого сервиса, решено было каждому объекту бизнесс логике реализовать собственные рассписание (рассписание врачей, использование оборудоваиния, кабинета т.д) При планировании графика работы врача или времени использования оборудование необходимо в ручном самостоятельно проверить непересекаемость временных промежутков. Для увеличения удобства использования интерфейс работы с расписанием было решено выполнить в стиле текстового процессора «Word», так как сотрудники профилактического учреждения уже умеют навык использования данного программного продукта.

В качестве редактора расписания было решено использовать библиотеку «ckeditor», получившую широкое распространение в среде web-разработчиков. Помимо редактирования текста, система позволяет загружать документы, работать с таблицами.

Для установки редактора необходимо выполнить команду:

```
rails generate ckeditor:install --orm=active_record --backend=carrierwave
```

Параметрами команды является тип библиотеки для реализации объектно-реляционного отображения и библиотека для работы с изображениями. Редактор выглядит как представлено на рисунке ниже.

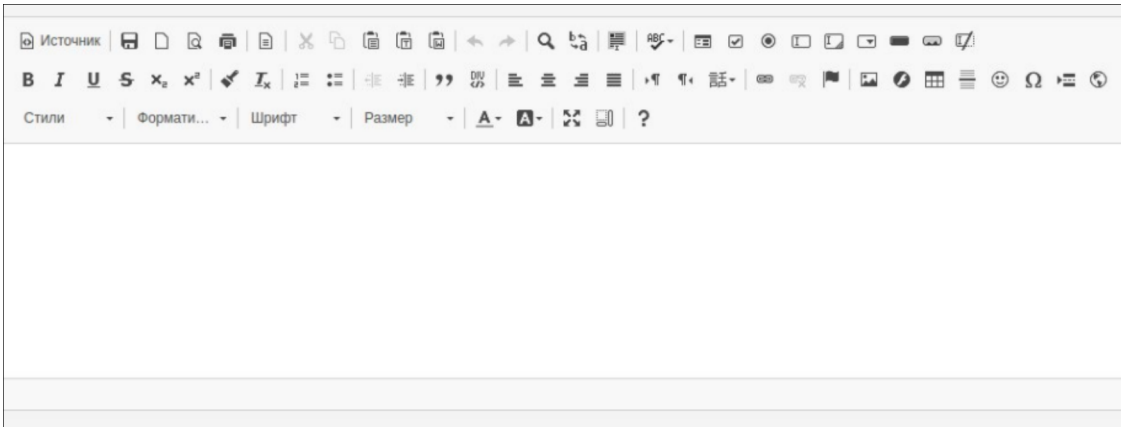


Рисунок 1 – Редактор расписания

Необходимо ограничить функционал редактора, доступный пользователю. Для этого необходимо написать скрипт на языке JavaScript с указанием необходимого функционала.

```
CKEDITOR.editorConfig = function( config ) {
    config.toolbar_Custom = [
        { name: 'clipboard',   items : [ 'Undo','Redo' ] },
        { name: 'insert',      items : [ 'Table' ] },
        { name: 'tools',       items : [ 'Maximize' ] },
        { name: 'paragraph',   items : [
'NumberedList','BulletedList','-','Outdent','Indent','-
','Blockquote','-
','JustifyLeft','JustifyCenter','JustifyRight','JustifyBlock' ]
},
        '/',
        { name: 'styles',      items : [
'Format','Font','FontSize' ] },
        { name: 'text',        items :
[ "Bold","Italic","Underline","Strike","Subscript","Superscript",
"RemoveFormat"]},
        { name: 'colors',      items : [
'TextColor','BGColor' ] }
    ];

    config.language = 'ru';
    config.toolbar = 'Max';
};
```

ПРОЕКТИРОВАНИЕ БД

Согласно техническому заданию, необходимо реализовать модели: пациентов, оборудования и процедурных кабинетов. Реализация моделей должна упрощать составление расписания работы и использования, автоматизировать учет, позволять безопасно хранить данные.

Представленный ниже код, создает таблицу с пациентами. Помимо основной информации, данная таблица имеет поле с информацией о расписании лечения (в дальнейшем будет подключен функциональный редактор для работы с ним), поле, в которое можно разместить амбулаторную книгу, а также ссылку на лечащего врача. При реализации модели для данной таблицы будет наложено ограничение, что только пользователь с ролью врач, может быть лечащем врачом пациента.

```
class CreatePatients < ActiveRecord::Migration[ 5.0]
  def change
    create_table :patients do |t|
      t.string :name, null: false
      t.date    :birthday, null: false
      t.boolean :male, null: false, default: true#false
female
      t.date :departure, null: false#дата выселения
      t.timestamps
      t.string  :mobile_phone
      t.string :description# любая информация, включая
истории болезни
      t.string :schedule# расписание лечения
      t.references :user#лечащий врач
    end
  end
end
```

Далее создается таблица с информацией об оборудовании. Помимо названия, инвентарного номера и описания существует ссылка на ответственного за него. При реализации модели для данной таблицы не будет наложено ограничение, что только пользователь с ролью сотрудник, может

быть записан в это поле. Так как все пользователи по умолчанию имеют данную роль.

Далее создается таблица, хранящая в себе информацию о медицинских кабинетах. Таблица аналогична таблице с информацией об оборудовании.

К каждой созданной таблице необходимо создать модель, установить ограничения по данным, отношения между моделями.

В разработке данной квалификационной работы используется принцип объектно-реляционного отображения. Он подразумевает соотношение класса с таблицей в базе данных, где запись — это экземпляр класса, а поле — это свойство класса. Для реализации используется стандартная библиотека фреймворка Ruby on Rails – ActveRecord. Данная библиотека инкапсулирует SQL-запросы и предоставляет их в форме методов и свойств класса. Библиотека имеет встроенные механизмы валидации данных и создания реляционных отношений между классами [10].

Так как система разрабатывается на высокоуровневом фреймворке, все модели имеют однотипную структуру, представим наиболее интересную модель – пациентов.

```
class Patient < ApplicationRecord
  belongs_to :user
  validates_presence_of :user_id, :name, :birthday, :male,
:departure

  validate :validate_doctor

  private
  def validate_doctor

    errors.add(:user, ' только врач может быть привязан к
пациенту')  if !user.blank? && !user.has_role?(:doctor)
  end
end
```

В модель добавлена валидация данных, дублирующая ограничения, созданные в базе данных. Дополнительно добавлена валидация пользователя, которому принадлежит пациент – им может быть только врач.

ПРОЕКТИРОВАНИЕ КОНТРОЛЛЕРОВ

Дальше необходимо создать контроллеры для моделей. Котроллеры описывают все действия, которые можно произвести с моделью. В Ruby on Rails используется модель CRUD.

Так как мы используем для реализации приложения библиотеку «ActiveAdmin», реализовывать необходимо не стандартные контроллеры, а контроллеры данной библиотеки. Они являются настойной над стандартными с готовыми файлами представления и механизмом фильтрации, которая является основным из главных требований в техническом задании.

Все файлы конфигураций состоят из трех необязательных блоков: конфигурирования списка, конфигурирование показа и конфигурирование редактирования. Так же имеются дополнительные блоки, один из которых отвечает за поиск объектов в базе данных, описание использования которого будет описано далее.

Первым созданным контроллером, был контроллер пользователей. Данный котроллер позволяет добавлять сотрудников, назначать им роли, редактировать информацию, а также просматривать списком с возможностью фильтрации. Реализация данного класса представлена в приложении Б.2.

Для реализации модели мандатного доступа ограничено количество действий, которые пользователь может произвести над записями данной модели. Управлять пользователями может только главный врач, остальные роли могут только просматривать список сотрудников и информацию о них. Для удобства управления сотрудниками было решено создать по вкладке для списка пользователей, принадлежащих каждой роли.

Остальные контроллеры полностью аналогичные данному. Различием является указание свойств модели, для которой специфичен контроллер. Разграничение доступа реализовано на уровне библиотеки, поэтому в других

контроллерах также вызывается метод «can?», только с указанием модели, для которой проверяется возможность доступа.

Обработка URL

Для обработки различных URL и перенаправления их методам контроллера в ROR имеется файл «Routes».

Маршрутами приложения управляют библиотеки «ActiveAdmin» и «devise», поэтому системе необходимо это указать. Также необходимо главную страницу перенаправить на страницу авторизации.

```
Rails.application.routes.draw do
  mount Ckeditor::Engine => '/ckeditor'
  devise_for :users, ActiveAdmin::Devise.config
  ActiveAdmin.routes(self)
  root to: 'admin/dashboard#index'
end
```

Организация поиска

Как было сказано выше, поиск в библиотеке «ActiveAdmin» работает из «коробки». Существует возможность ограничения полей, по которым будет происходить поиск, что было сделано для модели User. Были убраны из поиска служебные поля, а также хеш пароля (искать по паролю человека достаточно нелогично).

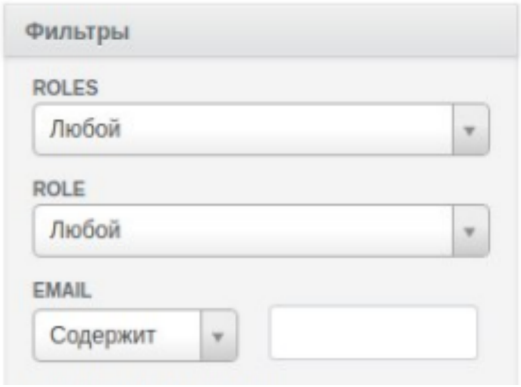


Рисунок 2 – Форма поиска

Просмотр списка записей

На рисунке представлено отображение списка сотрудников. В файле контроллера указывается список параметров, которые должны быть отображены в списке.

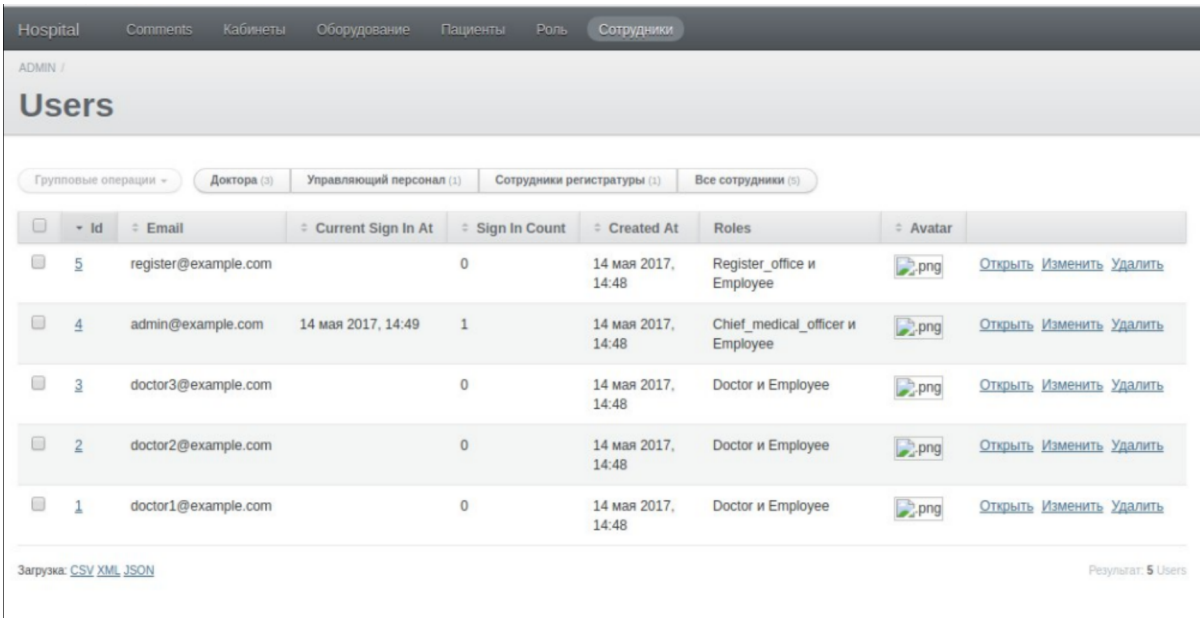


Рисунок 3 – Список сотрудников

Для остальных моделей отображение списка записей аналогично. Единственное отличие в выводимых полях. В сессии, в которой была сделана фотография, был пользователь с ролью главный врач, поэтому ему доступны ссылки на изменение, удаление, а также создание нового пользователя.

В парадигме CRUD за отображение списка отвечает метод Index, для изменения содержимого в контроллере необходимо перегрузить данный метод и указать параметры, необходимые для вывода на экран. Ниже представлена реализация для модели кабинета. На экран выводится идентификатор записи, номер кабинета и информация об ответственном за него.

```
index do
  selectable_column
  id_column
  column :code
  column 'Ответственный', :user
  actions
```

```
end
```

Просмотр подробностей записи

В библиотеке «ActiveAdmin» для изменения формы создания новой записи необходимо перегрузить метод «show». Реализация для модели пациент представлена ниже.

```
show do
  attributes_table do
    row :id
    row :name, label: 'Имя'
    row :birthday, label: 'Дата рождения'
    row :male, label: 'мужчина'
    row :departure, label: 'Дата выписки'
    row :mobile_phone, label: 'Номр телефона'
    row :created_at, label: 'Номр телефона'
    row :user, label: 'Лечащий врач'
    row 'Описание(амбулаторная книга)' do |user|
      user.description.html_safe
    end
    row 'Расписание лечения' do |user|
      user.schedule.html_safe
    end
  end
end
```

В представленном выше исходном коде при выводе расписания лечения происходит вывод не текста, а html станицы, которую данный текст

описывает. Страница, описываемая данным кодом представлена ниже.

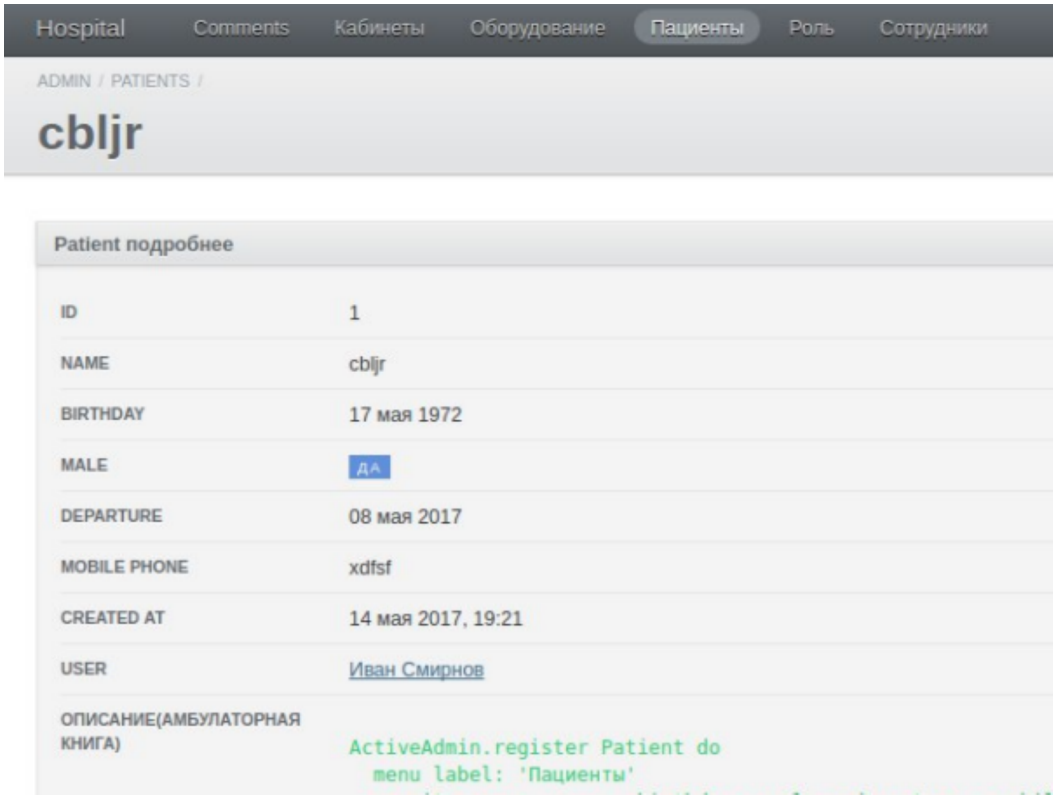


Рисунок 4 – Информация о пациенте

Данная запись была создана автоматически со случайным текстом во время тестирования системы.

Дизайн приложения

В разрабатываемом сервисе не создавалось файлов разметки и каскадных таблиц стилей. Все страницы автоматически сгенерированы библиотекой «ActiveAdmin». В стандартном виде сгенерированные формы имеют не привлекательный вид. Для того чтобы сделать приложение соответствующим стандартам современной разработки, было решено использовать расширение «ActiveAdmin_Addoins». Для установки необходимо выполнить команду, представленную ниже. Команда подключит стили и скрипты для изменения внешнего вида сайта и увеличения удобства использования.

```
rails g activeadmin_addons:install
```

Данная команды подключит стили и скрипты к системе.

ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Объект испытаний

Объектом испытания является система, которая представлена WEB-сервисом, предназначенный для автоматизации учета и безопасного хранения данных. Система написана на языке Ruby с использованием фреймворка Ruby on Rails.

Цель испытаний

Испытания проводятся с целью проверки работоспособности и надежности системы автоматизации работы медицинского учреждения, а также правильности выполнения всех функций, указанных в техническом задании.

Требования к программе

Система должна быть представлена WEB-сервисом, в котором сотрудники регистратуры санатория имеют возможность добавления, изменения и удаления информации о пациентах. Должен быть реализован удобный поиск информации. В системе должна быть реализована возможность оформления расписания лечения для пациентов, а также учет его исполнения. Система должна помогать в планировании работы врачей, кабинетов и медицинского оборудования.

В приложении должно быть предусмотрено разграничение доступа. Работник регистратуры должен иметь доступ только с записи и учету пациентов. Врачи должны иметь доступ к расписанию лечения пациентов и расписанию работы кабинетов и медицинского оборудования на

редактирование, к другим разделам только на чтение. Руководство санатория должен иметь доступ ко всем разделам системы и меть возможность изменять любые данные. Доступ к любому разделу системы возможен только после аутентификации.

Методы испытания

Приложение тестировалось двумя различными методами: поведенчески и при помощи Unit-тестов.

Поведенческое тестирование проходило в ручном режиме. В ходе него было выявлено множество ошибок, которые впоследствии были исправлены. В данном виде тестирования моделируется поведение пользователя.

При разработке частично использовался метод TDD (Test Drive Development), согласно которому сначала пишется тест, покрывающий желаемые изменения, затем пишется код, позволяющий пройти тест, затем проводится рефакторинг написанного кода и цикл повторяется.

Для тестирования системы были созданы профиль пользователей всех основных ролей.

```
User.transaction do
  password = '123456'
  u = User.create!(email: 'doctor1@example.com', name: 'Иван
Смирнов', password: password, password_confirmation: password)
  u.add_role :doctor
  u.save
  u = User.create!(email: 'doctor2@example.com', name:
'Витор Клочков', password: password, password_confirmation:
password)
  u.add_role :doctor
  u.save
  u = User.create!(email: 'doctor3@example.com', name:
'Ринат Хабибулин', password: password, password_confirmation:
password)
  u.add_role :doctor
  u.save
```

```
      u = User.create!(email: 'admin@example.com', name: 'Алла
Ходырева', password: password, password_confirmation: password)
      u.add_role :chief_medical_officer
      u.save

      u = User.create!(email: 'register@example.com', name:
'Нина Грушина', password: password, password_confirmation:
password)
      u.add_role :register_office
      u.save
    end
```

Для удобства тестирования пароль у всех пользователей одинаковый. Был произведен вход каждым из пользователей и была проверена возможность совершения различных действий, запрещенных для каждой роли. Так как на сайте нет кнопок для выполнения запрещенных действий, запросы выполнялись из консоли разработчика браузера Google Chrome. На все запрещенные действия сервер возвращал ошибку 403, что означает корректность работы системы разграничения доступа.

Были созданы объекты всех моделей, произведены действия над ними. Система вела себя корректно.

Для тестирования системы поиска были созданы объекты с заведомо известными значениями и производился поиск по ним. Во всех случаях система вела себя корректно.

Создание записи

В библиотеке «ActiveAdmin» для изменения формы создания новой записи необходимо перегрузить метод «form». Реализация для модели пациент представлена ниже.

```
      form do |f|
        f.inputs 'Пациент' do
          f.input :user, as: :select, include_blank: false,
                  label: 'Лечащий врач', collection:
User.all.joins(:roles)
                      .where('roles.name = ?', :doctor).map{ |u| [
u.name, u.id ]}
          f.input :name, label: 'Имя'
```

```
            f.input :birthday, as: :date_time_picker, label: 'Дата
рождения'
            f.input :male, label: 'мужчина'
            f.input :departure, as: :date_time_picker, label:
'Дата выписки'
            f.input :mobile_phone, label: 'Номр телефона'
            f.input :description, as: :ckeditor, label:
'Описание(амбулаторная книга)'
            f.input :schedule, as: :ckeditor, label: 'Расписание
лечения'
          end
          f.actions
        end
```

Для выбора лечащего врача происходит выборка только пользователей, которые имеют роль «врач». Также для ввода амбулаторной карты и создания расписания лечения используется расширенный редактор «ckeditor».

Созданная данным скриптом форма представлена ниже.

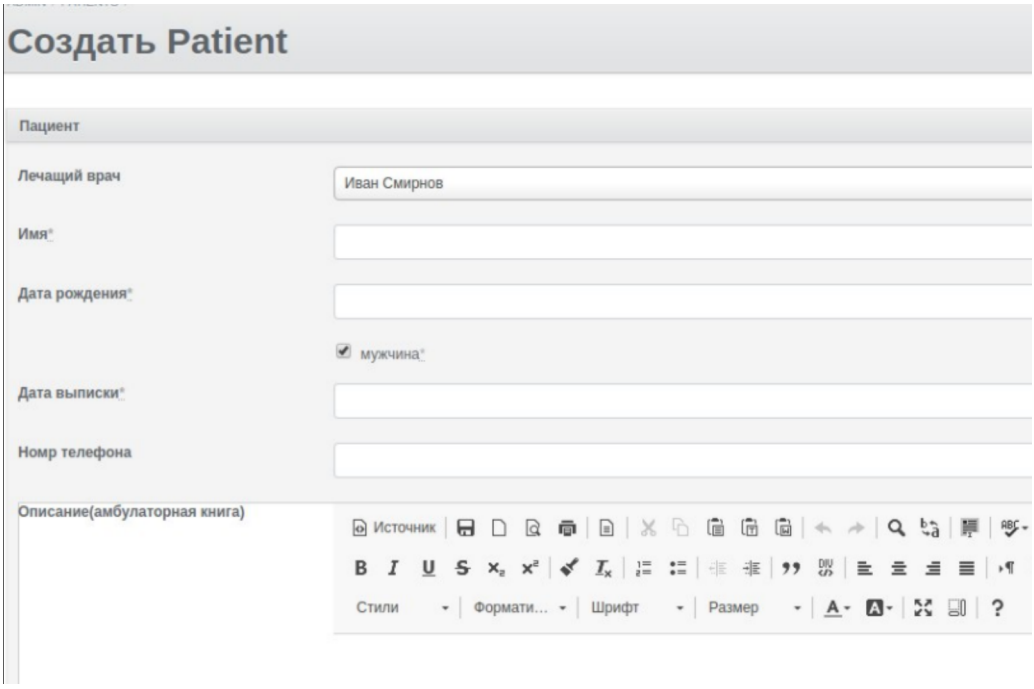


Рисунок 5 – Создание записи о пациенте

Пользовательский интерфейс максимально простой и удобный. В случае ввода не валидных данных, поля будут выделены красным и снизу появится подпись с информацией почему данное значение не подходит.

Для каждой записи в системе можно оставить комментарий или служебную заметку, либо вести дискуссию. Это удобно при обсуждении врачами результатов их работы и прогресса, наблюдаемого в лечении пациента.

Ошибки и способы их устранения

Ошибка 404 обозначает, что страница не найдена на сервере. Чаще всего это случается из-за того, что у пользователя неправильная ссылка на ресурс. Или ко времени перехода по ссылке ресурс был удален или перемещен. В случае возникновения данной ошибки, необходимо проверить правильность написания ссылки, если это не помогло, то необходимо обратиться в службу поддержки.

Ошибка 403 означает, что у пользователя не хватает прав для доступа к данному ресурсу. Для решения данной проблемы необходимо авторизоваться, если даже после этого у пользователя происходит такая ошибка, то ему действительно не хватает прав для доступа к данному ресурсу.

Ошибка 502 означает, что Nginx не смог обратиться к приложению. Возникает из-за того, что процесс приложения остановлен или процесс не отвечает на запросы. Для решения данной проблемы необходимо повторно запустить приложение на сервере, либо проверить настройки Nginx, в частности, расположение файла с номером процесса приложения.

Ошибка 500 обозначает ошибку на сервере. Данная ошибка самая распространенная и может возникать по различным причинам. В большинстве случаев она говорит о том, что программист сделал ошибку в приложении.

Подробное описание причины возникновения ошибки, а также файл, метод и даже строчку можно найти в файле протокола на сервере. Для анализа работы сервера необходимо просматривать файл протокола. В нем

можно найти все происходящие ошибки, все запросы, приходящие к приложению. Данная ошибка требует вмешательства разработчика для устранения. Если проводить параллели с языками программирования, данная ошибка означает необработанное исключение времени исполнения.

ФУНКЦИОНИРОВАНИЕ ПРОГРАММЫ

Диаграмма использования

Диаграмма вариантов использования (ДВИ) – это диаграмма, которая позволяет описать функциональность и поведение системы. Она отражает отношения между действующим лицом (актером) и вариантами использования (прецедентами). ДВИ позволяет сформулировать требования к системе и определить ее функциональность. ДВИ отражает концептуальную модель системы.

На рисунке 6 представлена диаграмма использования. На диаграмме представлено четыре субъекта: главный врач, работник регистратуры, сотрудник и врач. Субъекты являются основными ролями в приложении. В качестве вариантов использования на диаграмме указаны основные пункты технического задания. Главный врач имеет полный доступ в системе. Работник регистратуры имеет право только просматривать информацию на сайте. Врач не имеет доступ только к управлению сотрудниками. Рядовой сотрудник имеет доступ только к просмотру информации.

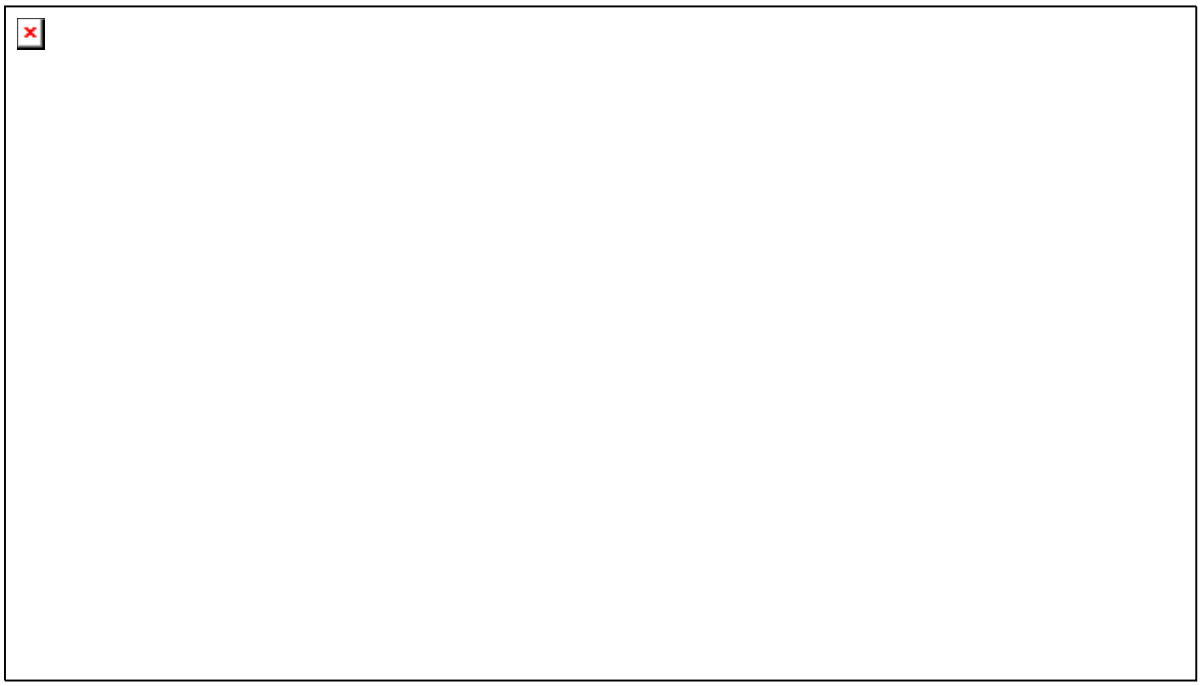


Рисунок 6 – Диаграмма использования

Описание схемы работы системы

На рисунке 7 представлена схема работы системы. Пользователь отправляет запросы к приложению через сеть интернет, прокси-сервер Nginx получает их и перенаправляет приложению. Приложение в свою очередь получает данные из базы данных, представляет их в нужном виде и передает HTML-страницы обратно через сеть интернет.

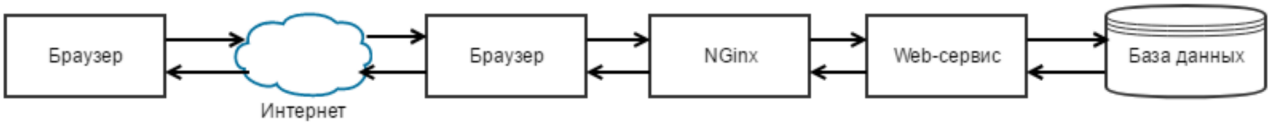


Рисунок 7 – Схема работы системы

Данная схема является стандартной для web-приложений. Необязательным элементом на данной схеме является Nginx. Система без него работоспособна, но будет ряд ограничений, самым важным их которых будет невозможность разместить на одном IP-адресе нескольких web-приложений. Если рассматривать схему без привязки к конкретному приложению, схема превращается в дерево, с многими приложениями,

связанными с Nginx. Во избежание монопольного использования ресурсов сервера была использована данная схема.

Диаграмма классов

Диаграмма классов служит для отражения статической структуры модели системы и представляет собой логическую модель системы. Она позволяет определить какие объекты, основные методы и функции необходимы для нормального функционирования проектируемой системы.

Диаграмма классов состоит из различных сущностей и отношений между ними. В роли сущности могут выступать различные классы и интерфейсы. Класс служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношением с объектами других классов. В роли отношений между классами могут выступать следующие виды связей: зависимость, ассоциация, обобщение и реализация.

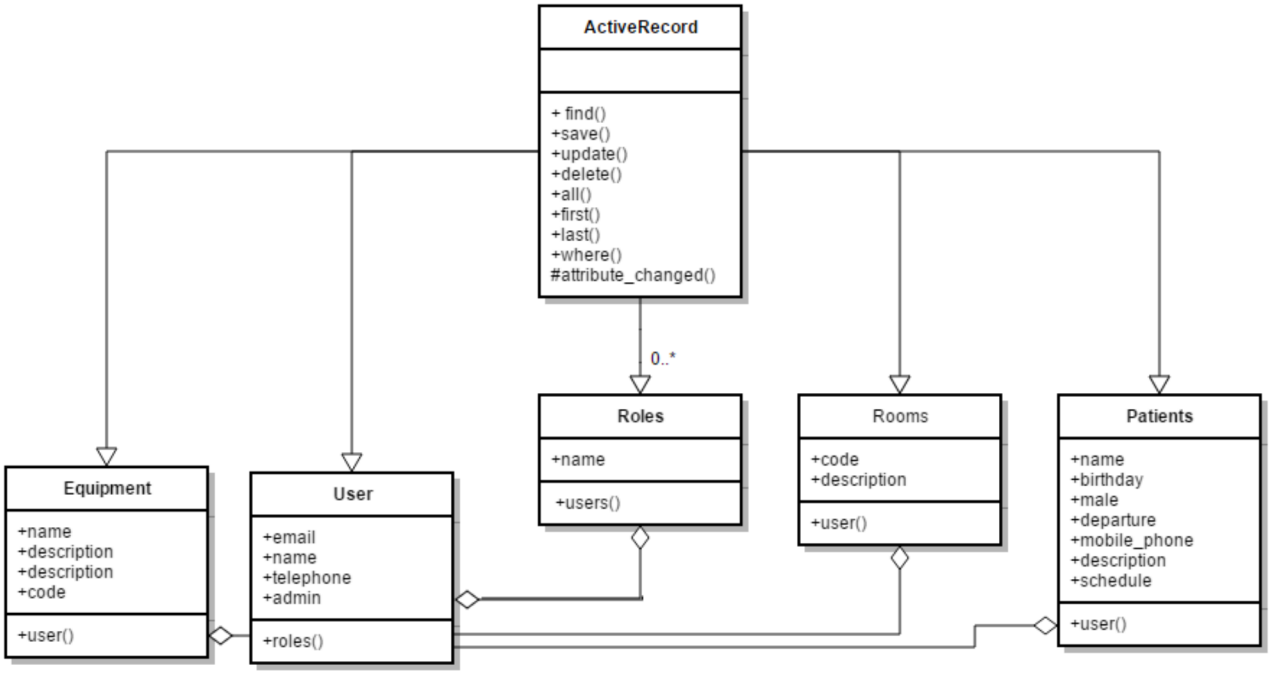


Рисунок 8 – Диаграмма классов

Все классы про наследованы от базового «ActiveRecord». Базовый класс реализует методы доступа, поиска, изменения, создания и удаления

записей. Классы наследники имеют одноименные с полями в таблице атрибуты. Также классы имеют методы отношений, возвращающих ссылающиеся объекты или объект, на который ссылается данный. Каждый класс может быть расширен своим уникальным поведением, но в данной системе этого не требовалось.

Логическая модель данных

В приложении А представлена логическая схема базы данных. Особенностью данной базы данных является то, что таблица «active_admin_comments» полиморфна, тесть может относиться к любой другой таблице, так как в разработанной системе можно отставать заметки о любом объекте. Данная возможность реализуется двумя полями: внешним ключом и именем таблицы, на запись в которой ссылается объект.

Модели роли и пользователи имеют связь многие ко многим. Для реализации отношения системой была автоматически создана третья таблица. Название у вспомогательной таблицы формируется из имен связуемых таблиц в алфавитном порядке.

Логическая схема данных представлена в приложении А.

Физическая модель данных

На рисунке 9 представлена физическая модель данных применительно к базы данных SQLite, но привязки к базе данных нет, существует возможность подключиться к любой, для которой написан адаптер.

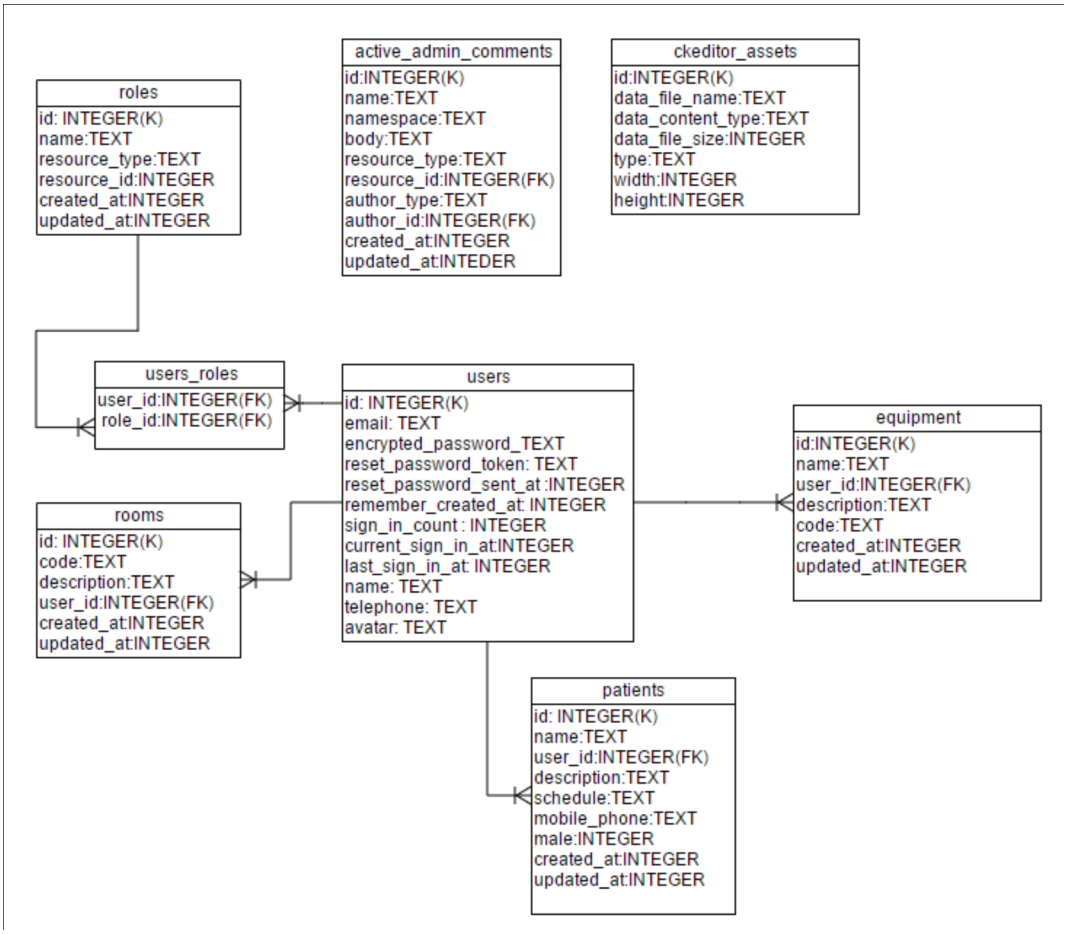


Рисунок 9 – Физическая модель данных

Все отношения находится в третьей нормальной форме (3НФ), так как: все атрибуты каждого отношения атомарны; каждый не ключевой атрибут отношения неприводимо (функционально полно) зависит от его потенциального ключа; отсутствуют транзитивные функциональные зависимости не ключевых атрибутов от ключевых [11].

Скрипт создающий базу данных, представлен в приложении Б5.

Для разграничения между базами данных различного окружения создается файл, в котором указываются авторизационные данные для подключения к базе данных. Так как SQLite является встраиваемой системой, не поддерживающей аутентификацию и разграничение доступа, его содержание осталось стандартным.

ЗАКЛЮЧЕНИЕ

Задачей дипломного проектирования являлось написание и отладка компонента медицинской информационной системы, предназначенный для автоматизации учета данных, безопасного хранения данных. Весь функционал выполнен в полной мере и в качестве, соответствующем требованиям, предъявляемыми к современному программному обеспечению.

Система представлена WEB-сервисом, в котором сотрудники регистратуры санатория имеют возможность добавления, изменения и удаления информации о пациентах. Реализован удобный поиск информации. В системе реализована возможность оформления расписания лечения для пациентов, а также учет его исполнения. Система помогает в планировании работы врачей, кабинетов и медицинского оборудования.

В приложении реализовано разграничение доступа. Работник регистратуры имеет доступ только с записи и учету пациентов. Врачи имеют доступ к расписанию лечения пациентов и расписанию работы кабинетов и медицинского оборудования на редактирование, к другим разделам только на чтение. Руководство санатория имеет доступ ко всем разделам системы и имеет возможность изменять любые данные. Доступ к любому разделу системы возможен только после аутентификации.

В ходе выполнения дипломного проектирования были продемонстрированы навыки, полученные в ходе обучения в ПГУ. Такие как: умение программировать на объектно-ориентированном языке программирования, умение создавать web-приложения, умение проектировать базы-данных, умение тестировать программные продукты, умение писать документацию, соответствующую требованиям ЕСПД.

Работа выполнена в полном обьёме в соответствии с ТЗ в установленные сроки.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Д. Румянцев. Сам себе Web-программист. Практикум создания качественного Web-сайта. – М.: ИНФРА-М, 2001. – 360 с.

2. Фримен Э., Робсон Э. Изучаем программирование на JavaScript. – СПб.: Питер, 2015. – 640 с.

3. Закас Н. JavaScript для профессиональных веб-разработчиков / [Пер. с англ. А. Лютича]. – СПб.: Питер, 2011. – 960 с.

4. Бенедетти Р., Крэнли Р. Изучаем работу с jQuery. – СПб.: Питер, 2012. – 528 с.

5. Коннолли Т., Бегг К., Страчан А. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание.: Пер. с англ. – М.: Издательский дом "Вильяме", 2003. – 1440 с.

6. Фултон Х., The Ruby Way. – М.: Питер, 2012 – 348 с.

7. Хартл М, Ruby on Rails Tutorial. – М.: Питер, 2011 – 514 с.

8. Ruby on Rails API [Электронный ресурс]. – Электрон.дан. – URL: http://api.rubyonrails.org (дата обращения 2.04.2017).

9. Ruby on Rails по-русски [Электронный ресурс]. – Электрон.дан. – URL: http://rusrails.ru/ (дата обращения 16.04.2017).

10. Ruby on Rails [Электронный ресурс]. – Электрон.дан. – URL: https://ru.wikipedia.org/wiki/Ruby_on_Rails (дата обращения 20.04.2017);

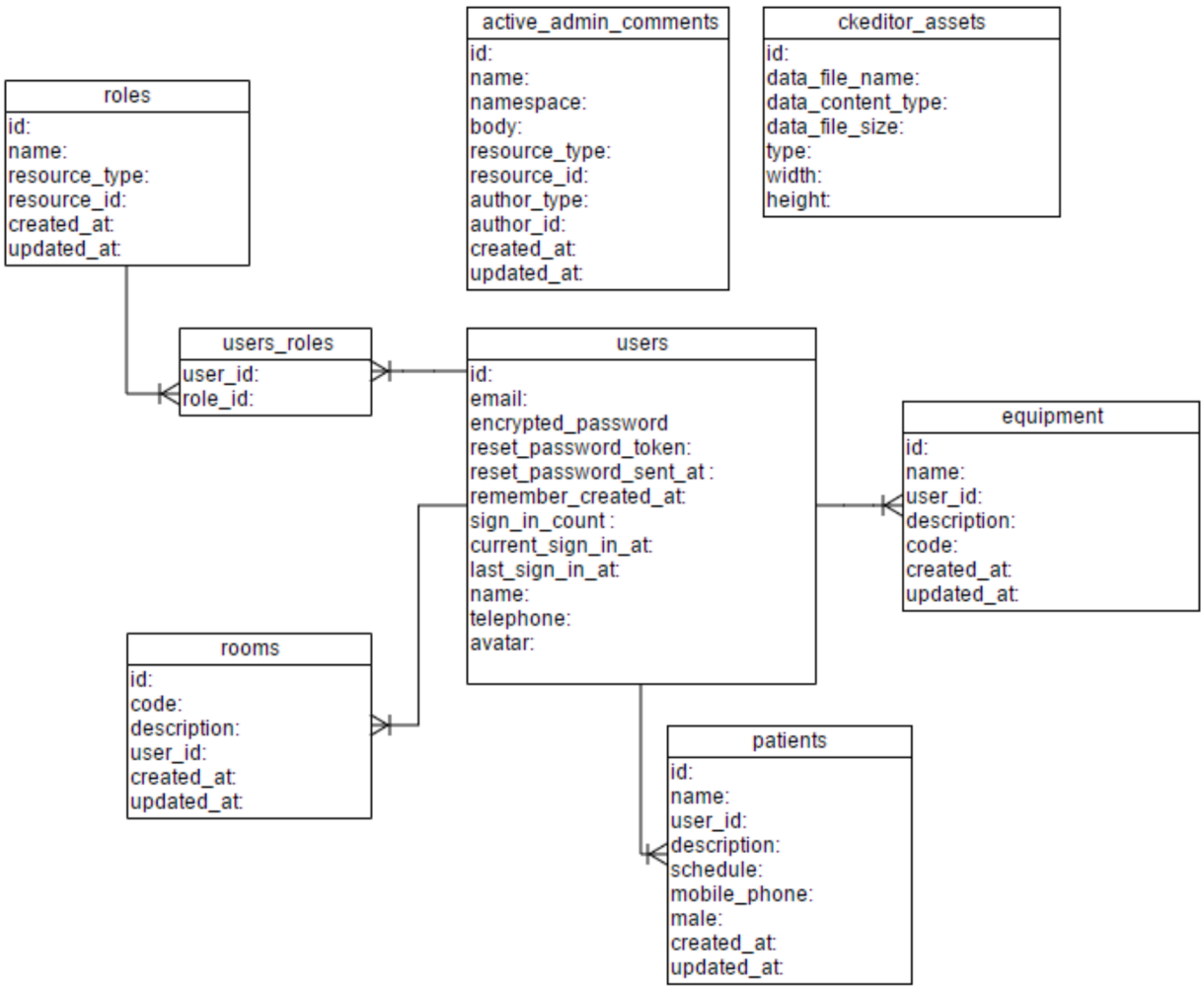
11. PostgreSQL [Электронный ресурс]. – Электрон.дан. – URL: https://ru.wikipedia.org/wiki/PostgreSQL(дата обращения 20.04.2017);

12. Ruby [Электронный ресурс]. – Электрон.дан. – URL: https://ru.wikipedia.org/wiki/Ruby (дата обращения 20.04.2017)

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А. СХЕМЫ

Модель данных.



ПРИЛОЖЕНИЕ Б. ЛИСТИНГИ

Приложение Б1. Модели

Файл – ability.rb

```
          class Ability
        include CanCan::Ability

        def initialize(user)
          user ||= User.new # guest user (not logged in)
          if user.has_role? :chief_medical_officer#управление
может все
            can :manage, :all
          else
            #так как нет прямой иерархии в ролях, роли дополняют
друг друга и нужно проверить все
            if user.has_role? :doctor
              can :read, :all
              can :manage, Patient
              can :manage, Room
              can :manage, Equipment
            end
            if user.has_role? :register_office
              can :read, :all
              can :manage, Patient
            end
            if user.has_role? :employee
              can :read, :all
            end
          end
        end
      end
```

Файл – equipment.rb

```
      class Equipment < ApplicationRecord
        belongs_to :user
        validates_presence_of :name, :user_id, :description, :code
      end
```

Файл – patient.rb

```
      class Patient < ApplicationRecord
        belongs_to :user
```

```
      validates_presence_of :user_id, :name, :birthday, :male,
:departure

      validate :validate_doctor

      private
      def validate_doctor

        errors.add(:user, ' только врач может быть привязан к
пациенту')  if !user.blank? && !user.has_role?(:doctor)
      end
    end
```

Файл – role.rb

```
    class Role < ApplicationRecord
      has_and_belongs_to_many :users, :join_table =>
:users_roles

      belongs_to :resource,
                 :polymorphic => true,
                 :optional => true

      validates :resource_type,
                :inclusion => {  :in => Rolify.resource_types },
                :allow_nil => true

      scopify
    end
```

Файл – room.rb

```
    class Room < ApplicationRecord
      belongs_to :user
      validates_presence_of :user_id, :code
    end
```

Файл – user.rb

```
    class User < ApplicationRecord
      mount_uploader :avatar, AvatarUploader
      rolify
      devise :database_authenticatable,
             :recoverable, :rememberable, :trackable,
:validatable
```

```
    belongs_to :role
    delegate :can?, :cannot?, to: :ability
    after_create :assign_default_role
    validates_uniqueness_of :email

    def assign_default_role
      add_role :employee
    end
  end
```

Приложение Б2. Контроллеры

Файл – equipment.rb

```
  ActiveAdmin.register Equipment do
    menu label: 'Оборудование'
    permit_params :name, :user_id, :description, :code

    controller do
      def action_methods
        if current_user.has_role?(:chief_medical_officer) ||
current_user.has_role?(:doctor)
          super
        else
          #если это не главный врач, то он может только
посмотреть инфу о сотруднике
          super - [ 'edit', 'destroy', 'new']
        end
      end
    end

    index do
      selectable_column
      id_column
      column :code
      column :name
      column 'Ответственный', :user
      actions
    end

    show do
      attributes_table do
```

```
            row :id
            row :name, label: 'Название'
            row :code, label: 'Инвентарный номер'
            row :created_at, label: 'Номр телефона'
            row :user, label: 'Ответственный'
            row 'Описание и расписание использования' do |user|
              user.description.html_safe
            end
          end
        end

        form do |f|
          f.inputs 'Оборудование' do
            f.input :user, as: :select, include_blank: false,
                    label: 'Ответственный', collection:
User.all.map{ |u| [  u.name, u.id ]}
            f.input :name, label: 'Название'
            f.input :code, label: 'Инвентаризационный номер'
            f.input :description, as: :ckeditor, label: 'Описание
и расписание использования'
          end
          f.actions
        end

      end
```

Файл – patient.rb

```
      ActiveAdmin.register Patient do
        menu label: 'Пациенты'
        permit_params :name, :birthday, :male, :departure,
:mobile_phone, :description, :schedule, :user_id

        controller do
          def action_methods
            if current_user.has_role?(:chief_medical_officer) ||
                current_user.has_role?(:doctor) ||
current_user.has_role?( :register_office)
              super
            else
              #если это не главный врач, то он может только
посмотреть инфу о сотруднике
              super - [ 'edit', 'destroy', 'new']
            end
          end
```

```
        end

        show do
          attributes_table do
            row :id
            row :name, label: 'Имя'
            row :birthday, label: 'Дата рождения'
            row :male, label: 'мужчина'
            row :departure, label: 'Дата выписки'
            row :mobile_phone, label: 'Номр телефона'
            row :created_at, label: 'Номр телефона'
            row :user, label: 'Лечащий врач'
            row 'Описание(амбулаторная книга)' do |user|
              user.description.html_safe
            end
            row 'Расписание лечения' do |user|
              user.schedule.html_safe
            end
          end
        end

        form do |f|
          f.inputs 'Пациент' do
            f.input :user, as: :select, include_blank: false,
                    label: 'Лечащий врач', collection:
User.all.joins(:roles)
                        .where('roles.name = ?', :doctor).map{ |u| [
u.name, u.id ]}
            f.input :name, label: 'Имя'
            f.input :birthday, as: :date_time_picker, label: 'Дата
рождения'
            f.input :male, label: 'мужчина'
            f.input :departure, as: :date_time_picker, label:
'Дата выписки'
            f.input :mobile_phone, label: 'Номр телефона'
            f.input :description, as: :ckeditor, label:
'Описание(амбулаторная книга)'
            f.input :schedule, as: :ckeditor, label: 'Расписание
лечения'
          end
          f.actions
        end

      end
```

Файл – role.rb

```
ActiveAdmin.register Role do
  menu label: 'Роль'
  permit_params :name, :resource_id

  controller do
    def action_methods
      if current_user.has_role?(:chief_medical_officer)
        super
      else
        #если это не главный врач, то он может только
посмотреть инфу о сотруднике
        super - [ 'edit', 'destroy', 'new']
      end
    end
  end
```

Файл – room.rb

```
ActiveAdmin.register Room do
  menu label: 'Кабинеты'
  permit_params :user_id, :code, :description

  controller do
    def action_methods
      if current_user.has_role?(:chief_medical_officer) ||
current_user.has_role?(:doctor)
        super
      else
        #если это не главный врач, то он может только
посмотреть инфу о сотруднике
        super - [ 'edit', 'destroy', 'new']
      end
    end
  end

  index do
    selectable_column
    id_column
    column :code
    column 'Ответственный', :user
    actions
  end

  show do
```

```
        attributes_table do
          row :id
          row :code, label: 'Инвентарный номер'
          row :created_at, label: 'Номр телефона'
          row :user, label: 'Ответственный'
          row 'Описание и расписание использования' do |user|
            user.description.html_safe
          end
        end
      end

      form do |f|
        f.inputs 'Кабинет' do
          f.input :user, as: :select, include_blank: false,
                  label: 'Ответственный', collection:
User.all.map{ |u| [  u.name, u.id ]}
          f.input :code, label: 'Номр кабинета'
          f.input :description, as: :ckeditor, label: 'Описание
и расписание использования'
        end
        f.actions
      end

    end
```

Файл – user.rb

```
    ActiveAdmin.register User do
      menu label: 'Сотрудники'
      permit_params :email, :password, :password_confirmation,
:role_ids, :avatar

      controller do
        def action_methods
          if current_user.has_role?(:chief_medical_officer)
            super
          else
            #если это не главный врач, то он может только
посмотреть инфу о сотруднике
            super - [ 'edit', 'destroy', 'new']
          end
        end
      end

      controller do
```

```
          def update
            if params[ :user][ :password] .blank?
              params[ :user] .delete('password')
              params[ :user] .delete('password_confirmation')
            end
            super
          end

        end

        scope 'Доктора' do |u|
          u.joins(:roles).where('roles.name = ?', :doctor)
        end
        scope 'Управляющий персонал' do |u|
          u.joins(:roles).where('roles.name = ?',
:chief_medical_officer)
        end
        scope 'Сотрудники регистратуры' do |u|
          u.joins(:roles).where('roles.name = ?',
:register_office)
        end
        scope 'Все сотрудники' do |u|
          u
        end

        index do
          selectable_column
          id_column
          column :email
          column :current_sign_in_at
          column :sign_in_count
          column :created_at
          column :roles do |user|
            user.roles.collect { |c| c.name.capitalize
} .to_sentence
          end
          column :avatar do |user|
            image_tag(user.avatar.url)
          end
          actions
        end

        show do
          default_main_content do
            row :roles do |user|
```

```
            user.roles.collect { |r| r.name.capitalize
} .to_sentence
          end
          row :avatar do |user|
            image_tag(user.avatar.url)
          end
        end
      end

      form do |f|
        f.inputs 'Пользователь' do
          f.input :email
          f.input :avatar
          f.input :name
          f.input :password
          f.input :password_confirmation
          f.input :roles, collection: Role.global,
                  label_method: lambda { |el| t
"simple_form.options.user.roles.#{ el.name} " } ,
                  include_blank: false
        end
        f.actions
      end

    end
```

Приложение Б3. Представление

Файл – application.html.erb

```
    <!DOCTYPE html>
    <html>
      <head>
        <title>Hospital</title>
        <%= csrf_meta_tags %>

        <%= stylesheet_link_tag    'application', media: 'all',
'data-turbolinks-track': 'reload' %>
        <%= javascript_include_tag 'application', 'data-
turbolinks-track': 'reload' %>
      </head>

      <body>
```

```
    <%= yield %>
  </body>
</html>
```

Приложение Б4. Миграции

Файл – 20170513123122_devise_create_users.rb

```
class DeviseCreateUsers < ActiveRecord::Migration[5.0]
  def change
    create_table :users do |t|
      ## Database authenticatable
      t.string :email,              null: false, default: ""
      t.string :encrypted_password, null: false, default: ""

      ## Recoverable
      t.string   :reset_password_token
      t.datetime :reset_password_sent_at

      ## Rememberable
      t.datetime :remember_created_at

      ## Trackable
      t.integer  :sign_in_count, default: 0, null: false
      t.datetime :current_sign_in_at
      t.datetime :last_sign_in_at
      t.string   :current_sign_in_ip
      t.string   :last_sign_in_ip


      t.string  :avatar
      t.string  :name

      t.timestamps null: false
    end

    add_index :users, :email,                unique: true
    add_index :users, :reset_password_token, unique: true
  end
end
```

Файл – 20170513123125_create_active_admin_comments.rb

```
class CreateActiveAdminComments <
ActiveRecord::Migration::Current
  def self.up
```

```
      create_table :active_admin_comments do |t|
        t.string :namespace
        t.text   :body
        t.references :resource, polymorphic: true
        t.references :author, polymorphic: true
        t.timestamps
      end
      add_index :active_admin_comments, [ :namespace]

    end

    def self.down
      drop_table :active_admin_comments
    end
  end
```

Файл – 20170513124154_rolify_create_roles.rb

```
  class RolifyCreateRoles < ActiveRecord::Migration
    def change
      create_table(:roles) do |t|
        t.string :name
        t.references :resource, :polymorphic => true

        t.timestamps
      end

      create_table(:users_roles, :id => false) do |t|
        t.references :user
        t.references :role
      end

      add_index(:roles, :name)
      add_index(:roles, [  :name, :resource_type, :resource_id
])
      add_index(:users_roles, [  :user_id, :role_id ])
    end
  end
```

Файл – 20170513135523_create_patients.rb

```
  class CreatePatients < ActiveRecord::Migration[ 5.0]
    def change
      create_table :patients do |t|
        t.string :name, null: false
        t.date    :birthday, null: false
        t.boolean :male, null: false, default: true#false
female
        t.date :departure, null: false#дата выселения
```

```
            t.timestamps
            t.string  :mobile_phone
            t.string :description# любая информация, включая
истории болезни
            t.string :schedule# расписание лечения
            t.references :user#лечащий врач
          end
        end
      end
```

Файл – 20170513141917_create_equipment.rb

```
      class CreateEquipment < ActiveRecord::Migration[ 5.0]
        def change
          create_table :equipment do |t|
            t.string :name, null: false
            t.references :user, null: false# ответственный за
оборудование
            t.string :description, null: false# описание предмета
            t.string :code, null: false #инвентарный номер
            t.timestamps
          end
        end
      end
```

Файл – 20170513143104_create_rooms.rb

```
      class CreateRooms < ActiveRecord::Migration[ 5.0]
        def change
          create_table :rooms do |t|
            t.references :user# ответственный за кабинет
            t.string :code, null: false#номер кабинета
            t.string :description # описание кабинета
            t.timestamps
          end
        end
      end
```

Файл – 20170514211009_create_ckeditor_assets.rb

```
      class CreateCkeditorAssets < ActiveRecord::Migration
        def self.up
          create_table :ckeditor_assets do |t|
            t.string  :data_file_name, null: false
            t.string  :data_content_type
            t.integer :data_file_size
            t.string  :type, limit: 30

            # Uncomment it to save images dimensions, if your need
it
```

```
            t.integer :width
            t.integer :height

            t.timestamps null: false
          end

          add_index :ckeditor_assets, :type
        end

        def self.down
          drop_table :ckeditor_assets
        end
      end
```

Приложение Б5. Создание БД

```
      ActiveRecord::Schema.define(version: 20170514211009) do

        create_table "active_admin_comments", force: :cascade do
|t|
          t.string   "namespace"
          t.text     "body"
          t.string   "resource_type"
          t.integer  "resource_id"
          t.string   "author_type"
          t.integer  "author_id"
          t.datetime "created_at",    null: false
          t.datetime "updated_at",    null: false
          t.index ["author_type", "author_id"], name:
"index_active_admin_comments_on_author_type_and_author_id"
          t.index ["namespace"], name:
"index_active_admin_comments_on_namespace"
          t.index ["resource_type", "resource_id"], name:
"index_active_admin_comments_on_resource_type_and_resource_id"
        end

        create_table "ckeditor_assets", force: :cascade do |t|
          t.string   "data_file_name",               null: false
          t.string   "data_content_type"
          t.integer  "data_file_size"
          t.string   "type",              limit: 30
          t.integer  "width"
```

```
    t.integer  "height"
    t.datetime "created_at",                   null: false
    t.datetime "updated_at",                   null: false
    t.index ["type"], name: "index_ckeditor_assets_on_type"
  end

  create_table "equipment", force: :cascade do |t|
    t.string   "name",        null: false
    t.integer  "user_id",     null: false
    t.string   "description", null: false
    t.string   "code",        null: false
    t.datetime "created_at",  null: false
    t.datetime "updated_at",  null: false
    t.index ["user_id"], name: "index_equipment_on_user_id"
  end

  create_table "patients", force: :cascade do |t|
    t.string   "name",                         null: false
    t.date     "birthday",                     null: false
    t.boolean  "male",         default: true, null: false
    t.date     "departure",                    null: false
    t.datetime "created_at",                   null: false
    t.datetime "updated_at",                   null: false
    t.string   "mobile_phone"
    t.string   "description"
    t.string   "schedule"
    t.integer  "user_id"
    t.index ["user_id"], name: "index_patients_on_user_id"
  end

  create_table "roles", force: :cascade do |t|
    t.string   "name"
    t.string   "resource_type"
    t.integer  "resource_id"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.index ["name", "resource_type", "resource_id"], name:
"index_roles_on_name_and_resource_type_and_resource_id"
    t.index ["name"], name: "index_roles_on_name"
  end

  create_table "rooms", force: :cascade do |t|
    t.integer  "user_id"
    t.string   "code",        null: false
    t.string   "description"
```

```
      t.datetime "created_at",  null: false
      t.datetime "updated_at",  null: false
      t.index ["user_id"], name: "index_rooms_on_user_id"
    end

    create_table "users", force: :cascade do |t|
      t.string   "email",                  default: "", null:
false
      t.string   "encrypted_password",     default: "", null:
false
      t.string   "reset_password_token"
      t.datetime "reset_password_sent_at"
      t.datetime "remember_created_at"
      t.integer  "sign_in_count",          default: 0,  null:
false
      t.datetime "current_sign_in_at"
      t.datetime "last_sign_in_at"
      t.string   "current_sign_in_ip"
      t.string   "last_sign_in_ip"
      t.string   "avatar"
      t.string   "name"
      t.datetime "created_at",                          null:
false
      t.datetime "updated_at",                          null:
false
      t.index ["email"], name: "index_users_on_email", unique:
true
      t.index ["reset_password_token"], name:
"index_users_on_reset_password_token", unique: true
    end

    create_table "users_roles", id: false, force: :cascade do
|t|
      t.integer "user_id"
      t.integer "role_id"
      t.index ["user_id", "role_id"], name:
"index_users_roles_on_user_id_and_role_id"
    end

  end
```